Etude d'adequation algorithme-architecture pour terminaux multimedia portables: segmentation d'images par un reseau de processeurs asynchrones

B. Galilee

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque

THÈSE

pour obtenir le grade de

DOCTEUR DE L'INPG

Spécialité : «Microélectronique»

préparée aux laboratoires **France Telecom R&D** et **TIMA**
dans le cadre de l'**École Doctorale «Électronique, Électrotechnique, Automatique, Télécommunications, Signal»**

présentée et soutenue publiquement
par

Bruno GALILÉE

le 8 Octobre 2002

Titre :

ÉTUDE D'ADÉQUATION ALGORITHME-ARCHITECTURE POUR TERMINAUX MULTIMEDIA PORTABLES : SEGMENTATION D'IMAGES PAR UN RÉSEAU DE PROCESSEURS ASYNCHRONES.

Directeur de thèse :

Marc RENAUDIN

JURY :

M. Michel PAINDAVOINE	, Président
M. Alain MÉRIGOT	, Rapporteur
M. Thierry BERNARD	, Rapporteur
M. Marc RENAUDIN	, Directeur de thèse
M. Pierre-Yves COULON	, Co-directeur de thèse
M. Franck MAMALET	, Co-encadrant

Remerciements

S'il n'y avait que quelques personnes à remercier, alors la rédaction de cette partie du manuscrit serait une tâche aisée. Cependant tout au long de mes études doctorales, des équipes formidables m'ont entouré et chacune d'elles mériterait une attention toute particulière. Ne souhaitant pas vous perdre, cher lecteur, dès le début de votre investigation, je vais tenter d'être aussi bref que possible et je ne prendrai pas le risque de citer de noms car l'énumération complète des personnes envers qui je suis reconnaissant serait tout aussi fastidieuse que la recherche des liens d'héritage entre les objets de la librairie SPOT[1] : **les personnes concernées se reconnaîtront naturellement !**

Ces travaux de thèse ont pu être réalisés grâce à un partenariat entre le centre de recherche Norbert Segard de France Telecom R&D, le laboratoire des Techniques de l'Informatique et de la Microélectronique pour l'Architecture d'ordinateurs (TIMA), et le laboratoire des Images et Signaux (LIS), tous trois situés à Grenoble. J'exprime alors tout mon respect pour les directeurs de ces centres de recherche et ses équipes pour leur accueil dans leur établissement.

Je remercie vivement les membres du jury pour l'étude approfondie de mon travail, pour les nombreuses discussions scientifiques et pour leur rigueur qui ont largement participé à mon épanouissement intellectuel.

Je remercie également l'ensemble des voisins de bureau plus ou moins proche, du module plus ou moins jaune-canari virant à l'orange-marron pour leurs conseils, encouragements, démarches administratives et humour plus ou moins provocateur.

Je ne suis pas d'accord avec le proverbe qui dit : «loin des yeux, loin du cœur», car il est en contradiction avec les sentiments et remerciements que j'exprime aux professeurs, maîtres de conférence et doctorants des laboratoires CIME et TIMA car sans eux, l'aspect microélectronique de cette thèse n'aurait pas pu être approfondie.

Enfin, je tiens à remercier mes amis et proches avec qui j'ai passé des moments inoubliables et qui ont parfois pris le risque de s'intéresser à mes études...

... et Suliane (et oui, les exceptions sont réservées aux personnes d'exception).

[1] Signal Processing Object Toolbox : j'en profite pour faire un p'tit clin d'œil sympathique à toute l'équipe du pôle image du CCETT de Rennes !

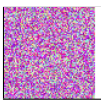
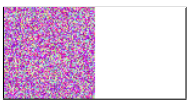
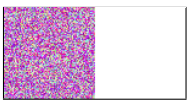


Table des matières





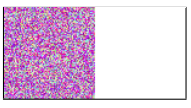
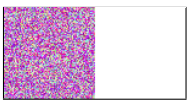


Table des figures



Liste des tableaux

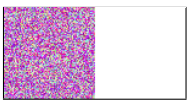
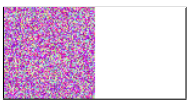
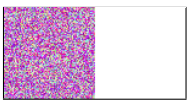
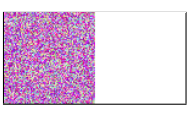


Table des définitions





Liste des algorithmes



Chapitre I

Introduction

Un domaine très actif dans la recherche de cette dernière décennie est le multimedia. Un des objectifs à long terme est d'intégrer ce support de communication dans les futurs terminaux portables. Nous décrivons ici un acte de recherche qui part des normes de codage vidéo orienté-objet pour aller jusqu'à l'implantation microélectronique d'un algorithme de segmentation d'images par un réseau de processeurs asynchrones.

I.1 Contexte et problématique

Le cadre de cette thèse est l'étude de l'intégration des traitements multimedia dans des environnements très contraints que sont les terminaux portables. Nous distinguons aujourd'hui deux grandes classes d'applications dans le domaine de la vidéo sur ces terminaux portables :

- **Les vidéophones.** Il sont uniquement destinés à recevoir une séquence vidéo codée, à la décoder, puis à l'afficher ;
- **Les visiophones.** En plus des fonctionnalités du vidéophone, ils sont destinés à filmer le locuteur, à coder et à envoyer la séquence.

La diffusion de flux vidéo vers des vidéophones est une technologie actuellement disponible grâce au *streaming* vidéo sur internet mobile. Cependant, ce système de communication "client-serveur" n'est pas adapté aux visiophones qui eux sont de type "point à point". Le visiophone est bien plus complexe car il intègre un codeur vidéo temps réel en plus du décodeur d'un vidéophone. Dans le domaine des terminaux portables, il s'agit d'allier puissance de calcul, surface du circuit (coût de fabrication et conditionnement *packaging*) et consommation d'énergie (maximisation de l'autonomie du terminal).

Si, aujourd'hui, la téléphonie mobile est un moyen de communication "classique", la visiophonie en situation de mobilité, quant à elle, est une nouvelle forme de communication qui tend à percer actuellement, notamment au Japon, mais qui soulève encore de nombreux défis technologiques. Tout d'abord, le débit d'information nécessaire pour une séquence d'images de qualité et de taille QCIF[1] est environ dix fois plus élevé par rapport à celui du transport de la voix (norme GSM[2] *Global*

[1] *Quart-Common Intermediate Format* : format d'images pour visiophones (176×144 pixels).
[2] `http://www.nortelnetworks.com/solutions/providers/wireless/gsm/etsi.html`

System for Mobile communication à 14.4kbits/s). La nouvelle norme UMTS (*Universal Mobile Telecommunications Service*) de transmission hertzienne à haut débit [OP98] permettrait, dans ses premières versions, de fournir un débit de transmission allant jusqu'à 384 kbits/s en voix descendante et 64 kbits/s en voix montante. La norme MPEG4 *Simple Profile* fournirait un flux vidéo respectant ce débit maximal de 64 kbits/s (pour un visiophone portable, la limite est fixée par le débit symétrique), sachant que 14 kbits/s sont dédiés à la voix, environ 5 kbits/s pour les codes correcteurs d'erreurs et 45 kbits/s pour la vidéo. Nous voyons aujourd'hui apparaître des solutions matérielles comme celle présentée dans [NYK+02] (circuit de 28 mm^2 consommant 9 mW pour coder 15 images QCIF par seconde), par exemple, qui implantent un codeur-décodeur MPEG-4 simple profil (*Simple Visual Profile* [MPE02]), en adéquation avec les terminaux portables mais dont la qualité à faible débit et en mobilité laisse encore à désirer.

La norme MPEG4 [MPE] prétend augmenter le taux de compression et fournir de nouveaux services en extrayant les objets de l'image. Afin d'améliorer entre autres la qualité des images, il serait intéressant d'implanter le profil *Core* [MPE02] de la norme tout en respectant les contraintes liées aux terminaux mobiles. Ce profil nécessite l'extraction des objets de la scène, il est donc nécessaire d'ajouter au système une brique de segmentation.

L'intégration d'un algorithme de segmentation, *i.e.* un des traitements clés d'une chaîne complète de codage vidéo orienté objet, dans un visiophone portable, reste à ce jour un verrou technologique à lever. Les principales contraintes sont les temps de traitement (puissance de calcul suffisante), la consommation (au maximum quelques milliwatts) et la faisabilité d'implantation (surface et coût de production). Cette thèse contribue à lever ce verrou en proposant un algorithme de segmentation original et rapide et une implantation matérielle peu consommante.

Dans cette étude, les critères importants sont la validation et la rapidité de l'algorithme car l'opération de segmentation doit être juste, et représenter une charge marginale par rapport à la chaîne complète de codage. De plus, la consommation d'énergie de son implantation matérielle doit être minimale. Deux idées clés pour répondre à ces critères composent cette thèse :

1. Le réordonnancement d'un algorithme de segmentation afin que son implantation sur une architecture parallèle soit la plus efficace possible.
2. La validation et la conception d'un réseau de processeurs asynchrones connus pour fournir un résultat au plus tôt et limiter la consommation d'énergie par une activité conditionnelle.

I.2 Démarche scientifique et plan de la thèse

La section précédente ayant présenté succinctement le contexte et les objectifs des études, nous présentons la méthodologie adoptée. L'axe principal de ces recherches va de l'algorithme de segmentation vers l'architecture asynchrone.

Cette thèse fait partie d'un projet exploratoire souhaitant répondre à la question : «Est-il possible d'intégrer une chaîne de codage vidéo orienté-objet dans un terminal multimedia portable ?»

Chapitre II. Le point de départ est une étude bibliographique sur les normes de codage vidéo de nouvelle génération (§II.2). Sachant que l'application visée est principalement la visiophonie, nous avons étudié particulièrement les applications s'intéressant aux séquences visiophoniques (séquences où un locuteur occupe la majorité de l'image).

Nous nous intéressons au profil *Core* de la norme MPEG4, et parallèlement, à l'intégration d'un codeur vidéo orienté objet dans un terminal portable. Lors de l'étude bibliographique sur ces codeurs, nous avons extrait un opérateur clé : **l'opérateur de segmentation**.

L'état de l'art sur les algorithmes de segmentation (§II.3) montre que les solutions séquentielles sont très coûteuses (un unique processeur effectue plusieurs balayages de l'image, accède aléatoirement à une mémoire de grande taille...). Les implantations matérielles existantes pour ces algorithmes séquentiels sont fortement consommantes lorsqu'on vise au moins 25 segmentations d'images QCIF par seconde. Il s'avère alors nécessaire de déterminer une implantation parallèle et économique d'un algorithme de segmentation.

De par l'inadéquation des implantations matérielles parallèles existantes de cet opérateur avec les terminaux portables, nous proposons de lever ce verrou technologique. Le problème soulevé est alors double : améliorer une implantation parallèle d'un algorithme de segmentation existant (aspect temps réel à respecter pour la chaîne totale de codage) tout en vérifiant la faisabilité d'intégration à faible coût (surface et consommation du circuit).

L'implantation parallèle d'un algorithme est d'autant plus efficace que les traitements sont locaux et désynchronisés [BT89]. Les points de synchronisation globale limitent les performances des implantations parallèles car avant de pouvoir initier la phase suivante de l'algorithme, les processeurs inactifs doivent attendre les autres qui sont encore en cours de traitements.

Nous nous sommes orientés vers les algorithmes de segmentation issus de la morphologie mathématique (§II.4) car ils n'utilisent que des opérateurs du type MIN ou MAX ayant une faible complexité d'implantation. Enfin, de par son caractère fortement localisé, nous choisissons l'algorithme de *Hill-Climbing* comme candidat à une implantation parallèle. Cet algorithme interprète l'image gradient comme un relief topographique, et propage les étiquettes de chaque minimum suivant les lignes de plus grande pente.

Chapitre III. Les études présentées dans ce chapitre constituent le cœur du travail de thèse en proposant et validant formellement un algorithme de *Hill-Climbing* réordonnancé. Les implantations existantes de l'algorithme de *Hill-Climbing* (§III.1) requièrent au minimum trois points de synchronisation globaux : la suppression des plateaux non-minima, la recherche des minima et détermination des partitions.

Nous proposons une version originale et réordonnancée de l'algorithme de *Hill-Climbing* (§III.2) où **seul un point de synchronisation subsiste :** le point de convergence. Les traitements s'effectuent au niveau pixel dont le comportement est régi par une simple machine à trois états. Cette dernière intègre, de façon désynchronisée au sein de chaque pixel, la détection des minima, étiquetage des plateaux minima, la propagation des étiquettes suivant les lignes de plus grande pente

et la suppression des plateaux non-minima (configuration pathologique pour les algorithmes de segmentation suivant la distance topographique).

Afin de prouver qu'il effectue bien une segmentation de l'image et que les partitions obtenues sont correctes, nous formalisons cet algorithme et démontrons **la convergence de l'algorithme et la justesse des régions** (§III.3) quelque soit l'image d'entrée. Cette formalisation se présente sous forme d'une succession de démonstrations où la théorie des graphes et le modèle de réseau associatif (modèle mathématique de calculs cellulaires) sont mutuellement employés.

Sachant que cet algorithme est destiné à segmenter des images au sein d'un terminal portable, nous devons déterminer une architecture qui soit en adéquation avec l'algorithme réordonnancé, et qui vérifie les contraintes des terminaux embarqués.

Pour une implantation sur un terminal portable, il est impératif de respecter des contraintes de poids, de volume, de consommation électrique, de compatibilité électromagnétique. De manière générale, minimiser la consommation va souvent à l'encontre du degré de liberté que l'on souhaiterait conserver vis-à-vis du système à réaliser (par exemple, garder la possibilité de modifier le codeur vidéo au cours de la vie du terminal). On sait en effet que le rendement algorithme-architecture est d'autant meilleur que la réalisation est dédiée, c'est à dire optimisée pour reproduire uniquement la fonction souhaitée. La maximisation des performances étant dans notre cas prioritaire au détriment de la flexibilité, nous avons opté pour la modélisation d'un ASIC (*Application Specific Integrated Circuits*).

Chapitre IV. L'algorithme réordonnancé et désynchronisé à présent défini, nous cherchons à explorer et à valider par simulation des architectures parallèles adaptées. En perspective d'une implantation faiblement consommante, les circuits asynchrones permettent de limiter les besoins en énergie par une activité conditionnelle du circuit : seuls les processeurs qui calculent consomment. Comme l'algorithme présente une forte localité des traitements, nous choisissons cette technologie car elle permet la mise en veille automatique des processeurs inactifs et ce, sans contrôle supplémentaire.

Un état de l'art sur les architectures parallèles (§IV.1) nous permet de converger vers un réseau de processeurs faiblement couplés disposés en grille ou en tore replié, suivant un parallélisme SPMD (*Single Program Multiple Data stream*), et un partitionnement des données LPGS (Localement Parallèle Globalement Séquentiel, §IV.2).

Une méthodologie de validation et de spécification du couple algorithme-architecture est établie. Nous choisissons l'environnement de simulation fourni par la bibliothèque de prototypage rapide SystemC^{TM} (§IV.3) développée en C^{++}. La modélisation à un haut niveau d'abstraction nous permet une modélisation générique du réseau, facilitant ainsi l'exploration du spectre des architectures, et de simuler un réseau de processeurs asynchrones suivant un parallélisme SPMD. En vue d'une étude de faisabilité d'implantation microélectronique, les systèmes parallèles allant de la granularité la plus fine (un processeur par pixel) jusqu'à la plus grosse (un processeur pour l'image) ont pu être évalués.

Nous montrons que pour toute taille de grain, les partitions obtenues sont cohérentes avec celles fournies par des algorithmes de segmentation classiques (§IV.4). La définition et l'utilisation d'une métrique adaptée nous permettent de quantifier la

vitesse et la consommation du réseau, et nous fournissent ainsi une base d'estimation quant à la pertinence des architectures modélisées pour un terminal portable (§IV.5).

