RENPAR'16 / CFSE'4 / SympAAA'2005 / Journées Composants
Le Croisic, France, 5 au 8 avril 2005

Parallélisation d'algorithmes irréguliers d'analyse d'images dans l'environnement ANET

Bertrand Ducourthial*, Nicolas Sicard**

*Laboratoire Heudiasyc (UMR CNRS 6599)
Université de Technologie de Compiègne
BP 20529 F-60205 Compiègne CEDEX, France.
Email : Bertrand.Ducourthial@hds.utc.fr

**Institut d'Électronique Fondamentale (UMR CNRS 8622)
Université de Paris-Sud, bât. 220
F-91405 Orsay CEDEX, France
Email : sicard@ief.u-psud.fr
Actuellement à l'IUT de Montreuil, F-93100 Montreuil, France

Résumé

Les algorithmes d'analyse d'images présentent généralement une grande irrégularité des données, en terme de tailles, de structures et de traitements induits. Leur parallélisation s'en trouve donc compliquée. Or, ces algorithmes manipulent d'abord des tableaux de pixels, puis des structures de haut niveau : les régions irrégulières de l'image. Dans cet article, nous discutons de l'intérêt d'exploiter ces structures de haut niveau pour guider la parallélisation.

Notre étude porte sur la parallélisation de l'environnement de programmation ANET dédié à l'analyse d'images. Cet environnement favorise un prototypage rapide des algorithmes grâce à des structures basées sur des graphes, et des primitives de calcul puissantes. Mais les mécanismes internes de ces primitives sont difficiles à paralléliser. Notre objectif est de fournir une parallélisation implicite de l'environnement ANET sans modification de l'interface de programmation et sans intervention du programmeur.

Mots-clés : bibliothèque de programmation parallèle, irrégularité, analyse d'image, scans.

---

1. Introduction

1.1. Analyse d'image et parallélisme

Par nature, les problèmes d'analyse d'image se confondent souvent avec ceux du parallélisme. En effet, l'analyse d'image est caractérisée (i) par une grande quantité de données habituellement structurées sous la forme de tableaux à deux dimensions de pixels, (ii) par de nombreux calculs simples et répétitifs au niveau des pixels (pré-traitements) et ensuite (iii) par des manipulations de données plus complexes et plus dynamiques (travail sur les régions de l'image). Le parallélisme de données représente une approche efficace pour les pré-traitements, puisqu'il fournit les outils pour une manipulation aisée de grandes quantités de données au travers d'un modèle de programmation simple. Mais après la phase de traitement au niveau des pixels, les algorithmes manipulent des structures irrégulières, telles que des régions, ou des contours dans

l'image. Le parallélisme de données doit alors être enrichi pour permettre la représentation et la manipulation efficace de tels objets. À côté des langages data-parallèles standard, des structures basées sur des graphes semblent plus appropriées pour l'analyse d'images [4, 7, 11].

1.2. Mise en œuvre

L'analyse d'image a donné lieu à de nombreuses propositions en terme d'architectures parallèles spécialisées. Mais les progrès importants dans le développement des processeurs génériques favorise l'émergence d'environnements pour l'analyse d'image sur réseaux de stations de travail ou bien sur stations multiprocesseurs (SMP). Ces environnements reposent généralement sur des bibliothèques de programmation, elles-mêmes basées sur des langages standard (C, C++...). Elles autorisent ainsi une certaine souplesse d'utilisation, et sont disponibles sur un grand nombre de plateformes [12, 13, 19].

La parallélisation de tels environnements se heurte à l'irrégularité des traitements [10, 17], qui peut conduire à un déséquilibre dans la répartition des calculs sur les processeurs et donc à de mauvaises performances. De façon générale, l'irrégularité des calculs induits par un algorithme est lié à la complexité de leur ordonnancement. Dans le cas de l'analyse d'image, cette complexité est liée à la nature de l'image traitée, et ne peut être prédite : le graphe de précédence est inconnu avant l'analyse de l'image (comment en effet savoir que la segmentation du ciel uni sera plus rapide que celle du paysage fourni avant l'analyse de l'image ?).

Pour offrir de bonnes performances, les environnements de programmation disposent d'optimisations particulières pour chaque architecture visée, notamment lorsqu'elles intègrent directement des mécanismes d'ordonnancement ou d'équilibre de charge [12]. Mais la programmation dans de tels environnement n'est pas toujours aisée : la conception, la lecture et le déboguage d'un programme peuvent être complexes. Des primitives de haut-niveau voire des macrocommandes sont alors utiles pour simplifier l'écriture des algorithmes. Une ou plusieurs phases de traduction vont permettre de transcrire ces instructions en une suite d'instructions de base, éventuellement optimisées pour l'architecture visée [13].

1.3. Notre contribution

L'environnement ANET permet un prototypage rapide des algorithmes d'analyse d'image. Il repose sur une extension du parallélisme de données (*les réseaux associatifs*), incluant la manipulation de données irrégulières au sein des images à l'aide de graphes, et des primitives de calculs de haut niveau sur ces structures (*associations*) [6, 7]. Ainsi, instanciée avec l'opérateur adéquat, une seule primitive permet de calculer une transformée en distance dans chaque région de l'image. En quelque sorte, les facilités qu'offre le parallélisme de données pour travailler sur les pixels sont étendues aux régions pourtant très irrégulières (en taille, forme...).

