Évaluation d'un modèle de calcul parallèle pour l'analyse d'images

A. Mérigot G. Constantinescu B. Ducourthial
D. Dulac S. Guezguez
Institut d'électronique fondamentale
Université Paris sud
91405 Orsay Cedex

Résumé

Cet article présente l'évaluation des performances potentielle d'un modèle de calcul parallèle, la maille associative, pour des applications complexes d'analyse d'images. Ce modèle repose sur des calculs globaux appliqués aux composantes connexes d'une image. Il peut conduire à une mise en œuvre matérielle efficace, et peut être utilisé pour la plupart des techniques d'analyse d'images. A titre d'exemple, la mise en œuvre d'une segmentation d'images en domaines de Voronoï homogènes est présentée. Les résultats obtenus par simulation sont très encourageants et les temps de calcul simulés, de l'ordre de la milliseconde permettant d'améliorer de plusieurs ordres de grandeur les performances obtenues sur stations de travail.

1 Motivations

Une des difficultés d'une mise en œuvre architecturale efficace de techniques d'analyse d'images est la grande diversité des fonctions utilisées et l'irrégularité des mouvements de données associés. Ces difficultés d'adéquation architecturale à l'analyse d'images font que, s'il existe des mises en œuvre matérielles de tel ou tel algorithme d'analyse d'images, aucune architecture spécialisée ne permet d'améliorer significativement l'ensemble des opérations requises dans ce domaine, où les temps de calcul se chiffrent pourtant en secondes, quand ce n'est en minutes.

Il est cependant possible de montrer que, au delà de l'apparente hétérogénéité des opérations mises en jeu en analyse d'images, on peut mettre en évidence une unité en termes de mouvements de données. En effet, la plupart de ces opérations peuvent, se ramener à une application d'un opérateur associatif et commutatif à un ensemble de pixels connexes, sommets d'un certain graphe. Les composantes connexes de ce graphe peuvent représenter des objets de l'image créés à partir des données (régions, contours), ou des entités géométriques construites implicitement en chaque point (droites, découpage hiérarchique de l'image, etc)[2].

Dans un premier temps nous présentons plus en détails ce modèle (*la maille associative*), et les mouvements de données primitifs qu'il nécessite. Puis nous évoquerons une architecture basée sur l'utilisation de calcul asynchrone, offrant un support très efficace pour réaliser de telles opérations. La rapidité et la bonne adéquation à l'application de tels mécanismes permet de supposer que les algorithmes d'analyse d'images peuvent s'exécuter de manière très efficace. Un outil de simulation que nous présentons ensuite a été conçu pour permettre de développer une algorithmique basée sur ces primitives et d'évaluer les performances obtenues. Il a ainsi été possible de vérifier les performances d'algorithmes complexes d'analyse d'image, comme des problèmes de segmentation reposant sur une décomposition en régions de Voronoï homogènes, comme le décrit la dernière section.

2 Modèle de calcul de la maille associative

Dans le modèle de la maille associative (MMA), on considère un modèle de calcul à parallélisme de données, dont les unités de traitement sont disposés suivant une topologie en maille à deux dimensions. Outre la présence de données (pixels) distribuées sur chaque processeur, on considère comme objet de base un sous-graphe du graphe d'interconnexion physique entre processeurs. Ces sous-graphes, que nous appellerons *mgraphes*, représentent des relations entre pixels pouvant correspondre dans le cas de problèmes d'analyse d'images à des objets à manipuler globalement dans l'application (régions contours, etc). Les objets sont formés par les composantes connexes de ces graphes, et les graphes servent de support à des calculs globaux. Leur codage repose sur la description ensembliste des arcs incidents en chaque nœud.

La principale primitive de calcul est une *association*, qui consiste, étant donné des données parallèles et un mgraphe, à appliquer à ces données un opérateur global — ou logique, maximum, addition, ... — sur les composantes connexes du mgraphe considéré. Chaque nœud reçoit ainsi le résultat de l'application de l'opérateur à sa propre composante connexe. A titre d'exemple trivial, si un graphe représente une région d'une image, l'application par une association d'un opérateur d'addition (plus-association) à une données distribuée identiquement à 1 en chaque pixel, permet de déterminer le nombre de pixels (la surface) de chaque région. Si le graphe considéré est un arbre orienté, l'association est alors dite préfixe, et chaque nœud reçoit l'application de l'opérateur aux nœuds qui sont en amont dans le graphe. Signalons également une autre primitive qui permet, étant donnée un graphe quelconque, d'en rechercher un arbre recouvrant, dont les racines sont en des points particuliers. D'autres opérations primitives, plus traditionnelles, sont également nécessaires : manipulations locales de graphes (création, union, etc) , opération combinant la valeur d'un nœud et de ses plus proches voisins suivant la présence d'arêtes dans un graphe (association élémentaire))[2].