Nous montrons que la segmentation d'une image QCIF ou SQCIF est de l'ordre de **1 000 fois plus rapide** avec le couple algorithme réordonnancé - architecture à fine granularité comparativement à un algorithme séquentiel implanté sur un processeur RISC (*Reduced Instruction Set Computer*) adapté aux systèmes embarqués.

Nous montrons par ailleurs que l'architecture présentant le meilleur compromis temps de traitement/charge de calculs est une grille 4-connexe composée de quelques centaines de processeurs. Pour les canaux de communication, une taille mémoire dans les canaux de communication de quelques points est suffisante.

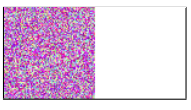
Chapitre V. L'étude de consommation et de faisabilité d'intégration est réalisée par une étude de bas niveau : la conception microélectronique en CHP (*Communicating Hardware Processes*) et VHDL (*Very high scale integration Hardware Description Language*) d'un processeur élémentaire respectivement asynchrone et synchrone pour la granularité la plus fine (un processeur par pixel).

La modélisation d'un processeur en CHP montre la faisabilité de conception d'un réseau asynchrone de processeurs. Cependant, la synthèse et les estimations de consommation d'un réseau complet n'ont pu aboutir faute de temps et de maturité suffisante des outils de conception des circuits asynchrones. C'est pourquoi nous nous sommes intéressés à la conception d'un réseau de pixels synchrones.

Bien que notre étude ne se soit pas attachée aux méthodes de chargement/déchargement des images, nous présentons une architecture synchrone implantant efficacement les traitements parallèles de l'algorithme de segmentation réordonnancé. L'utilisation des outils synchrones industriels nous permet d'estimer les principales caractéristiques d'une telle architecture. Chaque processeur synchrone au niveau de granularité le plus fin (un processeur par pixel) serait composé de 419 cellules standards, soit un circuit d'environ $1\,\mathrm{cm}^2$ pour un réseau de 88×72 processeurs en technologie $0.18\,\mu$m. **La fréquence maximale de segmentation est d'environ 120 000 images par seconde** (hors chargement des images) et la consommation de quelques milliwatts pour une cadence de segmentation de 25 images par seconde.

À partir des analyses d'activité du réseau, l'utilisation de **la technologie asynchrone** nous permettrait alors de **diviser par 20 la consommation** d'énergie, soit seulement **quelques dizaines de microwatts !** De par la rapidité des calculs et la faible consommation du réseau, une telle architecture serait adaptée à un terminal portable. Seule la surface trop élevée du circuit ne respecte pas les contraintes d'intégration des terminaux portables. La complexité est principalement due aux points mémoires qui sont sous forme de registres, et au comparateur 13 bits nécessaire. L'emploi d'une architecture à plus grosse granularité permettrait alors de réduire cette surface car la mémorisation des données dans un RAM (*Random Access Memory*) réduirait la surface occupée par les points mémoires, et le nombre d'unités arithmétiques serait moindre. Cette étude fait partie des perspectives de cette thèse.

Chapitre VI. Ainsi, l'originalité de cette thèse réside dans le réordonnancement de l'algorithme de *Hill-Climbing* et l'étude de son implantation microélectronique avec une technologie asynchrone. La contribution de ces travaux porte principalement sur



le traitement de l'information contenue dans le réseau de processeurs, c'est-à-dire le réordonnancement et la formalisation de l'algorithme de segmentation (chapitre III) et une exploration dans un langage de haut niveau des architectures adaptées à l'algorithme (chapitre IV).

Même si la mise en œuvre microélectronique reste perfectible, **le couple algorithme-architecture permet un gain en vitesse et en consommation de plusieurs ordres de grandeurs** par rapport aux solutions déjà proposées.

Tout au long de ce manuscrit, les termes anglais sont présentés *en italique*, les termes latins *en polices penchées*, et les termes importants **en gras**. L'ensemble des images situées en bas de chaque page impaire constitue une animation de la propagation des données dans un réseau de processeurs asynchrones.

Chapitre II

Codeurs vidéo orienté objet et algorithmes de segmentation

Le chapitre précédent présentait le contexte général des études, et plus particulièrement montrait l'intérêt de développer un outil de segmentation d'images adapté aux futurs terminaux multimedia portables.

À présent nous exposons le contexte scientifique, c'est-à-dire les éléments de base utiles au développement de nos travaux. Tout d'abord, l'image est présentée sous sa forme la plus habituelle : une matrice à deux dimensions de points (pixels). Ensuite, nous parlons de la nouvelle génération de codeurs vidéo orienté objet décrite par la norme MPEG-4. Cette dernière cherche à représenter les objets des images de façon indépendante, laissant ainsi libre cours à tout type d'applications et de stratégies de codage. Toutefois, le développement d'un tel codeur ne peut s'abstraire d'**un opérateur de segmentation** : un algorithme capable d'extraire les objets de l'image. Ce dernier point est étudié en détails et nous choisissons celui qui est en adéquation avec une implantation parallèle de faible complexité.

Au chapitre III, nous nous intéresserons au cœur de cette thèse : une version réordonnancée de l'algorithme de segmentation par ascension de colline.

II.1 La matière première : l'image

Cette section présente dans un cadre général l'élément que nous sollicitons tout au long de ce manuscrit : **l'image**. Nous présentons son aspect (couleur, luminance, fixe, séquence...), sa structure (maille carré, hexagonal...) et son partitionnement (partition régulière, irrégulière, maillage triangulaire, *quad-tree*...).

Cette présentation fait référence à quelques éléments de la théorie des graphes définis dans l'annexe A.

II.1.1 Espaces de représentation

Une image à deux dimensions est souvent représentée par une matrice de dimension $C \times L$ où C correspond au nombre de colonnes et L au nombre de lignes. Chaque point de cette matrice est nommé **pixel** (*picture element*). Le pixel est donc

le représentant le plus fin d'une image, l'élément indivisible de toute image digitale (définition A.35 page 181). Il est caractérisé par un scalaire dans le cas d'images luminances ou monochromes, un vecteur à trois composantes dans le cas d'images couleurs, ou plus si l'on intègre des informations de mouvement, de transparence...

$f : (x, y) \to \mathrm{Y}$	$n = 2, m = 1$	image statique luminance
$f : (x, y, z) \to \mathrm{Y}$	$n = 3, m = 1$	image statique 3D luminance
$f : (x, y, t) \to \mathrm{Y}$	$n = 3, m = 1$	séquence d'images animées luminance
$f : (x, y) \to (\mathrm{R,G,B})$	$n = 2, m = 3$	image statique couleur
$f : (x, y, z) \to (\mathrm{R,G,B})$	$n = 3, m = 3$	image statique 3D couleur
$f : (x, y, t) \to (\mathrm{R,G,B})$	$n = 3, m = 3$	séquence d'images animées couleur
$f : (x, y) \to (V_x, V_y)$	$n = 2, m = 2$	champ dense de vecteurs de mouvement
$f : (x, y, t) \to (V_x, V_y)$	$n = 3, m = 2$	séquence de champ dense
$f : (x, y, z, t) \to (\mathrm{R,G,B})$	$n = 4, m = 3$	image 4D
$f : (x, y, z, t) \to (\alpha, \beta, \dots)$	$n = 4, m = ?$	monde réel

Tab. II.1 – Différents types d'images multivaluées.

La généralisation des différentes types d'images (table II.1) est appelée **image multivaluée** [Gu95], une application d'un espace de départ de dimension n sur un espace d'arrivée de dimension $m \geqslant 1$.

Généralement, les supports des coordonnées des pixels x et y, de la luminance Y et des couleurs rouge R, vert G, bleu B, sont inclus dans $\mathbb{N}$. Les autres variables (t, V_x, α...) sont à support dans $\mathbb{R}$.

Les images luminance. Chaque pixel d'une image luminance (parfois appelée image en niveaux de gris) fait correspondre **l'intensité lumineuse** de chaque point de l'image. L'information au niveau pixel est représentée par un scalaire.

Les images en couleurs. Les standards de vidéo numérique représentent, généralement, les images couleur par l'espace RGB (*Red*, *Green*, *Blue*) ou YUV (luminance, chrominance rouge, chrominance bleue). L'utilisation de l'espace RGB ou YUV est directement liée au système d'acquisition des cameras numériques, et la restitution des images par les écrans de télévision couleur (RGB uniquement dans ce cas). L'espace YUV, obtenu par transformations linéaires de l'espace RGB, représente l'image par sa **luminance** Y (image en niveaux de gris) et ses **composantes chromatiques** U et V.

Ce dernier espace exploite les propriétés de notre système visuel. En effet, nous sommes plus sensibles aux variations de luminance qu'aux variations de teinte[1]. C'est pourquoi, afin de réduire le volume d'informations, il est d'usage de sous échantillonner les composantes U et V. Cette réduction d'information dégrade peu la perception des images naturelles.

[1]Une description du système visuel humain et les représentations normalisées des images couleurs peuvent être consultées dans [All99].

II.1.2 Connexité et interaction spatiale

Afin d'extraire un niveau de représentation de l'image plus élevé, l'information portée par le pixel seul n'est pas suffisante. La définition des interactions spatiales entre plusieurs pixels permet d'introduire la notion de **voisinage**.

En règle générale, **la connexité** fait appel à des notions **d'existance de chemins** dans un ensemble, alors que **l'interactivité** ou **connectivité** correspond plutôt à une idée **d'influence** ou de communication entre deux éléments d'un ensemble.

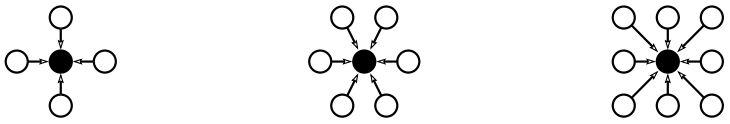
Afin d'exprimer plus formellement ces notions, la structure de l'image est représentée par un graphe régulier orienté ou grille $G = (V, A)$ où V correspond aux nœuds du graphe (les pixels) et A les arcs joignant les nœuds deux à deux. Les coordonnées de chaque pixel déterminent les points de la grille $V \subset \mathbb{N} \times \mathbb{N}$, tandis que les interactions spatiales sont définies par l'ensemble des arêtes $A \subset V \times V$. Dans l'ensemble de ce manuscrit, les coordonnées des pixels sont exprimées dans $\mathbb{N} \times \mathbb{N}$ par (x, y) où l'origine $(0, 0)$ est le pixel en haut à gauche, x est le numéro de colonne, et y est le numéro de ligne.

Comme nous le verrons au cours du chapitre III, le sens des interactions est fondamental, c'est pourquoi le voisinage d'un pixel est défini suivant un concept orienté. **Le voisinage** d'un pixel u dans la grille G est l'ensemble des pixels qui lui sont directement connectés :

$$\forall u \text{ et } v \in V, \quad v \text{ est voisin de } u \Leftrightarrow (u, v) \in A \tag{II.1}$$

où la paire ordonnée (u, v) est l'arc qui relie le pixel u à v. Le voisinage de u, noté $\mathcal{N}(u)$, est défini par :

$$\mathcal{N}(u) = \{v \in V \mid (v, u) \in A\} \tag{II.2}$$



(a) 4-connexité (b) 6-connexité (c) 8-connexité

FIG. II.1 – Mailles d'une image numérique les plus fréquemment rencontrées (en noir le pixel u et en blanc, ses voisins v).

Soit n le nombre de voisins d'un pixel, une image est dite associée à une grille n-connexe, ou simplement n-connexe, si pour tout pixel u, la forme géométrique du voisinage $\mathcal{N}(u)$, *i.e.* la maille de la grille, est invariante par translation (l'image est supposée de taille infinie). Par exemple, l'image est dite 4,6 ou 8-connexe si, pour tout pixel u, $\mathcal{N}(u)$ contient les 4,6 ou 8 voisins les plus proches[1] (figure II.1).

[1]Suivant une métrique Euclidienne par exemple.

(a) Grille 4-connexe (b) Grille 8-connexe

FIG. II.2 – Représentations d'une grille à mailles carrées.

La figure II.2 présente deux connexités (4 et 8) d'une image de dimension 4×4, pour une maille carrée. Ces grilles sont largement utilisées car elles correspondent à la position ligne colonne des capteurs photosensibles des caméras.

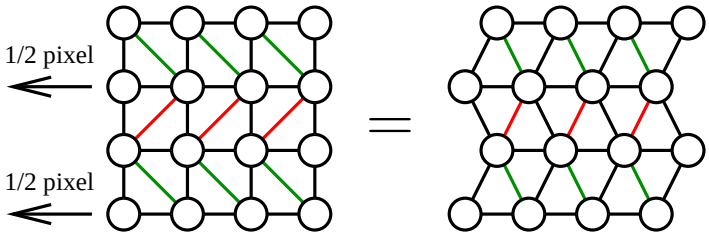


FIG. II.3 – Représentation 6-connexe d'une matrice de points.

Cependant, la trame hexagonale (6-connexe) présente de meilleures propriétés de symétrie. Sa construction, à partir d'une matrice ligne-colonne, est aisée si l'on considère la construction de la grille telle que présentée figure II.3.

II.1.3 Les partitions

Dans la suite de ce document, les partitions et les régions sont définies dans la dimension spatiale. Dans le cas contraire, la dimension temporelle ou spatio-temporelle est précisée.

Une partition (spatiale) est définie à partir de la notion de région (spatiale).

Définition II.1 (Région). *Une région dans une grille G est un ensemble de nœuds tel que pour tout couple de nœuds appartenant à la région, il existe*[1] *un chemin sur cette région.*

Une partition établit des contours qui délimitent un ensemble recouvrant de régions d'une image. Plus formellement :

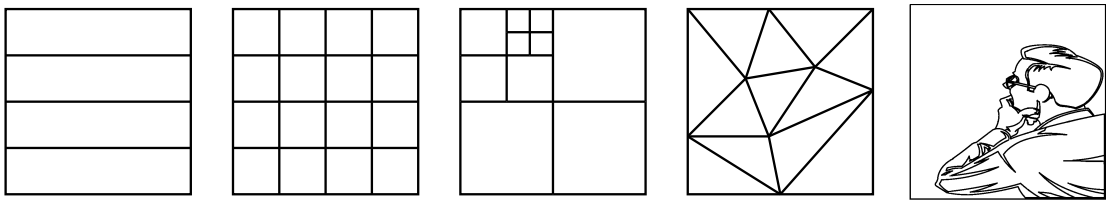
Définition II.2 (Partition). *Une partition $\mathcal{P}$ d'un espace I quelconque est un ensemble de régions $\{R_i\}$ disjointes, dont l'union forme l'espace complet :*

$$\forall i \neq j,\ R_i \cap R_j = \emptyset \tag{II.3}$$

$$\cup R_i = I \tag{II.4}$$

[1]Concept non-orienté ici.

Des notions plus approfondies telles que la relation d'inclusion et la hiérarchie de partitions sont présentées dans [Gom01, chap.4].



(a) 1D-rectilinéaire (b) 2D-rectilinéaire (c) *Quad-Tree* (d) Maillage triangulaire (e) Quelconque

FIG. II.4 – Types de partitionnement allant du plus régulier (du plus simple) au moins régulier (au plus complexe).

Les principaux types de partitionnements (figure II.4) sont :

Les partitions 1D et 2D-rectilinéaires. De par leur simplicité (figure II.4a-b), ces partitionnements sont souvent utilisés.