Dans cet article, nous présentons la parallélisation d'ANET. Notre objectif est d'offrir une parallélisation implicite sans modification de l'interface de programmation. Une fois détectées, les informations relatives aux régions devraient guider la parallélisation en terme d'équilibrage de charge notamment (§3). En fait, nous montrons que, dans les algorithmes d'analyse d'images, ces structures sont par trop irrégulières et dynamiques pour aider la parallélisation, notamment lorsque les architectures sous-jacentes et leur hiérarchie mémoire sont performantes sur données régulières. Nous proposons alors une généralisation du calcul de la transformée en distance par balayages réguliers permettant de paralléliser les primitives *associations* tout en satisfaisant les contraintes des caches (§4). Nous complétons la parallélisation de certaines primitives en proposant une parallélisation d'une réduction de graphe (§5). Nous présentons ensuite les résultats obtenus sur des applications complexes (§6). Nous commençons par présenter ANET.

2. L'environnement ANET

L'environnement ANET repose sur un module de calcul intensif écrit en C, et sur une interface de programmation écrite en C++, indépendante de la plate-forme d'exécution [6, 20].

2.1. Structures de données

La structure de base dans ANET est un graphe orienté symétrique $G(V, E)$, où V désigne l'ensemble des pixels, et E l'ensemble des connexions unidirectionnelles entre pixels voisins (le fait de disposer d'un graphe orienté symétrique permet de considérer des sous-graphes tels que des arborescences orientées de la racine vers les feuilles par exemple). Dans la suite, nous considérons une *maille*, c'est-à-dire une grille à deux dimensions avec les diagonales. L'objet de base `anet` code cette topologie et différentes informations relatives à l'environnement.

Une variable parallèle P est un $|V|$-uplet composé d'une variable locale par sommet de la maille. La valeur de P sur le sommet v est notée $P[v]$. L'objet de base `pvar` représente une variable parallèle, qui peut être spécialisée afin de s'adapter aux différents types élémentaires (`pvarInt`, `pvarFloat`...). Une image est codée par une pvar de pixels.

Une région est un sous-ensemble de pixels connectés au sein d'une image. Une segmentation d'image est une partition d'une image en plusieurs régions. Elle est codée par un graphe couvrant H de la maille G, qui admet une composante connexe par région. Un tel graphe couvrant peut être construit en désactivant les arcs de G qui relient les pixels voisins n'appartenant pas à la même composante. Les arcs incidents activés (resp. désactivés) sont notés par un 1 (resp. 0) dans un champ de bits appelé *masque* au niveau de chaque sommet. Un graphe couvrant H peut donc être codé entièrement par une pvar de masques (objet `subnet`). Dans la maille 8-connexe, un masque est représenté par un octet puisque chaque sommet a huit voisins. Les objets `subnets` définissent ce qu'on appelle un *graphe d'exécution*, c'est-à-dire un graphe de relations inter-pixels sur lequel sont appliquées les primitives associations (§2.2).

Il n'est pas toujours intéressant de représenter une région par l'ensemble de ses pixels. En effet, cela complique le maniement des relations inter-régions, et entraîne des calculs redondants lorsque la valeur représentative d'une région aurait à être dupliquée sur chacun de ses pixels. ANET offre donc pour les régions les mêmes dispositifs que pour les pixels, au travers des *graphes virtuels* : graphes d'exécution (objet `vnet`), sous-graphes d'exécution (objet `vsubnet`), variables parallèles avec une valeur par région (objet `vpvar`).

Ainsi, ANET permet de manipuler aisément des objets aussi divers que des pixels, des relations inter-pixels, des contours, des régions... Toutes ces données, qu'elles soient régulières ou non, sont codées de manière unifiée par des pvars.

2.2. Primitives

ANET dispose des primitives de calcul data-parallèles classiques sur les pvars (cf. figure 2.3) : *element-wise*, réductions, *etc.* ANET dispose également de diverses primitives permettant de construire et de manipuler les sous-graphes, en modifiant les valeurs des masques des `subnets`. Ces bits peuvent être calculés en évaluant un prédicat portant sur les valeurs de pixels voisins dans des pvars données. Ainsi l'opération `LinkWithEquals` établit un lien (par activation d'un arc) entre deux sommets dans un `subnet` lorsque leurs valeurs sont égales. Cette primitive permet par exemple de construire des régions homogènes dans une image (cf. §2.3).

Les *associations* sont des primitives spécifiques, combinant des *calculs locaux*. Étant donné une pvar P et un sous-graphe $H \subset G$, un *calcul local* sur le sommet v consiste à combiner la valeur $P[v]$ avec toutes les valeurs $P[u]$ tel que l'arc (u, v) appartienne au graphe H. Cette opération produit une nouvelle valeur pour le sommet v.

- Une $\oplus$-*step-association*(P, H) consiste à appliquer de façon synchrone un tel calcul local sur chaque sommet de la maille. Cela revient à appliquer un masque 3×3 sur l'image.
- Une $\oplus$-*direct-association*(P, H) consiste à itérer des calculs locaux asynchrones sur chaque sommet v du graphe H avec l'opérateur $\oplus$ jusqu'à ce qu'aucune nouvelle valeur locale ne soit produite. L'opérateur $\oplus$ et/ou le sous-graphe H doivent vérifier un certain nombre de conditions pour assurer la convergence des calculs [5]. Ces conditions sont remplies par les opérateurs tels que le minimum, qui sont associatifs, commutatifs et idempotents (*s-opérateurs*). Elles le sont également par une classe plus large d'opérateurs (les r-opérateurs idempotents) tels que $\mathtt{minc}$, défini par $\mathtt{minc}(x, y) = \mathtt{min}(x, y + 1)$. Grâce à la direct-association, cet opérateur fournit la distance de chaque pixel depuis un pixel racine, ce qui permet de calculer une transformée en distance par exemple [5].
- La primitive $\oplus$-*association* est réservée aux opérateurs binaires $\oplus$ associatifs, commutatifs mais non-idempotents tels que l'addition. Étant donné une pvar P et un sous-graphe $H \subset G$, une $\oplus$-*association*(P,H) consiste à combiner la valeur $P[v]$ avec les valeurs de tous les prédécesseurs de v dans H. Par exemple, une plus-association appliquée sur des ensembles connexes de sommets (des régions) calcule en chaque pixel la somme des valeurs de tous les pixels de sa région. Si tous les éléments de P valent 1, alors l'association calcule le nombre de pixels dans chaque région (cf. §2.3).