3 Réalisation asynchrone des calculs globaux

Un des gros intérêt du modèle est la possibilité de le mettre en œuvre efficacement au moyen de mécanismes asynchrones. Pour réaliser efficacement une association sur un certain mgraphe, on peut montrer qu'il suffit dans un premier temps de *masquer* à zéro — qui est l'élément neutre des opérateurs considérés — les arcs n'appartenant pas au sous-graphe sur lequel le calcul s'opère. La mise en œuvre de l'association se fait alors simplement en combinant sur chaque processeur, les données (masquées) provenant des voisins avec la donnée locale, et en rediffusant *de manière asynchrone* le résultat vers les voisins. Au bout d'un temps au plus égal au produit du temps de traversée d'un nœud par le diamètre du graphe, les données se stabilisent, ce qu'un détecteur matériel de stabilité vérifie. Le processeur est de type SIMD et les manipulations de données qu'il effectue sont liées à la fréquence d'une horloge externe. Lors des opérations globales (associations), la propagation des informations est asynchrone ; par contre, la détection de la convergence du calcul se fait en testant la stabilité d'un nœud pendant une période de cette même horloge. Le principe d'une telle architecture est présenté en figure 1.

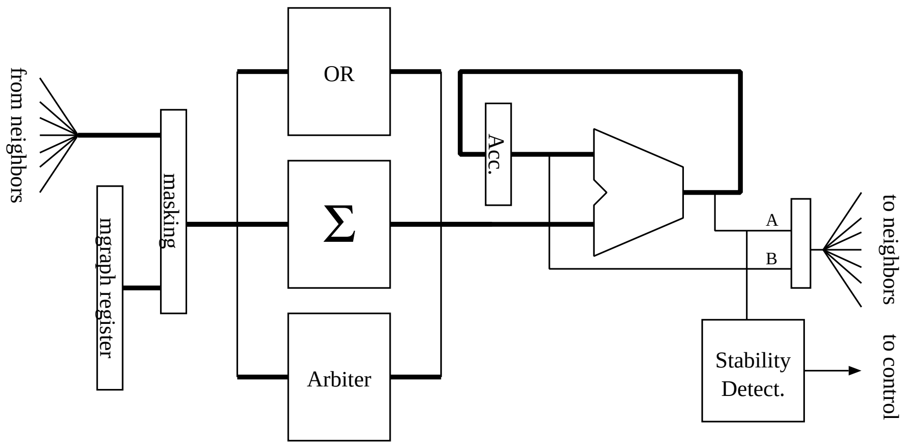

FIG. 1 – *Architecture de principe mettant en œuvre le modèle de la maille associative*

Un circuit intégré mettant en œuvre ces concepts a été réalisé[1]. Dans la technologie utilisée (CMOS 1μ), le temps de traversée mesuré d'un nœud varie entre 1 et 2 ns suivant la primitive considérée et une association de 16 bits nécessite un temps de quelques μs. L'évolution technologique aidant, on peut espérer gagner un ordre de grandeur en vitesse avec les possibilités d'intégration actuelles, et implanter une machine de 256×256 processeurs sur une carte PCI. Reste à vérifier l'impact que ces excellentes performances auront sur des algorithmes complet d'analyse d'images au moyen d'un simulateur fonctionnel et temporel.

4 Le simulateur de maille associative

La simulation de maille associative répond ainsi à un triple besoin : développer des algorithmes suivant ce nouveau modèle de programmation, évaluer finement leurs performances et aider à l'optimisation architecturale.

4.1 La bibliothèque de simulation

Le simulateur repose sur une bibliothèque C++, mettant en œuvre un modèle à parallélisme de données, de telle sorte que les algorithmes soient directement implantables sur une future machine. Dans la bibliothèque *sam* (simulate the associative mesh!), une classe C++ représente les images et une autre les mgraphes. Les mgraphes sont des images d'octets particulières dont chaque pixel indique par chacun de ses bits l'existence d'un arc incident depuis un des huit voisins du pixel. Des fonctions membres permettent de manipuler ces entités comme un tout (opérations ponctuelles sur des images ou des graphes) ou d'appliquer les primitives du modèle (association (globale, élémentaire, ou préfixe), arbre recouvrant).