Le partitionnement 2D-rectilinéaire prend tout son sens lors du codage de la texture de l'image. Par exemple, le système de compression JPEG [R.Z96] [D.S00] utilise des blocs de pixels 8×8 ou 16×16. Cette structure régulière permet des calculs matriciels relativement simples tels que la transformée en cosinus discret (DCT).

Le principal avantage de cette partition est que le décodeur la connaît *a priori*. Le codeur n'a donc pas besoin d'intégrer des informations de contour dans la description de l'image codée.

Quad-Tree. Cette structure, basée sur des régions carrées, prend plus en compte le contenu de l'image, dans le sens signal (figure II.4c). Le partitionnement est codé grâce à un graphe.

Lorsque l'image contient de grandes zones peu texturées, seul un jeu de paramètres permet de coder ces larges régions homogènes, donnant ainsi un gain de codage. Sinon, on retrouve le cas 2D-rectilinéaire régulier mais avec un surcoût dû au codage du graphe.

Maillage. Un maillage est un partitionnement de l'image où chaque région à une forme géométrique quelconque. Un **maillage simple** est constitué de formes géométriques au nombre de côtés constant.

Un **maillage irrégulier** est obtenu si les nœuds sont déplacés. Par exemple, la figure II.4d présente un maillage simple irrégulier à mailles triangulaires.

Régions de forme quelconque. Bien sûr, ce type de partitionnement peut représenter finement les régions d'intérêt d'une image, ou les objets (figure II.4e). Dans le cadre d'une segmentation adaptée au codeur, on essaie d'extraire les régions homogènes suivant certains critères afin de minimiser la redondance entre les jeux de paramètres.

II.2 Codeurs vidéo orienté objet

Les codeurs vidéo orienté objet (codeurs OO) cherchent à représenter le support multimedia sous forme d'entités indépendantes et permettent ainsi une élévation du niveau des informations encodées. Ils sont spécifiés par la norme MPEG-4 [MPE] [MPE95] et qualifiés de **nouvelle génération** par opposition aux codeurs précédents MPEG-1 et MPEG-2 où le codage est fait par traitement monolithique de l'image.

Dans ce manuscrit, nous nous intéresserons uniquement à la partie vidéo de la norme [MPE02] [Rou02], et plus particulièrement à la manipulation des objets.

II.2.1 Composition et segmentation

Comme nous le verrons dans la section II.3.1, la définition d'un "objet" dans une image est un problème complexe. Dans cette partie, nous dirons qu'un objet est un élément reconnaissable par un humain.

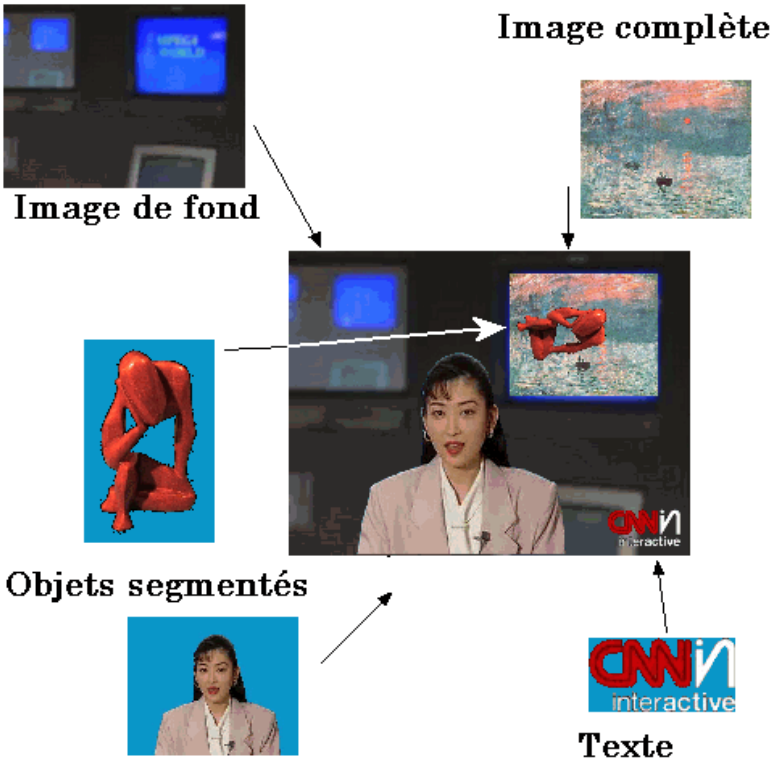


Fig. II.5 – Décodeur MPEG4 : exemple de composition.

Un des atouts des codeurs OO est sa capacité à traiter séparément chaque objet de l'image. Cette dernière est obtenue par **la composition** d'objets [Bre99]. Ainsi, la figure II.5 présente un exemple de composition où différents objets (image de fond, présentatrice, texte...) forment l'image finale centrale. Une des applications possibles est la visioconférence virtuelle, où des conférenciers physiquement éloignés sont réunis autour d'une table virtuelle [SK02].

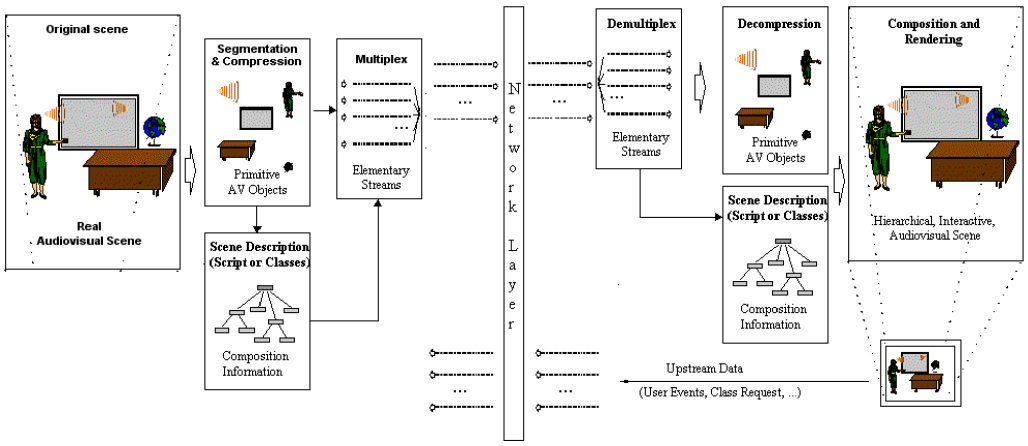


FIG. II.6 – Chaîne de codage et décodage MPEG4 (extrait de [MPE] et modifié).

L'opération inverse de la composition est la **segmentation** des objets et la description de scène : les éléments de l'image sont extraits et pour chacun d'eux, leur position et profondeur dans l'espace sont définies. Dans la figure II.6, cette opération est représentée par symétrisation, selon le schéma classique défini par le groupe MPEG. Dans un cadre de diffusion vidéo, la partie à droite de la couche réseau correspond au décodeur OO, et la partie à gauche au codeur.

Le terminal récepteur effectue le décodage puis recomposition des images. Une description complète de cette partie ainsi que des solutions de composition hiérarchique sont décrites dans [Bre99].

A contrario, le terminal émetteur segmente et code le flux vidéo issu de sa caméra. A notre connaissance, il n'existe pas d'opérateurs capables d'extraire les objets, et encore moins d'établir une description de la scène.

II.2.2 Codage des objets

Pour coder la texture de l'image, les standards précédents MPEG1 et MPEG2 [MPE] utilisent la DCT [ANR74] [Mit97] sur un partitionnement régulier du type 2D-rectilinéaire ou *quad-tree* (§II.1.3). Cependant, ces techniques ne peuvent plus s'appliquer pour un objet puisque le support de la texture est de forme quelconque (figure II.4e page 11).

Supposons que la partition de l'image soit déterminée. À présent, la texture et la forme des différentes régions la composant doivent être codées.

Le codage des contours. Une partition composée de régions de forme quelconque présente les contours les plus complexes à coder.

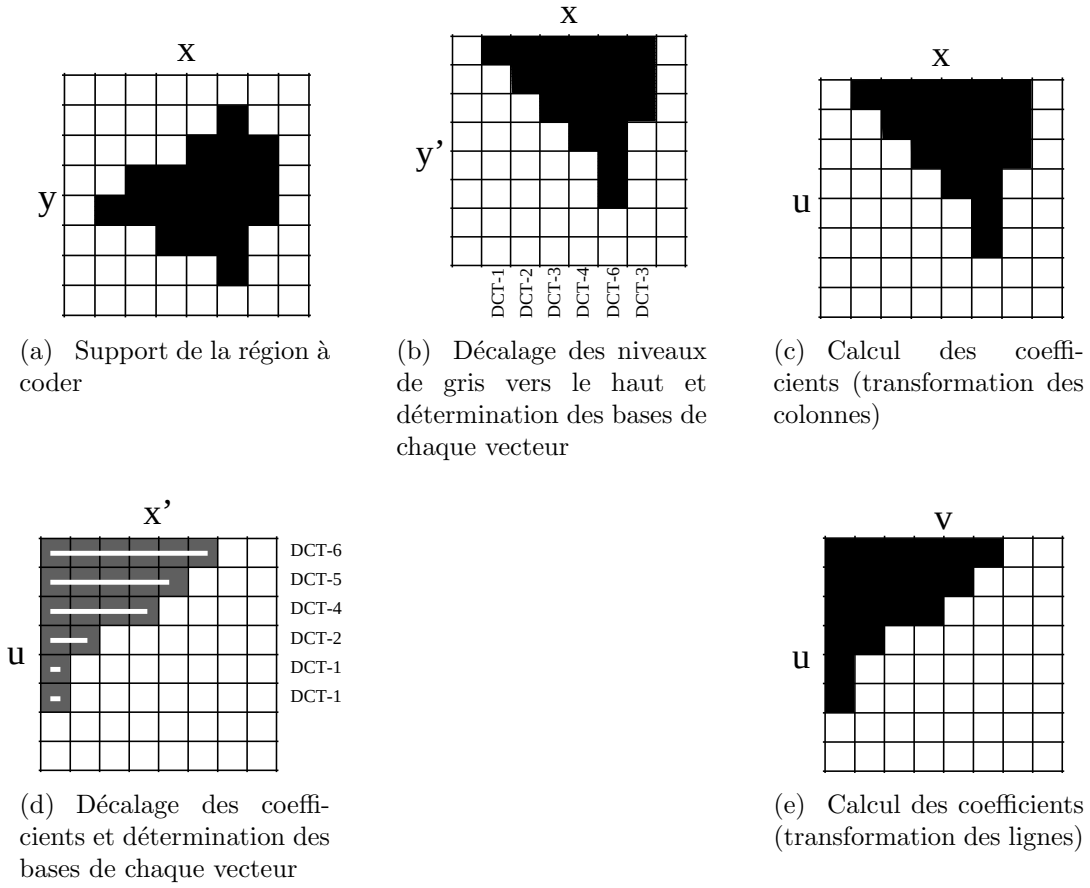
La technique la plus simple consiste à coder le masque de l'objet, *i.e.* une image binaire. D'autres plus complexes codent soit les contours par un lacet (*chain-code*) [Fre74] [LD91], soit effectuent une approximation géométrique des courbes, soit utilisent des contours actifs ou *snake* [DC99] [Del00], soit déterminent des splines [Bri95] ou des segments [Gu95].

Dans des perspectives de réduction du coût de codage, [Mar96, chap.8] montre qu'il est conseillé d'effectuer au préalable un lissage des contours afin d'obtenir de meilleurs taux de compression.

Si le support de l'objet est un maillage triangulaire [Lec99, chap.20], la localisation des différents sommets du maillage est codée par un arbre.

Codage de texture. Dans [PHKU01], un comparatif de différents algorithmes de codage de textures pour des régions de forme quelconque est présenté.

Dans le cadre de la norme MPEG-4, la DCT a évolué vers la SA-DCT (*Shape Adaptive-DCT*, figure II.7) : une version de la DCT s'adaptant à la forme de la région à coder [Sik95] [KS99] [KS98] [SP01].



(a) Support de la région à coder

(b) Décalage des niveaux de gris vers le haut et détermination des bases de chaque vecteur

(c) Calcul des coefficients (transformation des colonnes)

(d) Décalage des coefficients et détermination des bases de chaque vecteur

(e) Calcul des coefficients (transformation des lignes)

FIG. II.7 – Principe d'une SA-DCT.

Un algorithme corollaire à la SA-DCT est la RS-DCT (*Region Support-DCT*) [SS02] où les vecteurs de la base de transformations sont obtenus par fenêtrage et normalisation de ceux d'une DCT classique.

À partir de transformées en ondelettes DWT (*Discret Wavelet Transform*) [SDS96] [Mal89] sur des blocs carrés de pixels, [XLLZ01] et [SBJ98] proposent des transformées par ondelettes basé région SA-DWT (*Shape Adaptive-Discret Wavelet Transform*). Contrairement à la SA-DCT classique, les signaux ne sont pas déphasés, d'où un meilleur taux de compression [XLLZ01].

Dans les cas de partitionnement particuliers comme par exemple le maillage triangulaire, différentes stratégies sont adoptées [VBMZ+99] [Alt96]. Si le contenu

est peu texturé, le modèle d'interpolation de Lagrange est utilisé [LLS99]. Trois coefficients barycentriques situés sur chaque sommet du triangle suffisent pour coder la texture d'une large région. Si le contenu est fortement texturé, une variante de la DCT adaptée à un triangle est alors utilisée [BLDS01] afin de représenter plus fidèlement la texture.

Cependant, ces techniques incluent des informations de formes des régions lors de la détermination des paramètres de texture, d'où une compression moins efficace. Les techniques présentées dans [VD94] tentent de séparer ces deux informations en extrapolant la texture des régions.

II.2.3 Conclusion

Contrairement aux codeurs vidéo de première génération où l'image est codée indépendamment de son contenu, les codeurs vidéo orienté objet tentent de représenter une séquence vidéo par superposition de plans, chacun correspondant à un objet de l'image. Cette représentation implique de nombreuses opportunités (stratégie de codage dépendante de l'objet à coder, codage à très bas débit, scalabilité d'objet et temporelle [Rou02], composition...) et suscite beaucoup d'intérêts (télévision sur le net, terminaux multimedia portables...).

Nous nous intéressons ici qu'au couple composition-segmentation. À partir de masques prédéfinis, les études sur la composition aboutisent sur des solutions viables [Bre99]. Cependant, **il n'existe pas à ce jour de solution satisfaisante pour construire automatiquement ces masques et l'implantation de tels algorithmes dans un terminal portable** constitue un verrou technologique difficile à lever. C'est pourquoi, nous nous focalisons sur la brique élémentaire de segmentation en vue d'une implantation pour un codeur vidéo orienté objet.

II.3 La segmentation

Au cours du chapitre suivant, nous présenterons un nouvel algorithme de segmentation d'images. Afin de le positionner plus facilement parmi l'ensemble des techniques existantes, nous vous proposons ici un état de l'art sur les principaux algorithmes de segmentation.

Définir ce qu'est "la bonne segmentation" est un problème difficile à tout niveau, voire sans solution. La section II.3.1 présente les différentes formes de segmentation et soulève quelques problèmes classiques. Ensuite section II.3.2, nous présentons les principales techniques et stratégies de segmentation. A partir de cet état de l'art général, la section II.4 détaille les solutions proposées par la morphologie mathématique.

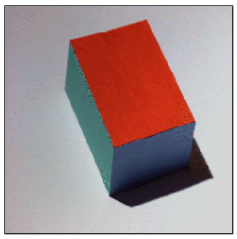
II.3.1 Le paradigme de la segmentation

Suivant l'encyclopédie universalis, on dira que deux unités u et u' appartiennent à un même paradigme si et seulement si elles sont substituables dans un même syntagme (groupe de morphèmes, *i.e.* formes minima douées de sens, formant une

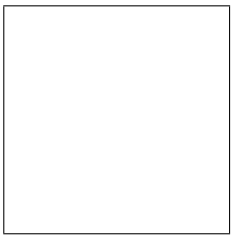
unité), c'est-à-dire s'il existe deux syntagmes vuw et $vu'w$; le paradigme de u est alors défini comme l'ensemble des unités qui auraient pu apparaître à sa place.