De façon générale, il existe une version spécialisée de chaque primitive pour chaque opérateur. On dispose ainsi des primitives `MinStepAssoc`, `PlusAssoc`, `MincDirectAssoc`, *etc.* Les types de données et les opérateurs sont indiqués à la compilation, afin de produire davantages d'optimisations et ainsi d'obtenir de meilleures performances.

2.3. Exemple

L'exemple suivant illustre la manière de calculer la surface de la plus grande région homogène (noire) dans une image binaire dans ANET.

```
1 Subnet sRegions = MASK(0);
2 PvarInt pSurface = 0;
3 WHERE(pImage == BLACK)
4     sRegions.LinkWithEquals(pImage, pImage);
5     pSurface.PlusAssoc(1,sRegions);
6     int maxSurface = pSurface.GlobalMax();
7 ENDWHERE;
```

L'image binaire est donnée par la pvar `pImage`. Après l'initialisation des variables parallèles (lignes 1 et 2), un contexte d'exécution permet de ne faire intervenir dans le calcul que les pixels de couleur noire (entre les lignes 3 et 7). D'abord les pixels noirs adjacents sont connectés entre eux pour former le graphe d'exécution `sRegions` (ligne 4). Puis, grâce à l'addition, l'association calcule les surfaces des différentes régions à partir d'une pvar constituée uniquement de 1, et diffuse ces valeurs sur tous les pixels (ligne 5). Enfin, une réduction globale par l'opérateur maximum (ligne 6) retourne la plus grande des surfaces calculées (`maxSurface`).

3. Parallélisation basée sur les données

ANET constitue une extension data-parallèle du C++ adaptée à l'analyse d'images. Les manipulations de pixels comme celles des régions sont exprimées aisément sous la forme de traitements data-parallèles. Notre étude porte sur la parallélisation des exécutions des programmes écrits avec ANET sans modification de son API, dans le but de disposer à la fois d'un outil de prototypage rapide et d'un environnement d'exécution performant.

3.1. Problématique

Les primitives purement data-parallèles se parallélisent aisément avec des méthodes classiques de distribution en blocs. Mais la parallélisation des primitives association (qui constituent l'originalité et l'intérêt d'ANET) est plus difficile. Inspirées d'une architecture SIMD asynchrone [8], elles procurent un gain substantiel également sur mono-processeur en évitant de longues itérations (phénomène de relaxation asynchrone). Toutefois, les architectures à usage général ne sont pas particulièrement adaptées à leur exécution.

Ces primitives sont appliquées sur des graphes d'exécution (`subnets`) irréguliers, généralement constitués de plusieurs composantes fortement connexes (régions, contours, *etc.*). Souvent même, ces ensembles de pixels ne sont pas connectés entre eux, ce qui élimine toute dépendance de données. Dès lors, une parallélisation par distribution de ces ensembles sur les différentes unités de traitement semble constituer une approche naturelle pour obtenir de meilleurs temps de calcul. Nous résumons ici plusieurs expérimentations dans ce sens [20].

3.2. Balayage sur les régions en mémoire distribuée

Nous avons étudié la faisabilité d'une distribution des calculs sur régions sur des stations de travail connectées à un réseau rapide. La structure du graphe d'exécution est d'abord analysée sur la station maître afin de détecter d'éventuelles dépendances de données (connexions entre régions) et de préparer le transfert des données. Puis, sur chaque station et dans chaque région, les calculs sont réalisés à l'aide de balayages séquentiels des pixels, appliqués une fois ou jusqu'à stabilisation des valeurs (selon les cas, cf. §2.2). Enfin les données sont regroupées sur la station maître, et un post-traitement est appliqué.

Nous avons évalué cette approche avec deux stations (Pentium II à 400 MHz) connectés par un réseau Myrinet (débit théorique 1Gb/s), c'est-à-dire dans une situation où le réseau n'apparaît pas comme dissuasif pour exporter des traitements vu la puissance modeste des processeurs. Nous avons utilisé l'environnement de programmation parallèle à grain fin PM2 [14] basé sur les bibliothèques de communication Madeleine et BIP [2] car cet ensemble permet d'excellentes performances sur ce type de plateforme. Les balayages ont été exécutés à distance avec des *Lighweight Remote Procedure Call*. Enfin, pour éviter tout surcoût liés aux dépendances de données, nous avons utilisé des graphes d'exécution dépourvus de connexions inter-régions.

Malgré ces optimisations, des résultats satisfaisants n'ont été obtenus que dans le cas de traitements lourds sur les régions, c'est-à-dire lorsque l'opérateur à appliquer aux pixels était suffisamment complexe et lorsque plusieurs itérations étaient nécessaires. Dans les autres cas, les pré- et post-traitements nécessaires aux exécutions à distance limitaient toute accélération globale. Par ailleurs, lorsque les régions sont peu nombreuses et de grande taille, il n'est pas possible d'assurer une bonne répartition des données sur les différents processeurs. De façon générale, les cas favorables ne sont pas majoritaires dans les applications d'analyse d'image.