Ces deux classes dérivent d'une classe *maille* qui représente l'entité machine et qui permet d'avoir accès à des informations de nature diverses : géométrie, contexte d'exécution, etc. Ce dernier point permet de mettre en œuvre une restriction du contexte d'exécution à un ensemble de processeur actifs, comme dans la plupart des langages à parallélisme de données. La plupart des primitives sont sensibles au contexte. Plus précisément, pour les primitives ponctuelles, le calcul ne prend place que sur les processeur actifs. Pour les primitives globales (associations) ou locales (associations élémentaires), l'ensemble des processeurs contribue au calcul, mais le résultat n'est réécrit que sur les processeurs actifs.

Les images manipulables sont de nature diverse (entiers ou flottants) et sont paramétrables en nombres de bits. Un préprocesseur a permit de mettre en œuvre systématiquement les opérateurs et les formats de données ; actuellement près de 400 fonctions élémentaires sont instantiables par l'utilisateur (la plupart sont forme d'opérateurs surchargés)[3].

4.2 Évaluation temporelle

Un aspect important de la simulation est la possibilité de réaliser une évaluation temporelle et un profil d'exécution du programme. Une variable globale permet de contrôler si l'exécution d'une primitive se fait en mode d'évaluation temporelle ou non. Sans évaluation temporelle, le simulateur emploie une technique rapide fortement optimisée à la fois pour les opérations ponctuelles et globales. En mode d'évaluation, la même technique est utilisée pour les opérations ponctuelles, mais le nom de la procédure appelée, avec les caractéristiques des arguments (type et nombre de bits) est mémorisé dans un historique. Pour les opérations globales, les calculs se font bit-à-bit, suivant le parcours effectivement suivi dans la mise en œuvre matérielle. Le plus grand nombre de nœuds traversé, ainsi que le type d'opération est mémorisé dans l'historique.

Les information inscrites dans l'historique sont indépendantes de la technologie et de l'architecture. Une fois la simulation réalisée, il est possible de les convertir en temps physique, à l'aide d'informations technologiques (temps de traversée par nœud suivant les opérations, période d'horloge) ou architecturales (cycles nécessaires aux différentes manipulations synchrones d'informations sur les processeurs). Un profil d'exécution, ou des statistiques de natures diverses sont ainsi réalisables.

5 Exemples de simulations

La manipulation d'objets connexes est de première importance en analyse d'images et se trouve être d'ailleurs la motivation première de cette architecture. Les objets peuvent indifféremment être de deux types : compacts (régions) ou filiformes (contours). Un certain nombre d'algorithmes ont ainsi été étudiés et dans certains cas implantés sur le simulateur. A titre d'exemple, nous présenterons les principes de la mise en œuvre d'une segmentation en régions reposant sur une fission et fusion (split and merge) sur des domaines de Voronoï.

Cette méthode présente l'avantage, par rapport aux techniques classiques de fission et fusion, d'éviter les artefacts provenant d'un découpage régulier de l'image[5]. Pour cela, les régions considérées sont les domaines de Voronoï[1] associés à des points particuliers de l'image appelés *germes*. Si une région est inhomogène, on introduit un nouveau germe en son sein, et on recalcule les domaines de Voronoï jusqu'à ce que toutes les régions soient homogènes. Dans un deuxième temps, on regroupe les régions suffisamment similaires (cf algorithme 1).

Algorithme 1 Principe d'une segmentation par diagrammes de Voronoï

1 Sélectionner au hasard quelques points comme germes initiaux
 ▷ *Division des régions inhomogènes*
2 Determiner les domaines de Voronoï associés aux germes
3 Calculer le critère d'homogénéité associé à chaque région
4 Si la région est insuffisamment homogène, lui ajouter un nouveau germe
5 Répéter les étapes 2-4 jusqu'à ce que toutes les régions soient homogènes.
 ▷ *Fusion des régions homogènes*
6 Si une region est voisine d'une ou plusieurs autres régions *similaires*, créer une nouvelle région en fusionnant les régions en question
7 Répéter 6 tant que des fusions sont possibles

5.1 Mise en œuvre sur maille associative

Dans le modèle de la maille associative, une région est représentée par une composante connexe d'un certain graphe. Ainsi la détermination de l'homogénéité d'une région se fait directement avec les primitives du modèle. En effet, les critères classiques d'homogénéité (max-min $< \epsilon$, variance $< \epsilon$) se ramènent simplement à des applications globales d'opérateurs (associations). Dans la mise en œuvre que nous avons réalisé, le critère choisi fut le suivant : une région est dite homogène si la différence entre ses niveaux de gris maximum et minimum est inférieur à un seuil (ce qui n'implique que des calcul de minimum et de maximum globaux sur une région).