La segmentation consiste à déterminer les régions vérifiant certains critères d'intérêts. Cependant quels critères doit-on employer afin de quantifier et paramétrer cette notion "d'intérêt" ? Ainsi pour une même image, il existe un paradigme de chaque région u d'une partition car généralement, la définition du critère précité est imprécise, d'où une segmentation multiple.

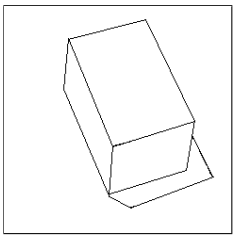
Par ailleurs, la notion même de région peut revêtir plusieurs formes : un ensemble connexe de pixels (segmentation spatiale), un ensemble d'images successives d'une séquence vidéo (segmentation temporelle), ou les deux (segmentation spatio-temporelle : suivi d'objets par exemple).



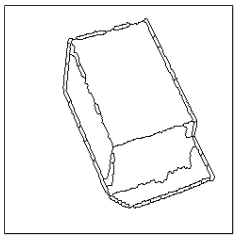
(a) Image originale



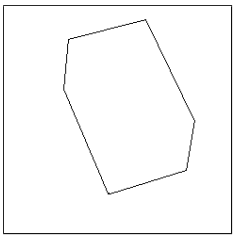
(b) Segmentation au sens du mouvement (pavé immobile)



(c) Segmentation spatiale manuelle,



(d) automatique basée sur le contraste,



(e) sémantique (extraction de l'objet)

FIG. II.8 – Paradigme de la segmentation : quel partitionnement adopter ?

Suivant l'objectif de l'outil de segmentation, chacune de ces partitions peut appartenir à l'une des deux catégories de segmentation suivantes :

La segmentation non-sémantique : Elle génère un ensemble de régions connexes vérifiant un certain critère d'homogénéité, sans se soucier de la signification des objets de la scène. Les régions n'ont, en général, pas de significations pour l'oeil humain.

Dans le cadre d'une segmentation en vue du codage, "l'intérêt" se porte sur la **réduction maximale du coût de codage** [Mar96] [MN98]. Afin d'obtenir un taux de compression maximum, la segmentation recherchée doit déterminer de larges régions pouvant être codées par un même jeux de paramètres, comme par exemple la texture, le mouvement (figure II.8b) [Gel98] [Pat98], *etc.*

La segmentation sémantique : C'est ce qu'effectue l'être humain pour extraire les objets du fond. Dans l'exemple de la figure II.8, les lignes séparant deux

régions différentes pourraient être les arêtes du pavé (figure II.8c) et le bord de son ombre portée. Cependant, nous venons d'introduire les informations "pavé" et "ombre" qui ne sont pas explicitement contenues dans l'image. C'est pourquoi, une segmentation automatique (figure II.8d) détermine généralement de façon insatisfaisante les frontières entre les régions (fusion de régions voisines insuffisamment contrastées, sursegmentation d'une même région...).

Un type de segmentation d'une complexité plus élevée consiste à extraire l'objet de l'image (figure II.8e). Cette classe d'algorithmes aborde la reconnaissance d'objets [Lev02]. En particulier, l'algorithme de segmentation doit savoir fusionner les régions appartenant à un même objet. Une solution intermédiaire consiste à introduire des informations *a priori*, comme par exemple dans [GM00] et [KH02], où la segmentation de l'image cherche à extraire le locuteur d'une séquence visiophonique.

Dans un cadre plus général, une autre alternative consiste à intégrer dans le processus de segmentation une interaction humaine : l'algorithme de segmentation devient alors semi-automatique [ZMM99] [GL98] [LMM02].

Dans la suite de ce document, nous faisons la distinction entre une **région**, qui est une composante connexe de l'espace définie généralement par un **critère d'homogénéité** (segmentation non-sémantique), et un **objet**, qui est une composante de l'espace respectant la **sémantique** de l'élément physique extrait de la scène (segmentation sémantique). Un objet peut être constitué d'une ou plusieurs régions.

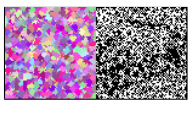
Dans un cadre général, le paradigme de la segmentation cherche à extraire une information de haut niveau à partir d'une importante quantité d'informations de bas niveau.

Pour une même image, on distingue plusieurs niveaux de représentation. Le plus élevé peut se présenter, par exemple, sous forme d'un script cinéma, c'est-à-dire que l'image est décrite par une liste d'objets situés dans une scène (ventail droit de la fenêtre ouverte, lampe de chevet allumée à gauche du lit...). La quantité d'informations contenue dans chacun des éléments de cette liste est importante car le contenu sémantique de l'image y est décrit.

A contrario, le niveau le plus bas représente l'information contenue au niveau d'un pixel : intensité lumineuse, couleur, vitesse de déplacement...

Orthogonalement à ces notions, deux modes de construction des images sont distingués : **les images naturelles** acquises par un dispositif numérique (caméra, scanner...), et **les images artificielles** construites suivant des règles bien précises (lois mathématiques, dessins, répétition d'une texture...).

Les images naturelles permettent une évaluation subjective ou qualitative des caractéristiques d'un algorithme de traitement d'images (partitions générées cohérentes, image pas trop dégradée...), alors que les images artificielles sont plutôt destinées à une estimation objective ou quantitative (le carré bruité est-il bien délimité, quelle est la vitesse de déplacement des pixels d'un cercle blanc...).



II.3.2 Les stratégies de segmentation

Cette section présente les principales techniques connues pour extraire les régions d'une image [SM99]. La suite de ce mémoire fait de nombreuses références à la morphologie mathématique, c'est pourquoi une description plus détaillée de ces techniques est présentée dans l'annexe B.

Dans le domaine de la segmentation vidéo, différentes stratégies sont utilisées. Des méthodes de segmentation du mouvement utilisant des champs de Markov sont présentés dans [Gel98] [Lie00]. D'autre méthodes telles que présentées dans [Pat98] effectuent une segmentation spatio-temporelle optimale afin de minimiser le coût de codage. Les méthodes exploitant les outils issus de la morphologie mathématique sont présentées dans [Mar96] [GS98] pour une segmentation par fusion de régions et [SM99] [Gom01] pour le suivi d'objets grâce à un graphe.

Comme la segmentation vidéo requiert généralement un partitionnement d'origine, la segmentation de la première image, nous nous focaliserons sur les stratégies de segmentation d'images fixes.

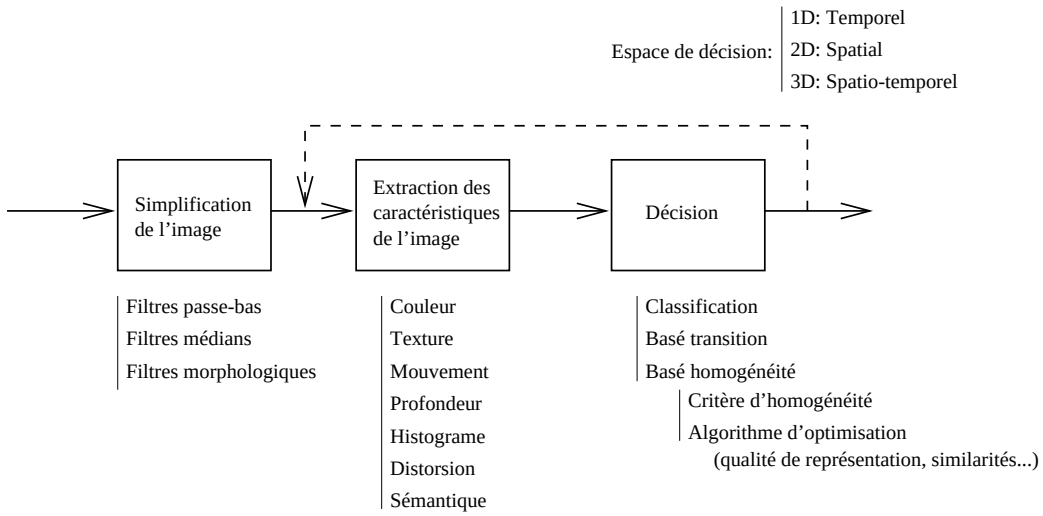


FIG. II.9 – Principales étapes de segmentation et leurs choix correspondants [SM99].

La segmentation (figure II.9) peut être globalement vue comme la succession de trois étapes : la simplification de l'image (§II.3.2.1), l'extraction des caractéristiques de l'image (§II.3.2.2) et la décision (§II.3.2.3).

II.3.2.1 La simplification de l'image

En général, les images naturelles sont bruitées et de ce fait perturbent les algorithmes d'analyse d'image. Pour améliorer les performances des opérateurs de segmentation, les chaînes de segmentation effectuent généralement différentes transformations afin de faire ressortir les informations pertinentes recherchées en atténuant les informations parasites (le bruit).

La simplification de l'image est donc une succession de prétraitements destinés à construire, à partir d'une image d'entrée bruitée, une image adaptée à l'algorithme

d'extraction des caractéristiques (§II.3.2.2).

Les principaux opérateurs de simplification, ou filtres, sont les suivants :

Les filtres de convolution passe-bas. Ces filtres consistent à calculer, pour chaque pixel, la moyenne pondérée de l'ensemble des pixels voisins de ce pixel. Le voisinage et la pondération est déterminée par la matrice de convolution (par exemple un débruitage léger est obtenu grâce à la matrice (II.5), un filtre passe-bas gaussien de dimension 3×3).

$$\omega = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{II.5}$$

Ces filtres sont d'une simplicité remarquable, et sont généralement utilisés dans les premières étapes de traitement d'image afin de lisser légèrement l'image.

Les filtres médians. Ces filtres non-linéaires sont particulièrement adaptés à la réduction du bruit impulsionnel.

Les filtres morphologiques. Ces filtres (reconstructions géodésiques, filtres aréolaires, nivellements morphologiques...) permettent de conserver l'information de contour tout en simplifiant l'image [Gom01, chap.3] [Vac95] (annexe B.4 page 188). Ils sont pour cette raison bien adaptés au préfiltrage de l'image en vue de sa segmentation.

A cette étape de la chaîne de segmentation nous disposons d'une image où l'information jugée non-pertinente (le bruit) est atténuée.

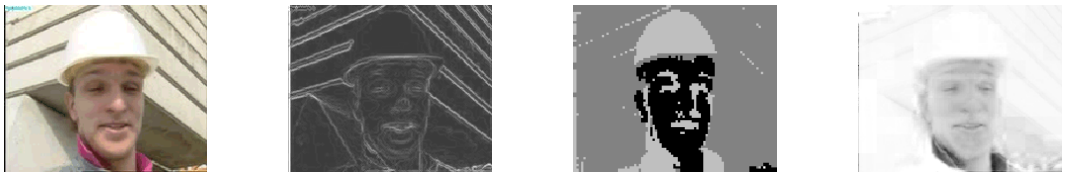
II.3.2.2 L'extraction des caractéristiques

L'extraction des caractéristiques consiste à quantifier des paramètres d'homogénéité suivant divers critères dépendants de l'application visée.

Dans certains cas particuliers, les données procurent directement l'espace des caractéristiques nécessaires pour la segmentation. C'est le cas par exemple pour l'extraction d'un présentateur de télévision placé devant un écran de chromaticité constante bleue : la couleur des pixels correspond aux caractéristiques recherchées.

Le calcul du gradient. Cet opérateur permet de mettre en avant les zones de transition (figure II.10b). Une faible valeur du gradient en un point correspond à une homogénéité locale tandis qu'une forte valeur correspond à une variation prononcée de l'information.

Une description détaillée du gradient morphologique est présentée dans l'annexe B.3.3 page 187. [ANB01] propose un gradient multiéchelles couleurs pour la segmentation de séquences vidéo.



(a) Image RGB originale

(b) Image gradient : dédiée à la recherche des discontinuités

(c) Classification des pixels en 4 classes suivant leur chromaticité

(d) Loi logarithmique : pixels de couleur chair en gris moyen

FIG. II.10 – Exemples d'extraction des caractéristiques.

L'analyse de couleur. Suivant le type de segmentation recherché, il est parfois utile de mettre en avant les pixels respectant un certain critère de couleur. Par exemple, l'extraction semi-automatique d'une personne nécessite généralement la détection de son visage [Rou02], [Gom01], [Lie00]. Une large enquête sur ce sujet est présentée dans [YJA02].

Loin d'être exhaustif, nous pouvons citer les principales méthodes d'analyse de couleur :

- **La modélisation de l'espace UV :** En général, la technique consiste à utiliser la distribution chromatique caractérisant la peau humaine. Celle-ci peut-être modélisée par une fonction de distribution gaussienne bidimensionnelle. Une carte des probabilitées est alors dressée [Gom01]. Dans [Rou02], [Wan01], cette statistique est à la base d'un partitionnement de l'espace UV en classes (figure II.10c).
- **La transformation logarithmique :** Ce prétraitement effectue une transformation logarithmique de l'image. L'image obtenue (figure II.10d) caractérise les pixels à teinte de chair par un niveau de gris localisé [Lie00].

Une présentation plus complète sur la couleur à des fins de segmentation peut-être consultée dans [Hur89], [YMS99] ou [YM01].

L'extraction des caractéristiques est parfois réalisée sur le support d'une région, c'est pourquoi une boucle issue de l'étape de décision décrite ci-après est introduite dans le procédé de segmentation. L'estimation dépend alors des résultats de segmentation de l'itération précédente (algorithme de suivi dans [GM00], segmentation d'une route [BB94], fusion de régions suivant des paramètres de texture [Mar96], détection de visage [Rou02]...).

II.3.2.3 La décision

La section précédente présentait les principales techniques d'extraction des caractéristiques de l'image. Nous étudions la troisième étape (schéma de la figure II.9) de la chaîne de segmentation : l'analyse de l'espace des caractéristiques afin d'obtenir une partition des données.

L'étape de décision détermine la position des frontières afin de former les partitions dans l'espace des décisions. Chaque région obtenue est homogène suivant un critère fixé. Par exemple, pour une segmentation spatiale, chaque région vérifie un

certain critère de texture. Pour une segmentation temporelle, chaque région détermine un plan de la séquence d'images et les frontières sont les instants où la scène a changé.

Étudions les trois principales stratégies de construction des frontières.

La classification. Les techniques de classification organisent un ensemble de données en différentes classes.

La technique la plus simple procède par **seuillage**. L'image binaire (composée de deux classes) obtenue classifie les pixels appartenant au fond et ceux appartenant aux objets. La difficulté réside évidemment dans le choix du seuil (s'il existe). Cette technique est malheureusement très sensible aux bruits et ne peut être employée que dans des applications spécifiques. Des techniques utilisant une logique floue proposent des solutions à ce problème [Bra95].

Les algorithmes de partionnement de l'espace UV font également partie de cette catégorie d'algorithmes [Rou02].

La segmentation basée sur l'homogénéité. Les algorithmes par **fusion de régions** et par **croissance de région** font partie de cette technique. A partir d'une segmentation originale fine (un pixel dans le cas d'une segmentation spatiale ou une image dans le cas d'une segmentation temporelle), les premières consistent à fusionner les régions voisines respectant un certain critère d'homogénéité [Mar96] [LD99] [LC02] [GER00] [SZC+00]. Les secondes font croître des marqueurs locaux (graines) tant qu'un certain critère d'homogénéité est respecté [VD94].