3.3. Balayage sur les régions en mémoire partagée

Pour éliminer les pré- et post-traitements ainsi que les transferts de données, nous avons pratiqué des expérimentations similaires sur des plates-formes SMP : un PowerMacintosh (deux PowerPC 970 à 2GHz) et une station Sun Fire 880 (8 processeurs UltraSparc III à 750MHz). Les associations sont calculées au moyen d'un *thread* par région, chargé de balayer les pixels. Des résultats corrects ont été obtenus eu égard à l'irrégularité des traitements. Ainsi, une transformée en distance dans chaque région réalisée avec une `MincDirectAssoc` permet d'obtenir une accélération de 3 sur 8 processeurs (64 régions irrégulières, un pixel racine par région).

Mais cette approche est pénalisée par la nécessité d'analyser le graphe d'exécution (`subnet`) afin d'y détecter les composantes fortement connexes (régions). Ce pré-traitement est assuré

par un parcours du graphe qui présente deux inconvénients. Premièrement, c'est un algorithme essentiellement séquentiel, dont le coût est trop important comparativement au calcul d'une association simple. Deuxièmement, les pixels se retrouvent rangés selon l'ordre de parcours du graphe (irrégulier), qui d'une part n'est pas optimal pour la convergence des associations, et d'autre part ne permet pas d'exploiter les caches. Une optimisation de l'ordre des pixels améliore certes les choses, mais augmente le surcoût du prétraitement séquentiel.

L'approche par balayage des régions est donc viable en mémoire partagée, mais elle reste pénalisée par le pré-traitement. En outre, le parallélisme est faible lorsque le nombre de régions l'est aussi : avec une seule région, il n'y aurait pas d'accélération possible.

3.4. Propagation d'onde

Pour contourner ces contraintes, nous avons expérimenté une approche différente, davantage liée aux valeurs des pixels qu'à leur structuration. Cette méthode s'apparente à la propagation d'une onde à partir de pixels racines que constituent les minima locaux [20]. En effet, tant les s- que les r-opérateurs induisent une relation d'ordre sur les données [5], et donc des minima locaux au sein de l'image. Les pixels racines voient leur valeur se propager (éventuellement en se modifiant) vers leurs voisins. Nous avons expérimenté cette méthode en mémoire partagée, en répartissant les différentes racines sur les *threads*. Chaque *thread* procède à une mise à jour des valeurs d'un entourage rectangulaire à une distance n de la racine à partir des valeurs de l'entourage à distance $n-1$. Ce mode de diffusion est régulier et ne nécessite pas d'analyse préalable du graphe. Les valeurs ne progressent d'un pixel à un autre que s'il existe un lien dans le graphe d'exécution (`subnet`). Le degré de parallélisme potentiel est lié au nombre de racines (souvent nombreuses), et le pré-traitement consiste simplement à détecter les minima locaux, ce qui est aisément parallélisable par balayages simultanés sur plusieurs blocs.

La plus grande régularité des traitements a permis d'obtenir de meilleurs résultats, aussi bien en terme de temps de calcul que d'accélération. Ainsi, une `MincDirectAssoc` nécessite moins de deux fois moins de temps qu'auparavant. En outre, l'accélération est maintenant de 1,7 sur le biprocesseurs (contre 1,4 auparavant, cf. §3.3). Cependant, lorsque les régions ont des formes compliquées, plusieurs propagations d'ondes peuvent s'avérer nécessaires pour obtenir la stabilisation de la `DirectAssoc`. En outre, cette méthode se comporte moins bien en cas de racines trop nombreuses, ou lorsque leur densité est importante dans certaines régions et moins dans d'autres (risque de déséquilibrage de charge).

4. Parallélisation basée sur des balayages réguliers

4.1. Principe

Si la méthode de parallélisation par propagation d'ondes offre des résultats intéressants, elle reste malheureusement inefficace dans certains cas. Nous proposons ici une parallélisation des traitements sur les régions qui élimine ces inconvénients. D'après [16], on sait qu'une transformée en distance (dans différentes métriques) peut être réalisée efficacement à l'aide de balayages réguliers (*scan*). Un balayage consiste à parcourir l'image ligne par ligne. La valeur d'un pixel est calculée à partir des valeurs de ses 4 pixels voisins déjà traités par le même balayage. Un balayage descendant suivi d'un balayage ascendant (appelé *passe*) suffit à calculer toutes les distances. Or, une telle transformation est en réalité une forme de mise en œuvre d'une direct-association sur le graphe d'exécution correspondant à la maille complète non torique. De la même manière, on peut calculer toute `DirectAssoc` avec un s- ou un r-opérateur.

De plus, nous avons montré que lorsque le graphe d'exécution n'est plus la maille complète mais un sous-graphe admettant différentes composantes fortement connexes (régions), cette

méthode permet encore de calculer une `DirectAssoc`, à condition de l'itérer jusqu'à stabilisation des valeurs [20]. Nous avons proposé une méthode permettant de déterminer le nombre d'itérations nécessaires pour un graphe donné. Ainsi, un sous-graphe de la maille ne présentant que des régions convexes nécessitera trois passes (*e.g.*, régions de Voronoï). En fait, dans la majorité des cas rencontrés en analyse d'image, le nombre de passes n'excède pas 5.

Une passe sur l'image ne nécessite aucun pré-traitement, que ce soit l'analyse du graphe d'exécution ou la recherche de minima comme précédemment. En outre, c'est un calcul fortement régulier en terme d'accès aux données, ce qui permet d'exploiter au mieux les hiérarchies mémoires des stations à mémoire partagée. Reste alors à paralléliser le calcul d'une passe sur une image. Il présente des dépendances de données, mais très localisées : un pixel dépend de 4 de ses voisins. En outre, ce qu'il nous faut paralléliser est l'application de plusieurs passes consécutives et non une seule, ce qui ouvre d'autres possibilités.