La détermination des régions de Voronoï (étape 2) est réalisée de manière itérative. Un arbre recouvrant le graphe physique et enraciné sur les germes permet de définir une première approximation de ces régions. Celle-ci serait parfaite si le recouvrement par des arbres était totalement isotrope. A défaut, chaque pixel voisin d'une autre région récupère les coordonnées du germe qui lui est associé. Ceci permet de parcourir les différents voisins d'une région et de redistribuer successivement leurs coordonnées. Chaque pixel connaissant les coordonnées des germes qui entourent sa région courante peut déterminer le germe le plus proche par une simple opération. On met ensuite à jour le graphe en créant des liens entre les pixels souhaitant s'associer au même germe. La distribution des coordonnées des germes se fait au moyen d'opérations globales sur les graphes représentant les régions[4].

L'étape de division nécessite de choisir un nouveau germe dans une région. Pour cela, nous avons étudié diverses heuristiques, mais la plus efficace consiste à favoriser les choix de pixels excentrés du germe actuel à la fois géographiquement et en niveau de gris. Pour cela, nous commençons par recouvrir la région par un arbre et nous calculons la distance à la racine (par une opération préfixe). Puis nous sélectionnons les pixels à une distance suffisante de la racine et parmi ceux-ci nous choisissons celui dont la différence à la racine est maximale. Un critère basé sur l'indice permet de limiter le choix à un unique pixel.

L'étape de fusion est la plus complexe à mettre en œuvre. Nous avons choisi une fusion qui ne se limite pas strictement à des régions de Voronoï, pour améliorer la qualité de la fusion (pour des principes de mise en œuvre d'une fusion dont chaque région soit un domaine de Voronoï, voir [4]). La fusion procède par itérations, chacune comprenant plusieurs étapes. Dans un premier temps, on recherche des fusions évidentes : si le domaine des niveaux de gris d'une région A est strictement inclus dans celui de B (supposé homogène), A et B peuvent de toute évidence fusionner sans précautions particulières. On recherche ainsi une suite d'inclusions sous forme d'un arbre dont le domaine de chaque enfant est inclus dans le parent. En cas d'égalité des domaines, la prise en compte des indices des processeurs permet de

1. Soit un ensemble de points P_i d'une image. On appelle domaine de Voronoï associé au point P_j l'ensemble des pixels de l'image plus proche de P_j que de tout autre point $P_i, i \neq j$.

maintenir une relation d'ordre partiel. Ensuite, il suffit de fusionner en bloc l'ensemble des nœuds des arbres ainsi formés.

On traite ensuite les fusions ne correspondant pas à une relation d'inclusion, mais dont l'homogénéité de la région formée est suffisante. Pour cela, chaque région commence par rechercher parmi ses voisines celle qui serait le meilleur choix pour une fusion (en minimisant le critère d'homogénéité), ce qui nécessite des opérations ponctuelles sur le graphe frontière (meilleur voisin local) et une centralisation globale par région (meilleur voisin de la région). On réalise dans une première passe les fusions dont les choix de meilleur voisin sont réciproques (A et B fusionnent si A est le meilleur voisin de B et réciproquement). On considère ensuite le graphe indiquant les relations de fusion possibles (meilleure région voisine) que l'on symétrise. Ce graphe possède ainsi un arc entre deux régions si l'une des deux a choisi l'autre comme meilleure fusion. On choisit par composante connexe de ce graphe la région de niveau de gris maximal, on recouvre le graphe d'un arbre, et une min-association préfixe permet de déterminer la zone autour de cette région qui peut fusionner.