Le critère d'homogénéité est généralement décrit par des :

- **paramètres stochastiques.** Les partitions sont obtenues par le calcul de probabilités statistiques. Cette méthode détermine les régions de pixels ayant une statistique proche. La théorie des champs aléatoires de Markov (MRF) est employée dans [Lie00], [CP96]. On trouvera dans [KB99] une analyse détaillée de ces méthodes ;
- **paramètres de texture couleur.** Une méthode de segmentation nommée JSEG [Wan98], [Wan01] procède en deux étapes indépendantes : une quantification de la couleur (extraction des caractéristiques) et une segmentation spatiale par croissance de germes ;
- **paramètres de texture luminance.** Cette méthode, plus classique, tente de regrouper les régions voisines présentant des paramètres de texture équivalents. Ces paramètres peuvent être obtenus par DCT, polynômes de faible degrés, ondelettes, coefficients de Lagrange...

La segmentation basée sur les transitions. Ces techniques tentent d'estimer la position des discontinuités, préalablement déterminées par un calcul de gradient durant l'étape d'extraction, contenues dans l'espace des caractéristiques.

Le principal inconvénient des techniques basées sur les transitions est leur manque de robustesse [SM99]. Une erreur locale peut impliquer des erreurs de segmentation significatives. Plusieurs propositions tentent d'accroître la robustesse en améliorant l'étape d'extraction des caractéristiques [ANB01] [Vac95] [YM01] [Lie00] [Gu95] [GL98]. D'autres techniques, tels que les contours actifs, permettent de minimiser

les effets des erreurs locales d'une image gradient.

Certaines méthodes font appel à plusieurs stratégies, ainsi la technique proposée dans [Bon98] allie les deux techniques basées sur les transitions et l'homogénéité, elle est dédiée au suivi d'objets dans une séquence d'images.

Enfin, l'algorithme de segmentation par construction des lignes de partage des eaux (LPE) est une technique de localisation des transitions par **croissance de région** (*region-growing*), issue de la morphologie mathématique. Une version du type division-fusion (*split-and-merge*) est proposée dans [DVG94], et une variante exploitant la topologie de l'image dans [GDB00].

Le chapitre III développe et justifie le choix d'un des algorithmes de construction des LPE, c'est pourquoi la section II.4 suivante établit un état de l'art plus détaillé sur cette technique de segmentation.

II.4 Les algorithmes de ligne de partage des eaux

La ligne de partage des eaux (LPE) est l'outil de segmentation par excellence en morphologie mathématique. Contrairement à ce que l'on pourrait penser, ce concept n'a pas ses origines dans le développement d'opérateurs morphologiques, mais est issu de la topographie et de l'hydrogéologie.

Nous devons ces algorithmes par croissance de région, entre autres, à S. Beucher, C. Lantuejoul [BL79] et F. Meyer [Mey91]. C'est une méthode ascendante (*bottom-up*) qui consiste à faire grossir des germes (points source) suivant des critères de distances.

La présentation proposée est intuitive, l'annexe B.6 page 195 présente plus formellement les différentes méthodes de construction des LPE. Une enquête sur les différentes méthodes est présentée dans [RM01] et [Nog98]. Pour une description plus détaillée, les références sont :

- [Beu90] pour l'algorithme de LPE par fléchage ;
- [Beu94] pour l'algorithme des cascades ;
- [Mey91] pour l'algorithme d'inondation suivant une file d'attente hiérarchique ;
- [Ser00] et [Ser88] pour l'algorithme d'inondation par calcul des squelettes par zone d'influence ;
- [Mey94] pour l'algorithme par calcul de distances topographiques.

Dans ce manuscrit, nous employons le terme «luminance» indifféremment pour désigner le niveau de gris d'une image originale et l'amplitude d'une image gradient.

Les algorithmes de LPE s'appliquent généralement[1] à une **image gradient**, considérée comme **un relief topographique**. La luminance correspond à l'altitude de chaque pixel : un pixel sombre (faible gradient) sera de faible altitude, alors qu'un pixel clair (fort gradient) sera à une altitude plus élevée. C'est pourquoi, au cours de cette présentation, nous faisons souvent référence à des termes issus de la

[1]Dans certains cas, il est utile d'effectuer une LPE sur l'image originale. Par exemple dans [Beu90, chap.6], une telle méthode est utilisée pour construire un marqueur pour le fond.

géomorphologie.

Ici, seules les techniques de segmentation basées **sans marqueur** sont présentées. Un marqueur est un ensemble connexe de pixels initiateur d'une région de l'image qui peut-être décorrélé de la topographie de l'image. Une description plus générale des techniques basées marqueur peut-être consultée dans [Mog97].

Pour l'ensemble des algorithmes présentés, chaque minimum correspond à une graine. Un minimum d'une image est un ensemble connexe de pixels tel qu'il n'existe pas de chemin descendant[1] (au sens large) joignant un pixel de cet ensemble vers un pixel d'altitude strictement inférieure.

II.4.1 Algorithmes par calcul de distances

Le principe est de déterminer les bassins d'attraction (voir définition B.35 page 194) de chaque pixel selon une certaine distance. Un pixel non-minimum est étiqueté par le minimum le plus proche : son minimum d'attraction. S'il se trouve à équidistance de plusieurs minima, c'est un pixel ligne de partage des eaux : il est étiqueté W (*watershed*).

II.4.1.1 Squelettes par zone d'influence géodésique

L'image d'entrée étant binaire, cet algorithme [Mey87] détermine les points qui sont géodésiquement à équidistance de plusieurs zones marquées (figure B.7 page 194). Le squelette de l'image ainsi obtenue est une segmentation de l'image d'entrée.

II.4.1.2 Bassins d'attractions suivant la distance topographique

La distance topographique (définition B.33 page 193) [Mey94] s'applique sur des images luminance, elle mesure les variations topographiques du relief formé par l'image. De même, un bassin d'attraction est déterminé par l'ensemble des pixels attirés par un même minimum.

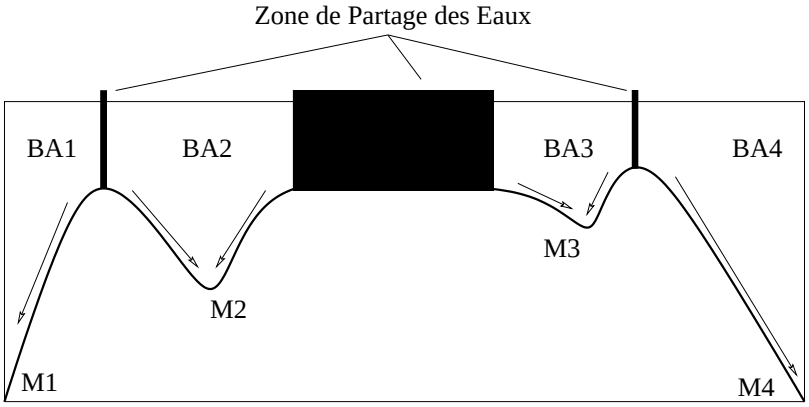
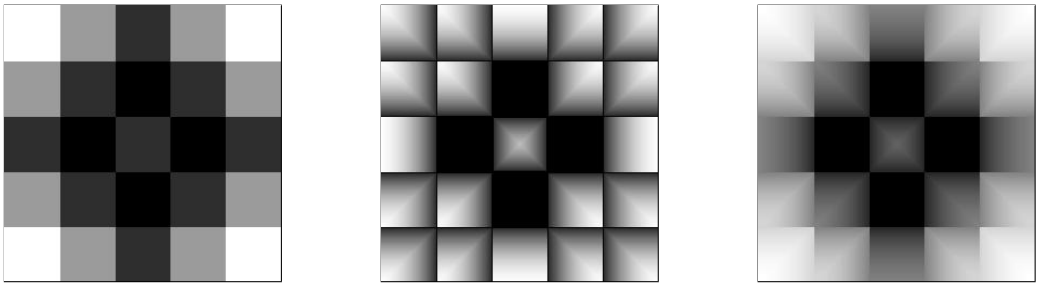


FIG. II.11 – Zone de Partage des Eaux monodimensionnelle (M : minimum, BA : bassin d'attraction).

[1] Suite ordonnée de pixels tel que leur altitude soit décroissante (définition A.36 page 181).

Cependant, la distance topographique n'est pas définie à l'intérieur des plateaux. Si elle est posée nulle pour tout pixel n'ayant aucun voisin d'altitude strictement inférieure, alors un algorithme de segmentation utilisant une telle distance construit des **zones de partage des eaux** (ZPE) pouvant être très larges (figure II.11). Ces zones localisent les lieux où la LPE d'épaisseur nulle ou un ne peut-être déterminée de façon unique. Toute ligne située sur une ZPE peut-être considérée exacte.



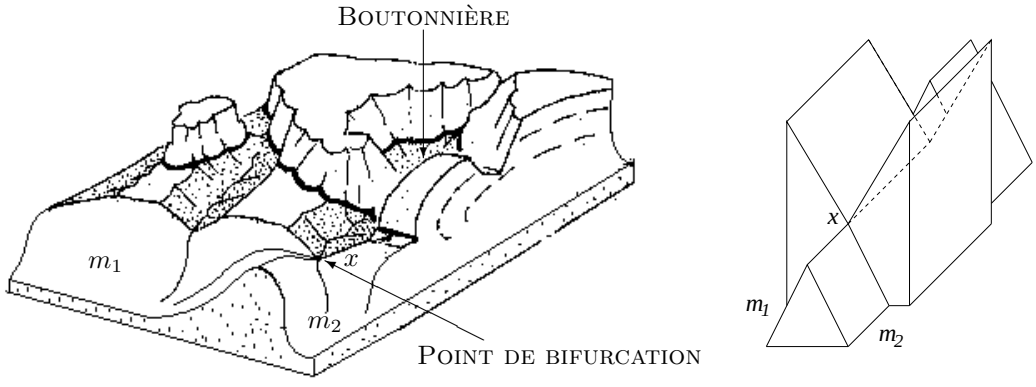
(a) Image 500 × 500 constituée de plateaux

(b) Image distance inférieure

(c) Image semi-complète inférieure

FIG. II.12 – Exemple de transformation d'une image en une image semi-complète inférieure (extrait de [RM01]).

Si des LPE d'épaisseur un pixel sont cherchées, le squelette des ZPE est une solution possible pour les déterminer. Une autre alternative consiste à transformer auparavant l'image en une image semi-complète inférieure (*lower complet image*, figure II.12) [RM01] [Mog97] [BM98] [MR98]. La luminance des pixels de l'image distance inférieure (figure II.12b) correspond à la distance géodésique séparant le pixel d'un plateau à sa frontière inférieure (définition III.3 page 34).



(a) Relief géographique

(b) Relief synthétique

FIG. II.13 – Boutonnière et point de bifurcation.

Une seconde configuration topographique engendrant des ZPE est **la boutonnière** (figure II.13)[1], connue également sous le nom de fenêtre en géomorphologie. Il est démontré dans [Mey94] que la ligne de plus grande pente minimise la distance topographique. Donc tout chemin de coût minimal joignant un point de la boutonnière à m_1 ou m_2 (figure II.13b) passe par le point x. Si le coût de x à m_1 est

[1]Illustration extraite et modifiée de `http://www.home.ch/~spaw2988/0_accueil/glossaire_geomorpho.html`.

identique à celui de x à m_2, alors le coût sera également identique pour tout point de la boutonnière, engendrant ainsi une ZPE. Ce point x est appelé **point de bifurcation**.

II.4.1.3 Algorithme par ruissellement

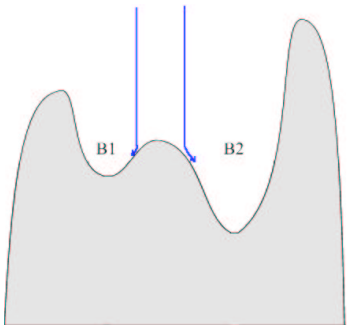


Fig. II.14 – Algorithme par ruissellemment.

Cet algorithme interprète aussi l'image gradient à segmenter comme un relief topographique (figure II.14) : imaginons qu'il pleuve et suivons le parcourt de chaque goutte d'eau. La nature respecte la loi du moindre effort, c'est-à-dire que l'eau coule suivant la ligne de plus grande pente, ligne où le gradient local est le plus élevé. Si toutes les gouttes, pour un point précis du relief, sont attirées par un même minimum, alors ce point reçoit l'étiquette du minimum d'attraction. Au contraire, si certaines gouttes associées à un point sont attirées par un minimum et d'autres gouttes du même point sont attirées par un autre minimum, alors ce point est répertorié comme un point de partage des eaux.

Cet algorithme n'est pas recommandé car il est fortement biaisé [Beu00], il génère des ZPE (*cf* boutonnière, figure II.13) et il est de complexité algorithmique élevée.

II.4.1.4 Algorithme de fléchage ou par ascension de colline

Les algorithmes de fléchage cherchent à déterminer une relation d'ordre des points avec leurs voisins immédiat. Contrairement à l'algorithme par ruissellement, les flèches sont orientées dans le sens ascendant. Un pixel initie autant de flèches qu'il a de voisins d'altitudes strictement supérieures.

L'algorithme de segmentation par ascension de colline (*Hill-Climbing*) est un algorithme de fléchage qui ne conserve, pour tout pixel, que la flèche issue du voisin d'altitude la plus basse (toutes les autres sont supprimées). Il propage l'étiquette de chaque minimum (un identifiant unique préalablement attribué) suivant une ligne de plus grande pente.

Supposons que le relief soit issu d'une image semi-complète inférieure, c'est-à-dire qu'en tout point non-minimum, il existe au moins un point voisin d'altitude strictement inférieure. L'algorithme séquentiel consiste à [Mey94] :

- localiser les minima locaux ;
- attribuer à chaque minimum une étiquette différente ;
- propager les étiquettes suivant une ligne de plus grande pente ;

– arrêter le processus dès que tous les pixels sont étiquetés.

Afin de définir la notion de pente dans une image numérique, F. Meyer a défini une fonction de coût [Mey94] (équation B.34 page 193). La ligne de plus grande pente est alors le chemin strictement ascendant (*i.e.* dans le sens strictement croissant des altitudes) joignant deux points du relief tel que cette fonction de coût soit minimisée.