4.2. Parallélisation des balayages réguliers

Les balayages descendants et ascendants partent de côtés opposés de l'image, et progressent en sens contraire. Ils peuvent être réalisés simultanément à condition de synchroniser leur croisement (figure 1-gauche). On peut donc espérer une accélération proche de deux sur un biprocesseur, et ce sans aucun prétraitement. Par ailleurs, l'existence de plusieurs régions dans l'image permet d'envisager l'exécution de balayages partiels presque indépendants. Des synchronisations entre blocs voisins permettraient d'assurer la convergence des calculs y compris sur les régions à cheval sur deux blocs (figure 1-milieu).

Nous avons expérimenté une combinaison de ces deux approches. L'image est segmentée en blocs de même taille, indexés de 0 à N, correspondant au nombre d'unités de calcul diponibles. Sur chaque bloc, un *thread* réalise alternativement des scans descendants et ascendants (figure 1-droite). Sur les blocs d'indices pairs (resp. impairs), les *threads* commencent par un balayage descendant (resp. ascendant). La progression est alors globalement équivalente au croisement de deux scans sur des blocs de taille double (cf. les flèches en gras sur la figure 1-droite). Des synchronisations sont effectuées à chaque croisement de balayages afin de garantir la meilleure progression des données possible sur l'image. L'avantage de cette méthode est que dans le pire cas (par exemple une valeur minimale à propager entre le premier pixel de l'image, en haut à gauche, et le dernier, en bas à droite) on peut espérer une accélération équivalente à celle de la méthode de croisement de deux balayages.

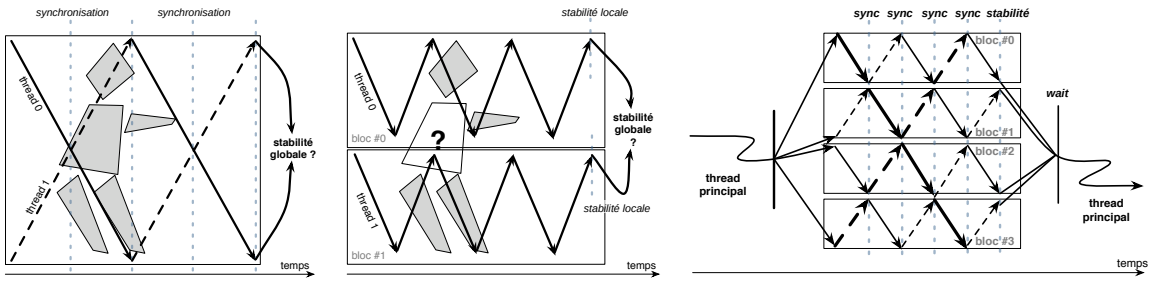

Figure 1: *À gauche, des balayage descendants et ascendants exécutés en parallèle par deux threads différents jusqu'à stabilisation. Au centre, même méthode sur des blocs, avec un thread par bloc, réalisant alternativement un balayage descendant et ascendant. À droite, combinaison de ces deux approches, avec alternance de la direction des balayages, et synchronisation lors des croisements.*

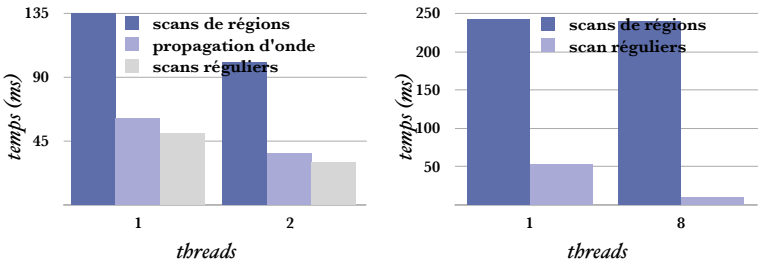

Figure 2: *Temps de calcul de trois méthodes de parallélisation. À gauche, une* `MincDirectAssoc` *(transformée en distance) sur la maille complète avec une racine a été réalisée sur le PowerMacintosh biprocesseurs à 2GHz. À droite, une* `MaxAssoc` *a été exécutée sur la Sun Fire 880 à 8 processeurs à 750MHz.*

4.3. Résultats

Nous résumons ici les résultats obtenus [20]. Cette méthode donne nettement de meilleurs résultats que les précédentes (§3), notamment sur la maille complète (ce qui est fréquemment utilisé dans les algorithmes d'analyse d'image). La figure 2 compare les résultats obtenus avec les méthodes de balayage sur régions (§3.3), de propagation d'ondes (§3.4) et de balayages réguliers sur la maille complète (§4.2).