6 Analyse des résultats de simulation

Cet algorithme a été mis en œuvre avec quelques 1000 lignes de C++ sur la bibliothèque sam. Il a été appliqué à plusieurs images (256x256) de la base de données du GDR ISIS, ainsi qu'à des images synthétiques simples. L'objectif étant plus l'étude des performances en termes de temps de calcul qu'en segmentation, l'algorithme a été appliqué aux image sans prétraitement, ou avec un simple filtrage médian. Les résultats de segmentation obtenus sont globalement satisfaisants, même si la qualité de la segmentation souffre de la forte dépendance au bruit du critère max−min choisi. Un exemple d'image et le résultat obtenu sont présentés en figure 2. En tout état de cause, la qualité serait améliorable, soit par des prétraitements adaptés, soit en considérant d'autres critères d'homogénéité (variance, etc), soit à l'aide des post-traitements.

FIG. 2 – *Exemple de résultats de segmentation*

6.1 Résultats temporels

La simulation temporelle a été conduite d'une part à partir des temps de traversée des nœuds mesurés sur le circuit intégré réalisé en CMOS 1μ, et qui donnaient des moyennes de respectivement 1 ns et 2.1 ns pour les associations avec des opérateurs logiques et arithmétiques, et une fréquence de fonctionnement de 50 MHz. Par ailleurs, des projections sur les technologies d'intégration actuelles (CMOS 0.25 micron) ont été faites. Ainsi, on peut espérer une diminution théorique minimale par 10 des temps de traversée des nœuds. La fréquence de fonctionnement devrait atteindre 100 MHz sans problème. Il devrait être possible, au moyen de techniques d'autosynchronisation, de surmonter les problèmes de synchronisation des milliers de processeurs et de travailler à des fréquences potentielles de 500 MHz, voire plus.

6.2 Résultats globaux

Les temps de calcul sont très variables suivant la nature de l'image et son degré d'homogénéité ainsi que le critère d'homogénéité utilisé pour la segmentation. Pour une même image, on a même pu constater des variations de 10 à 15% suivant l'initialisation des générateurs aléatoires. Les temps simulés varient entre 3 et 10 ms suivant les images (en CMOS 1μ) (7.1 ms pour la figure 2).

Pour les images considérées, avec des critères max−min variant de 10 à 50, le nombre de régions produit par division varie de quelques centaines (images très homogènes) a 10–20000 (images réelles). La phase de fusion ramène ce nombre de quelques unités (image synthétiques homogènes) à quelques centaines.

La majeure partie du temps est passée à des manipulation ponctuelles ou locales d'information (80–90%). De plus, du fait de l'existence de régions de taille très petites, la majorité des opérations asynchrones globales ne prend qu'un cycle d'horloge. On constate ainsi une dépendance quasi-linéaire entre les temps de calcul et la fréquence d'horloge. En CMOS 0.25μ, les temps de calcul tombent en dessous de la ms à partir d'une horloge de 300 MHz (700 μs à 500 MHz).

6.3 Analyse détaillée des différentes étapes de segmentation

Division L'étape de segmentation la plus coûteuse est la phase de division. Elle représente suivant les images de 70 à 80% du temps total. Elle nécessite entre 10 et 20 passes suivant les images et le critère d'homogénéité pour être menée à bien, et chaque itération nécessite en moyenne 450 μs. Mais les différentes étapes ne sont pas de durée identiques ; le temps de calcul augmente d'un facteur 3 à 5 entre la première itération, dans laquelle il y a peu de régions, et la dernière. Dans chacune de ces itérations, c'est la détermination des régions de Voronoï qui représente la quasi totalité du temps. En effet, l'algorithme de calcul des régions de Voronoï nécessite d'accéder à *toutes* les coordonnées des germes dans les régions voisines ce qui, si une grosse région est entourée de petites, peut impliquer une sérialisation importante des accès à ces régions. De plus, du fait de l'inhomogénéité du temps de traversée, la région initialement formée peut être déformée, et il est nécessaire d'itérer le calcul pour traiter le cas de régions *cachées* derrière d'autres (en général le nombre d'itération est toujours compris entre 1 et 5 pour des variations de temps de propagation simulés allant jusqu'à un facteur 3).

Fusion L'étape de fusion ne représente qu'une faible partie du temps total. En règle générale, seules quelques itérations (5 à 15) sont nécessaires pour achever la fusion. Ces phases ont une durée relativement constante (typ. 140 μs en CMOS 1μ). Au sein de chaque itération, en général plus de la moitié du temps est passé dans la phase de détermination du meilleur voisin. Ceci nécessite en effet, comme pour déterminer les domaines de Voronoï, de parcourir le voisinage. Par contre, il est possible de déterminer *sur les processeurs frontières* la qualité de la fusion, et le nombre d'itération est au plus limité par le nombre de régions voisines d'un pixel frontière (8 au maximum).