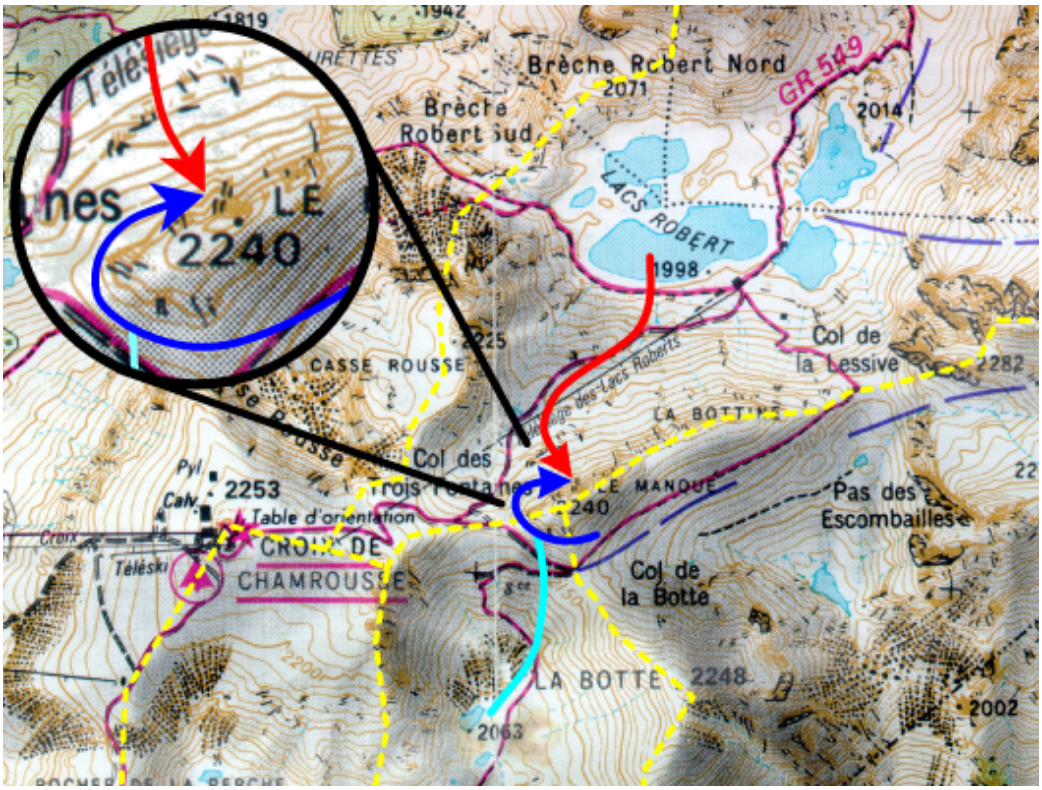


FIG. II.15 – Si les étiquettes ne se propagent pas suivant la ligne de plus grande pente, un pixel situé sur le bassin d'attraction `Lacs Robert` peut être étiqueté par un autre minimum (flèche cyan `La Botte` ou bleue `Pas des Escombailles`).

La figure II.15 présente intuitivement l'intérêt de suivre la ligne de plus grande pente. Cette ligne, représentée par la flèche rouge partant des `Lacs Robert`, forme un angle droit avec toutes les lignes de niveau qu'elle intersecte. En effet, si le coût de passage d'un point vers son voisin d'altitude supérieure n'est pas minimisé (flèches bleues de chaques côtés de `la Botte`, certains chemins de randonnée marqués en violet foncé...), le minimum d'une région peut étiqueter les pixels appartenant à une région voisine. La localisation des zones de transition serait alors biaisée.

Si l'image comporte des plateaux non-minima, il est impossible de déterminer une ligne de plus grande pente en ces lieux : il n'existe pas de direction de propagation privilégiée puisque la pente est nulle. Ce problème est résolu en supprimant préalablement ces plateaux par une transformation semi-complète de l'image ou en traitant les points du plateau non-minimum dans l'ordre croissant de leur distance géodésique par rapport à la frontière inférieure du plateau[1] [Mey94] [BM00].

[1]Points d'un plateau ayant au moins un voisin d'altitude strictement inférieure (définition B.28

Les deux principales propriétés de cet algorithme sont :

1. Une connaissance locale des données est suffisante pour simuler la propagation de l'eau issue des pixels minima vers tous les pixels non-minima (à l'exception des pixels situés à l'intérieur d'un plateau non-minimum).
2. Les lignes de partage des eaux sont d'épaisseur nulle : la frontière est située entre deux points d'étiquette différente.

La totalité de la boutonnière est étiquetée par l'étiquette du point de bifurcation.

II.4.2 Algorithmes par inondation

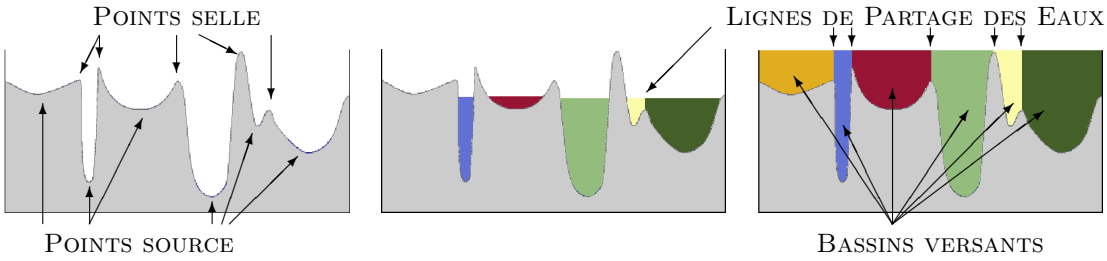
Le principe général de ces algorithmes consiste à traiter une image luminance par plans successifs, des niveaux de gris les plus faibles, vers les plus élevés. Le terme **inondation** est issu de l'interprétation de ces plans comme un niveau d'eau envahissant progressivement un relief topographique.

II.4.2.1 Processus d'inondation uniforme

Le processus de construction des lignes de partage des eaux par inondation uniforme s'effectue ainsi :

- L'algorithme perce un trou au niveau de chaque minimum.
- Il plonge ce relief dans l'eau qui s'introduit par le minimum global, les **points sources** d'altitude minimum : des lacs commencent à se former au niveau des minima les plus bas.
- Chaque minimum colore son eau d'une couleur unique. Dans la suite de ce document, l'étiquette d'un pixel fait référence à cette couleur.
- Lorsque deux eaux se rencontrent, un barrage est construit afin d'interdire tout mélange des couleurs.
- Lorsque le niveau d'eau atteint le sommet du relief, le processus de segmentation est terminé.

L'ensemble des barrages constitue les lignes de partage des eaux. La coloration de l'eau interdit toute construction de barrages, appelés **ligne de partage locale** [Beu90, chap.4], entre deux eaux issues d'un même minimum [VS91].



(a) Relief à inonder (b) Inondation en cours (c) Partitions finales

Fig. II.16 – Processus d'inondation monodimensionnel du relief afin de déterminer les LPE (extrait de [Gom01, chap.4]).

page 192).

La figure II.16 illustre un processus 1D d'inondation. Les lignes[1] de partage des eaux sont obtenues à la figure II.16c.

Ce processus peut déterminer soit des LPE d'épaisseur nulle (la ligne est située **entre** deux pixels d'étiquette différente), ou d'épaisseur un (le pixel d'altitude maximum ou le pixel intérieur n'ayant aucun voisin LPE, reçoit l'étiquette W=*Watershed*). L'ensemble des pixels associés à une couleur décrit le **bassin versant** associé au point source. L'image des lacs en pseudo-couleur décrit l'ensemble des régions.

Les implantations efficaces de cet algorithme ordonnent les pixels à traiter dans une FIFO (*First In, First Out*), ou dans une liste ordonnée de FIFO : **la file d'attente hiérarchique FAH** (chaque FIFO possède une priorité). L'avantage de cet algorithme est qu'il construit par définition des contours fermés, et que le processus d'inondation par une FAH est optimal (tout pixel non-minimum n'est traité qu'une seule fois).

Cependant ce processus est par nature global (niveau d'eau identique pour tous les bassins) et séquentiel (traitement des pixels dans l'ordre croissant de leur niveau de gris), et requiert des accès mémoire vers des données éloignées (traitement des pixels ayant le même niveau de gris).

II.4.2.2 Squelettes par zone d'influence

Cet algorithme est l'extension aux images luminance de l'algorithme présenté à la section II.4.1.1. Il entre dans la classe des algorithmes par inondation car un processus de recurrence met en œuvre les zones d'influences pour chaque niveau successif (§B.6.1 page 195).

Cet algorithme fait partie des algorithmes les moins biaisés. De par la construction de squelettes, il génère une LPE d'épaisseur un (les pixels LPE reçoivent l'étiquette particulière W). Cependant, il présente une complexité algorithmique non négligeable.

II.4.2.3 Algorithme par graphe des composantes.

Cet algorithme est une variante de l'algorithme présenté ci-dessus, où le processus s'effectue non pas dans une image, mais dans un graphe.

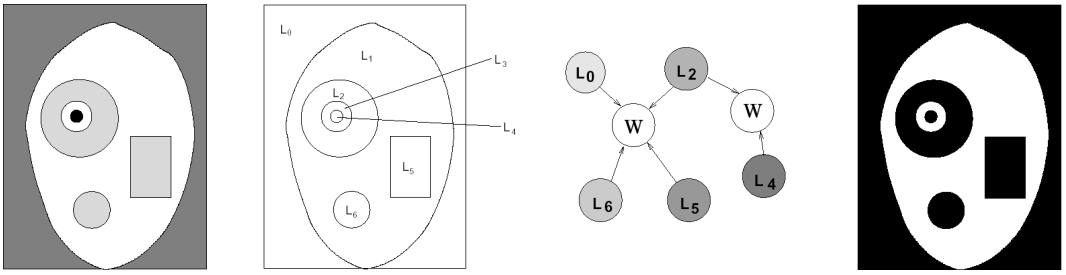
Une composante de l'image est un plateau composé d'au moins un pixel. L'algorithme [MR95] procède en trois étape :

1. **Transformation de l'image en un graphe orienté valué**[1]**.** Soit un graphe où chaque nœud correspond à une composante de l'image et possède une étiquette unique. Une arête[2] entre deux nœuds est crée si leurs composantes se touchent dans l'image.
2. **Inondation du graphe.** De façon similaire à l'algorithme par inondation (§II.4.2), l'étiquette de chaque minimum de l'image est propagée dans le graphe

[1]On devrait plutôt parler de "points" pour un processus monodimensionnel.
[1]Consulter l'annexe A page 177 pour une présentation générale des terminologies.
[2]Lien entre deux nœuds, concept non-orienté (définition A.3 page 177).

suivant l'algorithme dit de recherche en largeur[3]. Si un nœud quelconque est accessible par deux autres nœuds d'étiquettes distinctes, alors il est étiqueté W : c'est une composante de partage des eaux. Sinon, il est inondé : il reçoit l'étiquette de ses nœuds voisins.

3. **Construction des partitions.** La propagation terminée, le graphe est transformé en une image binaire localisant les zones de partage des eaux : chaque pixel est fixé à un si son nœud associé est étiqueté W, zéro sinon.



a) Image à segmenter.

b) Composantes étiquetées.

c) Graphe des composantes après inondation ("W" substitue les étiquettes L_1 et L_3).

d) Image binaire de sortie (les ZPE sont en blanc).

FIG. II.17 – Algorithme de segmentation par graphe des composantes [MR95].

Cet algorithme construit des zones de partage des eaux car une composante de partage des eaux peut être un plateau (figure II.17d).

Cet algorithme permet une implantation parallèle du processus d'inondation du relief [MR95]. Cependant en vue d'une implantation, il peut exiger des ressources matérielles importantes car la taille du graphe est directement liée au nombre de composantes. De plus dans ce cas, la répartition des charges est délicate pour une architecture au nombre de processeurs limité.

II.5 Conclusion

Le codage orienté objet de séquences vidéo est un nouveau standard de codage des supports multimedia. Son principal attrait réside dans sa capacité à différencier les différents objets (image, objet sémantique, son, texte...) qui composent le film, source de nouveaux services (qualité de codage variable suivant l'objet et codage très bas débit, composition...).

L'étape de segmentation est une brique de base fondamentale pour un tel codeur car elle extrait les objets de la scène. Elle constitue un problème complexe à tout niveau car elle dépend de l'application (segmentation non-sémantique adaptée au codeur ou sémantique adaptée à l'utilisateur), de la subjectivité de la notion de région ou d'objets, *etc.* De plus l'image étant souvent bruitée, les algorithmes de segmentation sont mis en défaut dans un cadre général. De par le paradigme de la segmentation, il n'existe pas de technique "universelle". C'est pourquoi il existe de

[3] *Breadth first alhgorithm* (`http://cui.unige.ch/eao/www/std/10.4.html`).

nombreux algorithmes de segmentation, nous avons présenté les principaux : ceux qui classifient les pixels, ceux qui localisent les transitions et ceux qui regroupent les régions suivant un critère d'homogénéité.

Cependant leur implantation logicielle et matérielle est complexe et coûteuse en termes de temps de traitement, consommation d'énergie et implantation microélectronique. Ces problèmes étant à ce jour toujours ouverts et les implantations matérielles dédiées aux terminaux portables inexistantes, nous proposons une avancée technologique dans ces domaines.

Nous souhaitons développer un algorithme capable d'extraire les régions d'une image sans connaissance *a priori* ainsi, son domaine d'application est moins restreint. Dans des perspectives d'intégration dans un terminal multimedia portable, nous nous orientons vers un algorithme de segmentation spatial non-sémantique, **l'algorithme de *Hill-Climbing***, utilisant un critère simple : les zones de transition d'une image gradient. Ce choix est également motivé par le caractère local de ses traitements. Nous verrons que cet algorithme se prête bien à une implantation parallèle en relâchant quelques contraintes de synchronisation.

Chapitre III

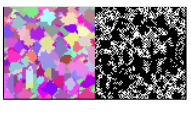
Algorithme de *Hill-Climbing* réordonnancé

Il est souvent délicat de paralléliser les algorithmes de segmentation d'images car la plupart des stratégies et méthodes de segmentation présentées au chapitre II requière des traitements globaux. Cependant, nous avons conclu ce chapitre en mettant en évidence que **l'algorithme de *Hill-Climbing*** possède **la** caractéristique indispensable pour une implantation parallèle efficace : **la localité des traitements** limitée aux proches voisins de chaque pixel.

Ainsi nous poursuivons cette étude d'adéquation algorithme-architecture sur la base de cet algorithme. Nous proposons **un réordonnancement original de cet algorithme** où seul un point de synchronisation global subsiste : le point de convergence. Ainsi les points de synchronisation globaux limitant les performances [BT89] des implantations parallèles existantes (transformation semi-complète inférieurement, recherche des minima, étiquetage...) sont supprimés. Chaque pixel possède initialement une étiquette unique et le niveau de gris du pixel associé. Son comportement étant régi par une machine à trois états, le processus de segmentation consiste à propager les données suivant les lignes de plus grande pente. Le point de convergence est atteint lorsque l'étiquette et l'altitude de chaque minimum se sont propagées sur la totalité de leur bassin d'attraction.

Nous verrons au chapitre IV que sous cette forme, l'algorithme de *Hill-Climbing* est aisément implanté sous une forme parallèle asynchrone à grains de toutes tailles. La simulation d'une telle architecture qualifiera et quantifiera ses caractéristiques.

Dans un premier temps, l'introduction et l'état de l'art (§III.1) motive le choix de paralléliser l'algorithme de *Hill-Climbing* en le comparant avec d'autres implantations parallèles connues. À partir de cette étude, nous en dérivons une solution originale où tous les traitements sont locaux à chaque pixel (§III.2). Nous démontrons (§III.3) la convergence et l'exactitude de l'algorithme, et nous estimons son comportement temporel. Un exemple de propagation sur une image simple est ensuite présenté (§III.4) et la conclusion (§III.5) clôt ce chapitre.



III.1 Introduction et motivations

Après avoir argumenté le choix d'un modèle mathématique non stochastique (§III.1.1) comme base algorithmique, nous présentons les principaux algorithmes parallèles de segmentation par construction des lignes de partage des eaux (§III.1.2), et justifions le choix de paralléliser l'algorithme de segmentation par ascension de colline.

Une présentation plus exhaustive des algorithmes parallèles peut-être consultée dans [Mog97] et [RM01].

III.1.1 Segmentation stochastique

Les algorithmes stochastiques de segmentation, basés sur les champs aléatoires de Markov MRF (*Markov Random Field*) [KB99] ou le formalisme MDL [Pat98] (*Minimum Description Length*), requièrent généralement une énorme puissance de calcul. La recherche du minimum global d'une fonction d'énergie (principe de ces algorithmes) s'effectue généralement par une méthode de relaxation itérative. Le point de convergence est atteint, suivant les données d'entrée, au bout de quelques centaines d'itérations où des opérateurs tels que logarithme, exponentiel, division, multiplication et génération de nombres aléatoires sont utilisés. Une telle complexité algorithmique est bien sûr écartée car nous devons déterminer un couple algorithme-architecture adapté aux terminaux multimedia portable.