Les expériences ont montré que l'accélération dépendait de l'opérateur, des valeurs initiales des pixels et de la nature du graphe d'exécution (figure 3). La figure 3-gauche montre les résultats d'une `MaxDirectAssoc` réalisée sur différents types de graphes d'exécution (en nombre de composantes) en fonction du nombre de processeurs utilisés. On confirme ici que les petites régions donnent de meilleurs résultats car elles limitent les dépendances de données inter-blocs. On vérifie également que l'accélération reste supérieure à deux y compris dans le cas le plus défavorable de la maille complète. La figure 3-centre illustre l'impact des valeurs

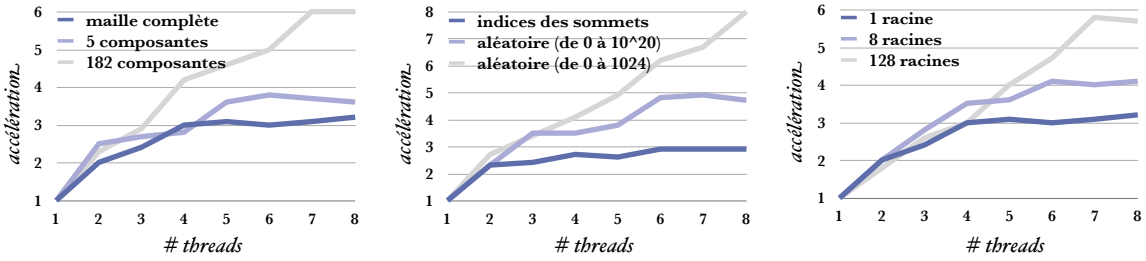

Figure 3: *Évolution des accélérations sur Sun Fire 880 à 8 UltraSparc III à 750MHz. À gauche et au centre :* `MaxDirectAssoc` *avec variation du graphe d'exécution ou des valeurs initiales de la pvar. À droite :* `MincDirectAssoc` *(transformée en distance) avec variation du nombre de racines. Images 1024x1024.*

initiales sur le même calcul que précédemment. Le cas le plus défavorable est obtenu lorsque les valeurs sont initialisées avec les indices des sommets, car le plus grand indice (situé en bas à droite de l'image) devra se propager sur toute l'image ; malgré tout, l'accélération obtenue oscille entre 2 et 3. Plus les valeurs sont regroupées dans un intervalle restreint, plus les propagations sont courtes et plus l'accélération obtenue est grande. Enfin, la figure 3-droite illustre l'impact du nombre de racines sur l'accélération d'une `MincDirectAssoc` (qui calcule

la distance de chaque pixel à la racine la plus proche). Plus les racines sont nombreuses, moins grandes sont les distances qui les séparent, et donc le temps de convergence. On constate ici aussi une accélération minimale y compris dans le cas le plus défavorable.

En conclusion, la parallélisation par balayages réguliers, qui généralise la transformée en distance, est pertinente.

5. Parallélisation de l'analyse du graphe

5.1. Problématique

La parallélisation la plus efficace des associations est donc celle par balayages réguliers et itératifs sur l'ensemble de la maille (§4). Cependant cette méthode ne fonctionne pas pour les opérateurs non idempotents, qui divergeraient en présence d'un circuit dans le graphe d'exécution. Même si les s- et les r-opérateurs idempotents couvrent une bonne partie des associations utilisées en analyse d'images, il est important de paralléliser les autres associations telles que les `PlusAssoc`. Pour cela, le graphe d'exécution est réduit, c'est-à-dire qu'il est transformé en un graphe acyclique en compressant les composantes fortement connexes en un seul sommet. Ce processus permet de gérer les dépendances de données et d'effectuer le calcul avec l'opérateur non idempotent sur chaque composante. L'algorithme de réduction s'apparente à un parcours en profondeur dérivé de [21].

5.2. Principe

Les parcours en profondeur sont réputés intrinsèquement séquentiels [15]. Cependant nous exploitons ici le fait que le graphe d'exécution est un sous-graphe de la maille, ce qui nous permet une parallélisation en blocs. En effet, on peut affirmer que deux sommets dans deux blocs non contigus ne sont pas voisins. En appliquant l'algorithme indépendamment sur chaque bloc, il n'y aura des problèmes qu'aux frontières de blocs. On applique donc l'algorithme séquentiel sur N blocs en parallèle afin d'obtenir N graphes réduits partiels. Pour mettre en commun deux blocs, des réductions partielles sont appliquées sur leurs graphes-résultats reconnectés afin de passer à $N/2$ blocs, puis à $N/4$ et ainsi de suite jusqu'à obtenir le graphe réduit final. Les réductions étant liées au nombre de sommets à explorer, les étapes successives sont de moins en moins coûteuses.

5.3. Résultats

La figure 4 donne les temps de calculs et les accélérations obtenues avec cet algorithme sur la station à 8 processeurs. Le temps de calcul est divisé par 5 en utilisant 8 processeurs, ce qui est un très bon résultat pour ce problème réputé difficile [3, 9]. En outre, on observe que les accélérations sont très peu dépendantes de la nature du graphe (contrairement aux temps de calcul), mais sont très sensibles à la taille de l'image (les accélérations sont très bonnes pour des images de très grande taille). Nous imputons ce phénomène à une dégradation importante des performances de l'algorithme d'exploration récursif avec l'augmentation de la profondeur du graphe.

En conclusion, cette parallélisation de la réduction du graphe est pertinente pour paralléliser les associations instanciées avec des opérateurs non idempotents (addition).