Pour les images considérées, les phases de fusion les plus efficaces sont celles qui concernent des ensembles complets de régions ; dans les premières itérations, elles permettent de réduire d'un facteur 2 à 3 le nombre de régions. L'étape de fusion par choix réciproque devient utile dans les dernières étapes ou le nombre de régions fusionnées devient faible.

6.4 Optimisations et travaux futurs

Il existe dans l'algorithme programmé une grande latitude d'optimisation, sur divers plans. Tout d'abord, pour simplifier la programmation, et notamment les problèmes de précision, l'ensemble des calculs est réalisé avec des images en entiers 32 bits, alors même que les niveaux de gris ne nécessiteraient que des calculs sur un octet, et les informations géométriques (indices, surfaces) un mot de 16 bits. Or, à la fois les opérations ponctuelles et les associations (de toute nature), ont un temps proportionnel au nombre de bits considéré. Un recodage permettrait ainsi probablement de gagner un facteur 2 à 3.

La phase de division étant la plus coûteuse, un gain important peut résulter de son optimisation. Nous partons déjà d'un nombre de points initiaux aléatoires élevé (1000) ce qui réduit de 40% le temps de la division sans dégrader l'image résultat. D'autres heuristiques d'ajout de points permettraient probablement de gagner quelques itérations. L'algorithme de calcul des régions de Voronoï est fortement améliorable pour limiter la sérialisation des accès aux régions voisines et/ou rendre l'approximation initiale des domaines meilleure. Enfin, on peut se questionner sur l'utilité de cette phase coûteuse (alors qu'elle est la plus rapide sur une machine séquentielle), vu la surface moyenne des régions produites (typ. 6 pixels !). Nous envisageons ainsi de nous passer de la phase de division, quitte à faire un découpage ultérieur en région de Voronoï si celui-ci est nécessaire pour le codage des régions.

La phase de fusion est également optimisable, notamment pour augmenter le nombre de régions fusionnées dans les dernières phases (souvent inférieur à 10). L'algorithme est très sensible aux heuristiques. Le fait de préférer comme meilleur voisin homogène celui de plus grande taille a ainsi déjà permis de réduire de 20 à 30 % le nombre d'itérations de fusion, et d'autres heuristiques sont envisageables.

Des progrès peuvent également être attendus, au moins sur la qualité de l'image résultat, en améliorant les techniques d'analyse d'images mises en œuvre, soit par des prétraitements sophistiqués, soit au moyen d'autres critères d'homogénéité, soit en prenant en compte des informations sur les contours.

7 Conclusion

Nous avons présenté un modèle visant à résoudre les problèmes d'adéquation algorithme-architecture en analyse d'image. La méthodologie a consisté à définir des mouvements de données élémentaires permettant d'exprimer les algorithmes concernés et à rechercher une mise en œuvre efficace au moyen de techniques asynchrones. Un simulateur a permis de montrer la viabilité et la facilité de programmation du modèle pour mettre en œuvre des techniques complexes d'analyse d'images. Par ailleurs, les temps obtenus laissent espérer, si une machine parallèle mettant en œuvre le modèle était réalisée aujourd'hui, des segmentations en des temps inférieurs à la ms, alors que ces techniques nécessitent couramment plusieurs secondes sur les stations de travail actuelles.

Références

[1] D. Dulac, S. Mohammadi, and A. Mérigot *Implementation and evaluation of a parallel architecture using asynchronous communication*, Proc. IEEE Workshop CAMP'95, IEEE Press, Los Alamos, 1995, pp. 106–111.

[2] A. Mérigot, *Associative Nets: A Graph-Based Parallel Computing Model,* IEEE Trans. on Computers, C46-5, may 1997.

[3] A. Mérigot, *La bibliothèque de simulation SAM (version 0.40b), manuel utilisateur*, 1997 (accessible sur <ftp://ftp.ief.u-psud.fr>).

[4] A. Mérigot, G. Constantinescu, B. Ducourthial, *Implementing image analysis with a graph based parallel computing model*, In *Graph Based Representation*, J. M. Jolion, W. Kropatsch Eds, Springer Verlag (to appear).

[5] M. Tuceryan, A.K. Jain, *Texture segmentation using Voronoi polygons* IEEE Trans. on PAMI, vol 13; 211–216, 1990.