Les réseaux de neurones ou CNN (*Cellular Neuronal/Nonlinear Network*) proposent des solutions de faible complexité[1] algorithmique et architecturale dédiées à la segmentation spatio-temporelle [LD99] [LC02] [GER00] ou spatiale [SZC+00]. Cependant ces méthodes sont généralement implantées par un circuit analogique où le contrôle de la segmentation (nombre, taille, forme des régions...) est délicat. Leur capacité d'intégration dans une chaîne de codage digitale est alors compromise.

Les approches hiérarchiques [GHP+95] sont de plus en plus utilisées dans le domaine du traitement d'image car elles permettent le développement d'algorithmes plus robustes face aux configurations pathologiques locales [SZC+00]. Cependant, l'implantation de ces méthodes implique une architecture microélectronique volumique (§IV.1.3 page 73) de complexité plus élevée par rapport aux architectures planaires. Le voisinage de chaque processeur n'est pas réduit à ses plus proches voisins, ce qui implique un réseau de communications (§IV.1.2 page 71) complexe, d'autant plus si plusieurs niveaux de hiérarchie sont utilisés.

De plus, ces systèmes ne sont pas adaptés à une application générale de segmentation car ils nécessitent une période d'apprentissage. Le contenu de l'image recherché est donc connu *a priori*, limitant ainsi le nombre d'applications possibles pour l'opérateur de segmentation développé. Nous nous sommes donc orientés vers les techniques déterministes de segmentation.

[1] Réseau analogique résistif, réseau d'opérateurs simples tels qu'addition, multiplication ou test d'égalité...

III.1.2 Segmentation déterministe

La "brique de segmentation" devant à terme s'intégrer dans une chaîne de codage vidéo orienté objet, nous préférons conserver l'aspect numérique des traitements tout en respectant les contraintes liées aux terminaux portables. Nous nous sommes donc focalisés sur les algorithmes issus de la morphologique mathématique car ceux-ci n'utilisent que des opérateurs simples du type MIN et MAX.

Comme nous l'avons vu à la section II.4, l'algorithme de LPE est l'outil de segmentation phare issu de la morphologie mathématique. L'optimisation de son implantation fait l'objet de nombreuses études. [Mog97] et [Lau98] proposent respectivement des implantations parallèles pour un partitionnement 2D-rectilinéaire et 1D-rectilinéaire utilisant une file d'attente hiérarchique (FAH). Celle-ci permet une implantation optimale des processus d'inondation [Mey91]. Cependant, sa mise en œuvre microélectronique est délicate [Lem96] car elle implique un accès aléatoire aux données. C'est pourquoi nous avons écarté les implantations parallèles proposées par [Mog97] (partitionnement 2D-rectilinéaire) et par [Lau98] (partitionnement 1D-rectilinéaire).

Un réseau de processeurs est ralenti par les points de synchronisation de l'algorithme, d'autant plus si ces points sont globaux. En d'autres termes, plus les traitements de la chaîne complète d'un algorithme sont 1) **locaux** et 2) **désynchronisés**, plus son **implantation parallèle est efficace**.

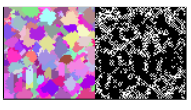
Pour répondre à la première contrainte (la localité des traitements), des architectures cellulaires sont proposées pour effectuer l'algorithme de ligne de partage des eaux par SKIZ [Nog98] (§B.6.1 page 195), par inondation uniforme [GDM98], par ruissellement ou par ascension de colline [Mog97]. Cependant, ces algorithmes mettent en œuvre un grand nombre de synchronisations globales limitant les performances du réseau de processeurs.

Pour un partitionnement à granularité intermédiaire, l'analyse des résultats de simulation [Mog97] montre une surcharge de calcul de l'algorithme *Hill-Climbing* par rapport à l'algorithme par ruissellement. Le balayage vidéo de chaque sous-image pour initialiser la queue des pixels candidats et pour propager les étiquettes issues de sous-domaines adjacents en serait la cause. Cependant, cette dernière version utilise une FAH, élément d'implantation que nous écartons car ses dimensions sont fortement dépendantes de la taille de l'image et du nombre de niveaux de gris.

[MB00] propose un algorithme de *Hill-Climbing* utilisant une FIFO simple. Bien que les traitements soient locaux, il nécessite trois étapes séparées par un point de synchronisation global : la suppression des plateaux non-minima[1], la détection des minima et la propagation de l'étiquette des minima vers tous les pixels non-minima.

Parmi toutes ces architectures et algorithmes référencés dans la littérature, aucune ne répond à la deuxième contrainte : **la désynchronisation des traitements**. Grâce à un réordonnancement de l'algorithme de *Hill-Climbing* présenté dans [MB00], nous proposons une solution capable de segmenter une image quelconque par des traitements locaux **et** désynchronisés. Nous supprimons l'ordonnancement de

[1] Calcul de l'image semi-complète inférieure (figure II.12 page 24) à l'aide d'une file d'attente.



ces trois étapes en effectuant simultanément au niveau de chaque pixel, la détection des minima et la propagation des données suivant la ligne de plus grande pente. La résolution du problème des plateaux non-minima est naturellement intégrée dans le processus de segmentation.

La réduction du nombre de points de synchronisation globaux de cet algorithme (seul le point de convergence subsiste) constitue **l'originalité de nos travaux**.

III.2 Présentation de l'algorithme réordonnancé

Cette partie présente une nouvelle forme de l'algorithme de segmentation par ascension de colline. Sa principale originalité est le réordonnancement des calculs : tous les pixels participent **simultanément** à la propagation de l'eau dans le relief topographique sans point d'attente global. Nous avons associé les travaux de D. Noguet [Nog98] à ceux de F. Meyer [Mey94] tout en s'inspirant du type d'ordonnancement des calculs présentés par F. Robin [Rob97] dans le cadre d'opérateurs morphologiques.

La section suivante III.2.1 fixe le vocabulaire utilisé, puis la section III.2.2 présente cet algorithme.

III.2.1 L'image et son relief topographique : terminologie

Cette partie est un bref rappel des éléments mathématiques décrivant le relief topographique d'une image luminance. Vous trouverez dans l'annexe B une description plus détaillée.

Définition III.1 (Graphe fortement connexe). *Un graphe orienté est dit* ***fortement connexe*** *si pour toute paire ordonnée de sommets distinct (u, v), il existe un chemin de u vers v ($u \rightsquigarrow v$) et de v vers u ($v \rightsquigarrow u$).*

Définition III.2 (Plateau). *Un plateau $P = (V, A, h)$ est un graphe fortement connexe de nœuds d'altitude constante :*

$$\forall u \text{ et } v \in V, \quad \exists \pi_1 = u \rightsquigarrow v \text{ et } \pi_2 = v \rightsquigarrow u \text{ dans } A \tag{III.1}$$

$$et \quad h(u) = h(v) \tag{III.2}$$

Définition III.3 (Frontière inférieure, supérieure). *La* ***frontière inférieure*** *∂_P^- (resp.* ***supérieure*** *∂_P^+) d'un plateau P est l'ensemble des pixels ayant au moins un voisin d'altitude strictement inférieure (resp. strictement supérieure).*

Définition III.4 (Intérieur d'un plateau). *L'intérieur d'un plateau P est le sous-graphe de P où les nœuds frontière $\partial_P = \partial_P^- \cup \partial_P^+$ sont inexistants :*

$$P_{Int} = (V_{Int}, A_{Int}) \text{ est l'intérieur de } P = (V, A) \Leftrightarrow V_{Int} = V \setminus \partial_P \tag{III.3}$$

et

$$\forall u \text{ et } v \in V_{Int}, \ (u, v) \in A_{Int} \Leftrightarrow (u, v) \in A \tag{III.4}$$

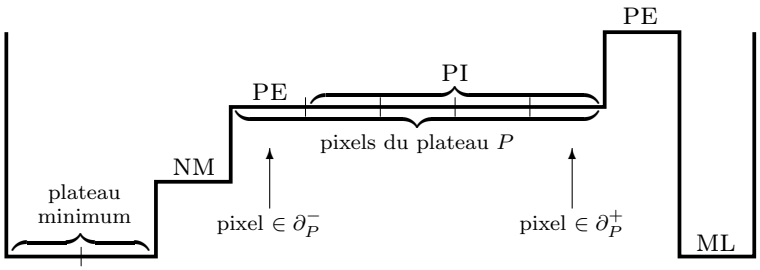


FIG. III.1 – Taxinomie des pixels d'une image.

Définition III.5 (Nature d'un pixel). *La nature est un qualificatif spécifiant la topographie locale d'un pixel.*

Soient les notations suivantes :

- ML **(minimum local) :** point ayant tous ses voisins d'altitude strictement supérieure.
- PI **(pixel intérieur) :** point ayant tous ses voisins d'altitudes supérieures ou égale dont au moins un de ses voisins est de même altitude (définition B.22 page 191).
- PE **(pixel extérieur) :** point ayant au moins un voisin d'altitude strictement inférieure et un autre de même altitude. Ce sont les seuls pixels d'un plateau à savoir qu'ils sont situés sur un plateau non-minimum.
- NM **(pixel non-minimum) :** point ayant au moins un voisin d'altitude strictement inférieure et dont aucun n'est de même altitude.

Remarque: Le terme **intérieur** est légèrement différent suivant s'il concerne un pixel (définition III.5) ou un plateau (définition III.4). Afin de simplifier les expressions, nous nommons **pixel intérieur** un pixel situé sur l'intérieur d'un plateau **ou** sur la frontière supérieure (figure III.1).

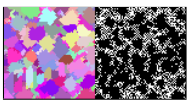
Définition III.6 (Minimum régional). *Un minimum régional $\mathcal{M}$ [Vin93] est un plateau d'altitude $h_{\mathcal{M}}$ tel que tous les pixels voisins de la frontière de $\mathcal{M}$ $\partial_{\mathcal{M}}$ à l'extérieur de ce plateau aient une valeur strictement supérieure.*

Dans notre cas, nous traitons indifféremment les minima **absolus** de l'image (points d'altitude minimale sur l'ensemble du relief) et les minima régionaux. C'est pourquoi, par abus de langage, nous appellerons "minimum", un minimum régional ou absolu.

Définition III.7 (Ligne de plus grande pente). *La ligne de plus grande pente est un chemin orienté d'un pixel vers un autre pixel d'altitude strictement supérieure tel que deux pixels consécutifs sur le chemin maximisent la pente (définition B.29 page 192).*

Définition III.8 (Pixel pivot, point de bifurcation). *Un pixel pivot ou point de bifurcation [Beu90] est un point où deux lignes de plus grande pente se rejoignent.*

Pour tout pixel p, soit $\Gamma(p)$ l'ensemble de ses voisins de plus grande pente (définition B.37 page 194).



Définition III.9 (Amont d'un pixel). *Un pixel q est localisé en amont [Mey94] d'un pixel p s'il existe un chemin π de plus grande pente joignant p à q : $\pi = (p_1 = p, p_2, \ldots, p_n = q)$ et $\forall i; p_{i-1} \in \Gamma(p_i)$.*

Définition III.10 (Inondation). *Lors du déroulement de l'algorithme de segmentation proposé, nous dirons qu'un pixel se fait **inonder** s'il reçoit de la part de son voisin un niveau de gris strictement inférieur modifiant ses variables internes.*

Ce terme est issu de l'analogie du processus de segmentation avec la simulation d'un flux d'eau dans un relief topographique.

Définition III.11 (Unification). *Un pixel **unifie** son étiquette s'il reçoit de la part d'un de ses voisins de même niveau de gris une nouvelle étiquette modifiant ses variables internes.*

III.2.2 Description de l'algorithme

Soit $G = (V, A, f)$ une image digitale (annexe A.4 page 181). Soit $\mathcal{N}(p)$, l'ensemble des pixels voisins de p dans V (G étant une image digitale, un pixel est un nœud et *vice versa*). Soit $f = (h, l, m)$ la fonction regroupant trois fonctions de V vers $\mathbb{N}$ où, pour tout $p \in V$, $h(p)$ correspond au niveau de gris de p, $l(p)$ l'étiquette et $m(p)$ le niveau de gris du minimum associé à p. Un pixel sera dit :

- **pixel minimum ou plateau** (MP) si : $\forall q \in \mathcal{N}(p)\,,\ h(q) \geqslant h(p)\,;$
- **pixel non minimum** (NM) si : $\exists q \in \mathcal{N}(p) \mid h(q) < h(p)$.

Soit les ensembles suivants :

$$\mathcal{N}^{\geqslant}(p) = \{q \in \mathcal{N}(p) \quad | \quad h(q) \geqslant h(p)\} \tag{III.5}$$

$$\mathcal{N}^{=}(p) = \{q \in \mathcal{N}(p) \quad | \quad h(q) = h(p)\} \tag{III.6}$$

$$\mathcal{N}^{I}(p) = \begin{cases} \emptyset \quad \text{si} \quad \mathcal{N}^{\geqslant}(p) = \mathcal{N}(p) \\ \{q\} \left| \begin{array}{l} q \in \mathcal{N}(p) \setminus \mathcal{N}^{\geqslant}(p) \\ h(q) = min\big(h(r), r \in \mathcal{N}(p)\big) \\ \text{choix arbitraire si solutions multiples} \end{array} \right. \end{cases} \tag{III.7}$$

Un pixel p est appelé **Minimum ou Plateau** (MP) s'il n'existe pas de voisin d'inondation ($\mathcal{N}^I(p)$ vide), sinon il est appelé **Non-Minimum** (NM). Durant la segmentation de l'image, chaque pixel se comporte suivant l'automate à trois états présenté par la figure III.2. Dans la suite de ce document, nous appelons **donnée** le couple constitué d'une étiquette et d'un niveau de gris.

Afin de caractériser les différents processus, nous dirons qu'un pixel se fait **inonder** lorsqu'il reçoit un niveau de gris strictement inférieure au sien, et qu'il **unifie** son étiquette si la nouvelle donnée reçue comporte un niveau de gris identique.

Remarques:

- Il est important de ne pas confondre le terme "inonder" qui fait référence ici à la propagation de l'étiquette et du niveau de gris du minimum d'attraction suivant la ligne de plus grande pente ou sur un plateau non-minimum, et celui d'un **algorithme par "inondation"** qui simule la propagation des

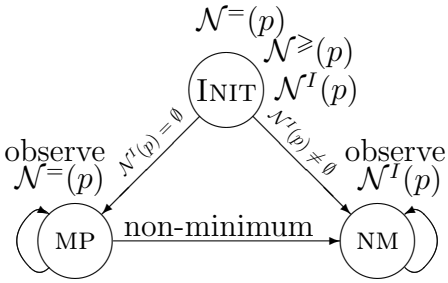


FIG. III.2 – Machine à 3 états associée à chaque pixel.

étiquettes par seuillages successifs (ou équivalent) de l'image. L'algorithme de segmentation par ascension de colline est bien un algorithme par calcul de distances topographiques et non pas un algorithme par inondation.

- Par abus de notations, nous utilisons indifféremment MP ou NM pour désigner la topographie locale d'un pixel et l'état de la machine. Le contexte permet alors de lever l'ambiguïté éventuelle.

Nous décrivons ci-dessous le comportement des pixels dans chacun des états de la machine à état fini (figure III.2).