6. Applications

L'environnement ANET permet le prototypage rapide d'algorithmes d'analyse d'images grâce à des structures de données et des primitives adaptées, qui enrichissent le parallélisme de

	# threads			
	1	2	4	8
maille	2194	1590	902	503
	1.0	**1,4**	**2,4**	**4,4**
182 comp.	820	581	300	170
	1.0	**1,4**	**2,7**	**4,8**
>1600 comp.	1116	812	455	260
	1.0	**1,4**	**2,5**	**4,3**

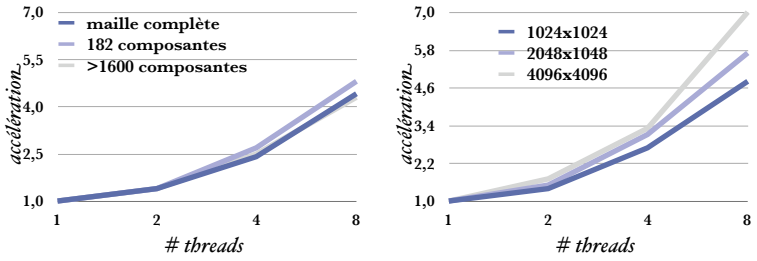

Figure 4: *À gauche, les temps de calcul et les accélérations pour la réduction d'un sous-graphe d'une maille 1024×1024 comportant une seule, plusieurs dizaines (182) ou plusieurs centaines (>1600) de composantes. Au centre, l'évolution des accélérations. À droite, l'influence de la taille de la maille sur l'accélération. Mesures effectuées sur une Sun Fire 880 à 8 UltraSparc III à 750MHz.*

données. Si la parallélisation des traitements data-parallèles classiques ne posait pas de problèmes non résolus, ce n'était pas le cas des primitives dédiées à l'analyse d'images (associations). Au §4, nous avons défini une parallélisation efficace des associations pour les opérateurs idempotents. Au §5, nous avons présenté une parallélisation performante de l'analyse du graphe nécessaire avant tout calcul d'une association avec un opérateur non idempotent. Nous présentons maintenant non plus les résultats des parallélisations de ces primitives prises isolément, mais ceux d'applications complètes d'analyse d'images (voir [20] pour d'autres résultats et pour les codes des programmes).

6.1. Programmes

Les deux programmes dont nous présentons les résultats ici sont la squelettisation et la découpe de Voronoï. La squelettisation permet de réduire les régions d'une image binaire sous forme filaire (1 pixel de large) tout en conservant leur géométrie et leur connexité ; elle est utilisée en reconnaissance des formes (figure 5-gauche). Pour ce problème assez complexe, nous avons adapté l'algorithme de [18], qui est basé sur des transformées en distance. L'algorithme nécessite environ 120 lignes de code C++ avec ANET, ce qui illustre son "expressivité".

La découpe de Voronoï est la première phase d'une segmentation en régions de Voronoï par la méthode du *split and merge* [1]. Il consiste à découper une image en un ensemble de régions homogènes selon un critère donné (figure 5-droite). À partir d'une distribution de *graines*, on calcule les régions de Voronoï [6, 7]. Puis on ajoute une graine dans chaque région non-homogène et on réitère le processus jusqu'à ce que toutes les régions soient homogènes. Écrit dans ANET, ce programme nécessite environ 50 lignes.

Figure 5: *À gauche : squelettisation. À droite : découpe en régions de Voronoï homogènes.*

6.2. Résultats

Pour des raisons de place, nous ne reproduisons ici que les résultats significatifs obtenus sur deux images. D'autres expérimentations sont reportées dans [20]. Nous nous sommes intéressé à l'évolution des temps de calcul et des accélérations des programmes complets en fonction de différents paramètres tels que la nature ou la taille de l'image [20]. Nous avons constaté que le principal paramètre de variation des accélérations est en réalité la taille de l'image. La figure 6 montre les accélérations moyennes obtenues sur différentes tailles d'image pour les deux programmes. Les accélérations sont meilleures sur les grandes images car certaines opérations (*e.g.*, allocations mémoire) restent séquentielles. Ces résultats sont très positifs, eu égard à l'irrégularité imprévisible des calculs induits par ces algorithmes (cf. §1.2). En outre, les formes manipulées sont très irrégulières, et sont constamment modifiées. En conclusion, notre parallélisation implicite d'ANET (§4 et 5) lui permet de conserver son API et ses facilités pour le prototypage tout en offrant des temps de calcul de 3 à 4 fois moindre sur une station à 8 processeurs pour des problèmes complexes.

	# threads				
	1	2	3	4	5
squelet. 512x512	2472	1283	926	736	660
	1.0	**1,9**	**2,7**	**3,4**	**3,7**
découpe 512x512	12745	6821	5359	4409	3845
	1.0	**1,9**	**2,4**	**2,9**	**3,3**

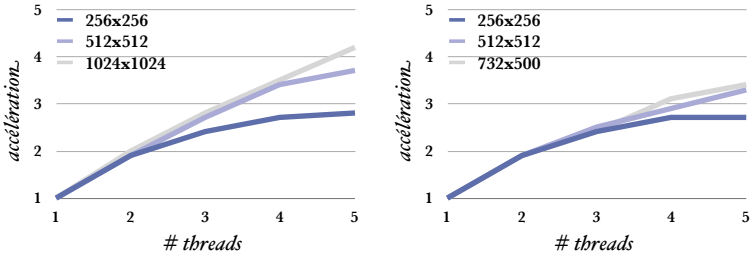

Figure 6: *À gauche, temps de calcul (en ms) et accélérations pour la squeletisation et la découpe de Voronoï sur des images de 512x512 pixels. Évolution de l'accélération en fonction de la taille de l'image pour la squeletisation au centre, et pour la découpe de Voronoï à droite. Mesures effectuées sur une Sun Fire 880 à 8 processeurs UltraSparc III à 750MHz.*

7. Conclusion

L'environnement ANET permet de développer aisément des applications d'analyse d'image sous la forme de programmes data-parallèles. Il est basé sur une bibliothèque C++, qui enrichit le data-parallélisme avec des structures de données (`subnets`) et des primitives (`Assoc`, `DirectAssoc`...) adéquates pour l'analyse d'images. En guise d'illustration, les applications de squeletisation et de segmentation présentées au §6 nécessitent respectivement 120 et 50 lignes de code. Nous souhaitions une parallélisation implicite d'ANET sans modification de son API afin de lui conserver ses facilités de prototypage. La difficulté réside dans les calculs sur les graphes, l'irrégularité imprévisible des traitements, et la manipulation de formes irrégulières (régions), qui sont souvent très dynamiques (*e.g.*, segmentations). Notre contribution peut se résumer ainsi :

- ANET est maintenant un environnement de prototypage offrant de bons temps de calculs vus les problèmes traités : avec 8 processeurs, le temps d'une transformée en distance est divisé par plus de 5, celui d'une squeletisation ou d'une segmentation par plus de 3.
- Nous avons également montré que la parallélisation des programmes d'analyse d'images tire peu parti d'une information de haut niveau sur l'organisation des données (§3). Ceci

est dû au fait que d'une part les structures de données sont trop souvent modifiées comparativement aux traitements qui leurs sont appliqués, et que d'autres part elles sont trop irrégulières pour bénéficier des caches.
– Enfin, nous avons montré que, pour un graphe "quasi" plannaire (*i.e.*, sous-graphe de la maille), l'algorithme de parcours en profondeur réputé intrinsèquement séquentiel peut se paralléliser par blocs : les temps de calcul avec 8 processeurs sont de 4,5 à 7 fois moindre.

En perspectives, nous estimons qu'une analyse des programmes ANET à la manière des préprocesseurs ou des compilateurs parallèles permettraient encore certaines améliorations des temps de calcul.

Bibliographie

1. Ahuja (N.), An (B.) et Shachte (B.). – Image representation using Voronoï tesselation. *CVGIP*, vol. 29, 1985, pp. 286–295.
2. Aumage (O.), Bougé (L.), Méhaut (J.-F.) et Namyst (R.). – Madeleine II : A portable and efficient communication library for high-performance cluster computing. *Parallel Computing*, vol. 28, n° 4, avril 2002, pp. 607–626.
3. Bader (D. A.). – *A practical parallel algorithm for cycle detection in partitioned digraphs*. – Rapport technique n° AHPCC-TR-99-013, Univ. New Mexico, Albuquerque, NM, ECE Dept., 1999.
4. Biancardi (A.) et Mérigot (A.). – Connected component support for image analysis programs. *In : Proc. of the 13th International Conf. of Pattern Recognition (ICPR)*. pp. 3620–624. – IEEE Press, 1996.
5. Ducourthial (B.) et Mérigot (A). – Parallel asynchronous computation for image analysis. *Proceeding IEEE*, vol. 90, n° 7, July 2002, pp. 1218–1229.
6. Ducourthial (B.), Mérigot (A.) et Sicard (N.). – Anet : a programming environment for parallel image analysis. *In : Proc. of IEEE CAMP 2000, Italie, pp. 280–289. – 2000.*
7. Ducourthial (B.) et Sicard (N.). – Nouvelles fonctionnalités pour le parallélisme de données. *Rencontres Francophones du Parallélisme (RENPAR 2001)*, Avril 2001.
8. Dulac (D.), Mohammadi (S.) et Mérigot (A.). – Implementation and evaluation of a parallel architecture using asynchronous communications. *In : Proc. of IEEE CAMP'95, Italie, pp. 106-111. – 1995.*
9. Fleischer (L.), Hendrickson (B.) et Pinar (A.). – On identifying strongly connected components in parallel. *In : IPDPS Workshop*, pp. 505–511. – 2000.
10. Gautier (T.), Roch (J.-L.) et Villard (G.). – Regular versus irregular problems and algorithms. *In : Proceedings Irregular'95*. pp. 1–26. – Springer Verlag, 1995.
11. *IAPTR-TC15 GbR biennial Workshop on Graph-based Representations in Pattern Recognition*.
12. Jamieson (L. H.), Delp (E. J.), Wang (J.), C. (Li C.) et Weil (F. J.). – A software environment for parallel computer vision. *IEEE Computer*, vol. 25(2), 1992, pp. 73–75.
13. Moore (M.-S.) et al. – A model-intergrated program synthesis environment for parallel/real-time image processing. *In : Proceedings of Par. Dist. Methods for Image Processing*, pp. 31–45. – 1997.
14. Namyst (R.) et Méhaut (J.-F.). – PM2 : Parallel multithreaded machine. a computing environment for distributed architectures. *In : Parallel Computing (ParCo '95)*. pp. 279–285. – Elsevier, sept. 1995.
15. Reif (J. H.). – Depth first search is inherently sequential. *IPL*, vol. 20, juin 1985, pp. 229–234.
16. Rosenfeld (A.) et Pfaltz (J.-L.). – Sequential operations in digital picture processing. *Journal of the Association for Computing Machinery*, vol. 13(4), 1966, pp. 471–494.
17. Roucairol (C.). – *STRATAGÈME : une méthodologie de programmation parallèle pour les problèmes non structurés*. – Rapport final, PRISM, Université de Versailles, France, 1996.
18. Sanniti di Baja (G.) et Thiel (E.). – Skeletonization algorithm running on path-based distance map. *Image and Vision Computing*, vol. 14, 1996, pp. 47–54.
19. Seinstra (J.) et Koelma (D.). – User transparency : a fully sequential programming model for efficient data parallel image processing. *Concurrency and Computation : Practice and Experience*, 2004.
20. Sicard (N.). – *Anet : un environnement à parallélisme de données pour l'analyse d'image*. – Thèse de doctorat, Institut d'Electronique Fondamentale, Université Paris-Sud, juillet 2004.
21. Tarjan (R.). – Depth-first search and linear graph algorithms. *SIAM Journal Computing*, vol. 1(2), 1972, pp. 146–160.