III.2.2.1 Initialisation

Cet état de la machine est le point d'entrée de l'algorithme :
- Le pixel envoie sa donnée vers ses quatre voisins ;
- Lorsque quatre données sont reçues de $\mathcal{N}(p)$, il détermine $\mathcal{N}^{=}(p)$, $\mathcal{N}^{\geqslant}(p)$ et $\mathcal{N}^{I}(p)$. Pour assurer la cardinalité maximale de un pour $\mathcal{N}^{I}(p)$, la machine choisi aléatoirement un **unique** voisin d'inondation si plusieurs voisins de plus grande pentes existent ;
- Si $\mathcal{N}^{I}(p)$ est vide, (un voisin d'altitude strictement inférieur n'est pas trouvé) la machine passe à l'état **Minimum ou Plateau** MP (figure III.3) sinon $\mathcal{N}^{I}(p)$ est non vide, la ligne de plus grande pente peut-être déterminée localement : la machine passe à l'état **Non-Minimum** NM (figure III.4).

La phase d'initialisation est terminée pour le pixel p, la propagation des données peut avoir lieu en ce point. Nous appelons phase (ou processus) de relaxation, ou plus simplement **relaxation**, les deux processus d'unification (p est MP, §III.2.2.2) et d'inondation (p est NM, §III.2.2.3).

III.2.2.2 Minimum ou Plateau (MP)

Dans cet état, le pixel p :
- écoute $\mathcal{N}^{=}(p)$;
- met à jour son étiquette s'il reçoit $l(q) < l(p)$ (processus d'unification) ;
- s'il reçoit un niveau de gris $m(q) < m(p)$ (processus d'inondation), cela signifie qu'il est situé sur un plateau non-minimum (figure III.3c) ;
  - fixe $\mathcal{N}^{I}(p)$ en conséquence,

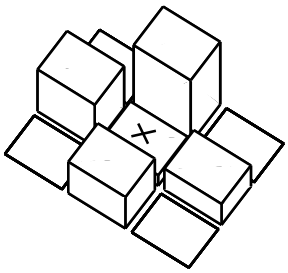
– passe à l'état NM.

Si le pixel est un minimum local (figure III.3a), ce dernier n'effectuera aucun calcul jusqu'au point de convergence puisque $\mathcal{N}^{=}(p)$ est un ensemble vide. Le processus d'unification permet de propager l'étiquette minimale d'un plateau minimum sur sa totalité (figure III.3b). Ainsi au point de convergence, chaque plateau minimum est référencé par une étiquette unique, et seuls les pixels des plateaux minima sont à l'état MP.

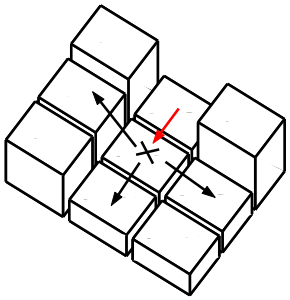
III.2.2.3 Non-Minimum (NM)

Dans cet état, le pixel p :
- n'écoute que le voisin $\mathcal{N}^{I}(p)$;
- copie la nouvelle donnée dès sa réception (processus d'inondation multiple, remise en question de décisions antérieures).

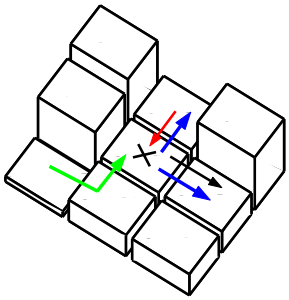
Dans les deux cas (MP ou NM) si les variables internes du pixel p sont modifiées, ce dernier met ses données ($m(p)$ et $l(p)$) à disposition de tous ses voisins $\mathcal{N}(p)$ puisque p ne sait pas quels pixels voisins l'écoutent.



(a) Minimum local



(b) Plateau minimum



(c) Plateau non-minimum

FIG. III.3 – Différentes topographies rencontrées pour le même état Minimum ou Plateau (MP) à l'issue de l'état initial.

III.2.3 Conclusion

L'algorithme de *Hill-Climbing* réordonnancé présenté détecte les minima, étiquette les plateaux minima, et propage les données sur le relief topographique **simultanément et indépendamment** au niveau de chaque pixel. Chaque machine

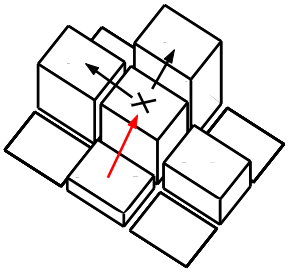


FIG. III.4 – Exemple de topographie pour l'état Non-Minimum (NM).

à état fini introduite au niveau pixel est indépendante. Seules les communications locales permettent la mise à jour de ses variables internes et les transitions entre états, et seul le point de convergence constitue un point de synchronisation global.

III.3 Présentation formelle et comportement de l'algorithme réordonnancé

La section précédente présentait l'algorithme de *Hill-Climbing* réordonnancé. Nous démontrons dans ce chapitre que le comportement de tous les pixels permet bien de segmenter l'image.

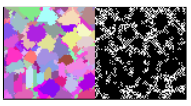
Nous modélisons le flux des données dans le réseau sous forme de passages de messages dans un graphe, et montrons que l'algorithme proposé réalise bien une segmentation de l'image suivant la distance topographique (définitions B.33 et B.42 page 193) et que les segments obtenus localisent avec précision les zones de transition de l'image gradient.

L'analyse de l'algorithme réordonnancé s'appuie, entre autres, sur deux concepts : la modélisation mathématique des calculs cellulaires (§III.3.1) et la caractérisation mathématique d'un relief topographique (§III.2.1 page 34). Ce modèle mathématique permet d'utiliser l'asynchronisme des traitements locaux dès la conception de l'algorithme [Duc99, chap.4]. De plus, la faisabilité d'une architecture microélectronique capable d'implanter des opérateurs associatifs élémentaires est montrée [Dul96]. Nous avons donc associé ces deux axes de recherche (algorithme réordonnancé et réseau associatif) car l'expression des calculs cellulaires du modèle de réseau associatif et celle de l'algorithme de segmentation réordonnancé sont similaires.

Afin que cette présentation formelle soit aussi claire que possible, nous présentons les éléments et concepts de base du modèle de réseau associatif. De plus, un exemple de construction du graphe pour une image de faible résolution est présenté à la section III.4 page 62.

III.3.1 Préliminaires : le modèle de réseau associatif

Nous utilisons dans la suite de ce chapitre un modèle mathématique de calcul cellulaire, mené originellement par A. Mérigot [Mér92], appelé **réseau associatif**. À partir de [Duc99] et [Mér97], nous présentons quelques éléments de base utiles



pour la validation de notre algorithme. Le lecteur désirant approfondir ce domaine peut consulter [Dul96] pour les aspects implantation matérielle du modèle, [Duc99] pour l'extension du modèle et la démonstration du comportement des opérateurs et [GDM98], [DM01] pour des exemples d'application dans le domaine du traitement d'images.

Plusieurs termes relatifs à la théorie des graphes sont utilisés, leur définition est rassemblée dans l'annexe A page 177.

La structure architecturale. Considérons un réseau $G = (V, A, f)$ où V est l'ensemble des nœuds (éléments de traitement) et A l'ensemble des arcs (lien de communication unidirectionnel). Nous supposons que G est une grille symétrique de degré constant : pour tout nœud v de V, $d_G^-(v) = d_G^- = C^{ste}$.

Soit $\mathbb{S}$ l'ensemble de définition de toutes les données véhiculées dans le réseau et calculées par les nœuds du réseau. Une **variable parallèle** est un $|V|$-uplet réparti à raison d'une donnée par nœud sur le réseau. Si f est une variable parallèle de $\mathbb{S}^{|V|}$, alors $f[v]$ est la valeur de f sur le nœud v :

$$\forall f \in \mathbb{S}^{|V|} \text{ et } \forall v \in V, \quad f[v] \in \mathbb{S} \tag{III.8}$$

Le réseau partiel $G' = (V, A', f)$ est obtenu en masquant les arcs entrants de certains nœuds du graphe G. Ceci permet de considérer divers schémas d'interconnexions dans le réseau. Un graphe partiel est manipulé par une variable parallèle net_G de mots binaires de d_G^- bits : au niveau de chaque nœud v, le $i^{\text{ème}}$ bit de la donnée $net_G[v]$, noté $net_G[v][i]$, permet de savoir si le $i^{\text{ème}}$ arc entrant de v, dans le sens horaire par exemple, est masqué ou non («0» masqué, «1» non masqué). Nous appelons **masque** la valeur locale $net_G[v]$ d'un nœud v.

La connexité est une notion fondamentale pour les réseaux associatifs car elle définit l'ensemble des voisins directs du nœud courant. Nous notons $\Gamma_{G'}(v)$ les ascendants de v dans G' :

$$\Gamma_{G'}(v) = \{u \in V \mid (u, v) \in A'\} \tag{III.9}$$

Remarque: Pour simplifier les notations, le même symbole Γ désigne les ascendants dans le cas du modèle de réseau associatif et les voisins de plus grande pente (définition B.37 page 194) lors de la description du relief topographique (équation III.22 page 45). Nous faisons ce choix car seuls des calculs sur les données des voisins de plus grande pente sont effectués.

Les ascendants de v et lui-même est noté $\overline{\Gamma_G}(v) = \Gamma_G(v) \cup \{v\}$. Cette définition peut-être étendue aux ancêtres de v (définition A.9 page 178) : $\Gamma_G^{\blacktriangle}(v)$. Nous notons $\overline{\Gamma_G^{\blacktriangle}}(v) = \Gamma_G^{\blacktriangle}(v) \cup \{v\}$.

Les opérations élémentaires. Soit $\diamond$ un opérateur n-aire sur $\mathbb{S}^{|V|}$.

Définition III.12 (Association). *Une **association** est un calcul **global** qui modifie la donnée $f[v]$ d'une variable parallèle f sur chaque nœud v en combinant, avec un opérateur **associatif et commutatif** $\diamond$, toutes les composantes de la variable*

parallèle situées sur $\overline{\Gamma^{\blacktriangle}_{G'}}(v)$ *: les ancêtres de* v *et lui-même dans un réseau partiel donné* G'*.*

Plus formellement, on note ce calcul global par $\diamond$-`association`(net_G, f) et on le définit par :

$$\diamond\text{-}\texttt{association}(net_G, f) \; : \; \begin{array}{ccc} \mathbb{S}^{|V|} & \rightarrow & \mathbb{S}^{|V|} \\ f & \mapsto & g \end{array} \tag{III.10}$$

avec

$$g[v] = \underset{u \in \overline{\Gamma^{\blacktriangle}_{G'}}(v)}{\diamond} f[u] \tag{III.11}$$

En d'autres termes, $\diamond$-`association`(net_G, f) retourne une variable parallèle g telle que la valeur $g[v]$ au nœud v est égale à l'application itérée de l'opérateur $\diamond$ sur f pour tous les nœuds appartenant à l'ensemble connexe des ancêtres de v. Les opérateurs élémentaires considérés sont les fonctions logiques (OR, AND, XOR), maximum, minimum et addition.

En utilisant les ascendants (pères) de v et lui-même $\overline{\Gamma_{G'}}(v)$ de v au lieu des ancêtres $\overline{\Gamma^{\blacktriangle}_{G'}}(v)$, on définit la variante de l'association appelée `step-association` :

$$\diamond\text{-}\texttt{step-association}(net_G, f) \; : \; \begin{array}{ccc} \mathbb{S}^{|V|} & \rightarrow & \mathbb{S}^{|V|} \\ f & \mapsto & g \end{array} \tag{III.12}$$

avec

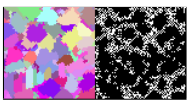
$$g[v] = \underset{u \in \overline{\Gamma_{G'}}(v)}{\diamond} f[u] \tag{III.13}$$

En d'autres termes, une `step-association` en v consiste à obtenir les valeurs de ses pères et la sienne, puis à effectuer un seul calcul local sur l'ensemble de ces valeurs.

Enfin, nous appelons $\diamond$-`direct-association`(net_G, f), l'application réitérée jusqu'à idempotence[1] (stabilisation) d'une $\diamond$-`step-association`(net_G, f). Cette dernière est intéressante du point de vue implantation matérielle, car elle fournit un résultat global qui ne fait intervenir que des calculs locaux limités au nœud lui-même et à ses plus proches voisins.

[Mér92] montre que si les données sont ordonnées et si l'ordre des calculs intermédiaires ne peut l'être, l'opérateur binaire $\diamond$ utilisé doit être associatif. Par ailleurs, si les données ne sont pas ordonnées, l'opérateur $\diamond$ doit alors être commutatif car on ne peut assurer que les calculs seront réalisés en prenant les données dans un ordre précis. C'est pourquoi, les propriétés d'associativité et de commutativité permettent de s'abstraire des contraintes d'ordonnancement et de répartition des données dans le réseau. Les réseaux associatifs, en tant que modèle de programmation, n'impliquent pas une implantation particulière des calculs.

[1] Si elle existe.



Stabilisation du réseau : la fin des calculs. Étudions la stabilisation du réseau par la propriété d'**idempotence** de $\diamond$.

Définition III.13 (k-idempotence). *L'opérateur $\diamond$ défini sur l'ensemble $\mathbb{S}$ est k-idempotent s'il est idempotent à partir du rang k dans les expressions : pour tout $x \in \mathbb{S}$, on a :*

$$\underbrace{x \diamond \cdots \diamond x}_{k+1} = \underbrace{x \diamond \cdots \diamond x}_{k} \tag{III.14}$$

La forme d'idempotence la plus courante pour les opérateurs correspond à la 1-idempotence : $x \diamond x = x$.

Définition III.14 ($\oplus$, un s-opérateur). *L'opérateur $\oplus$, défini sur l'ensemble de données $\mathbb{S}$, est un s-opérateur*[1] *s'il est associatif, commutatif et idempotent, et possède un élément neutre $e_\oplus$ dans $\mathbb{S}$.*

Par exemple, l'opérateur $\oplus =$`min` pour $\mathbb{S} = [0, N]$ est un `s`-opérateur car, pour tout x, y et z dans $[0, N]$:

- $\min(\min(x, y), z) = \min(x, \min(y, z))$ — associativité —
- $\min(x, y) = \min(y, x)$ — commutativité —
- $\min(x, x) = x$ — 1-idempotence —
- $\min(x, N) = N$ — élément neutre —

Les propositions III.1 et III.2 ci-dessous démontrées dans [Duc99] sont importantes car elles assurent la convergence du réseau et l'existance d'une valeur de la variable parallèle à la fin des calculs. L'ordre partiel défini par un `s`-opérateur permet de montrer la terminaison de la `min-direct-association` (proposition III.2).

Proposition III.1 ($\oplus$, un ordre partiel). *Si $\oplus$ est un s-opérateur sur $\mathbb{S}$, alors la relation binaire $\preccurlyeq_\oplus$ définie par $(x \preccurlyeq_\oplus y) \equiv (x \oplus y = x)$ est une relation d'ordre partiel*[2] *sur $\mathbb{S}$. De plus, le plus petit élément d'un ensemble $\{x_1, x_2, \ldots, x_n\}$ (au sens de $\preccurlyeq_\oplus$), noté $\bot_\oplus \{x_1, x_2, \ldots, x_n\}$, est obtenu par $x_1 \oplus x_2 \oplus \cdots \oplus x_n$.*

Rappel : Le graphe partiel G' de G est obtenu en masquant certains arcs entrants suivant la variable parallèle net_G.

Proposition III.2 (Fin déterministe). *Si $\oplus$ est un s-opérateur sur $\mathbb{S}$, alors la $\oplus$-`direct-association`(net_G, f) se termine sur un résultat déterministe sur tout réseau G', initialisé avec toute variable parallèle f de $\mathbb{S}^{|V|}$. De plus, le résultat sur le nœud v du réseau est $\bot_\oplus \{f[u], u \in \overline{\Gamma^{\blacktriangle}_{G'}}(v)\}$.*

En particulier, la convergence des calculs par un `s`-opérateur est assurée même si le graphe G' est cyclique.

[1] Un tel opérateur est parfois appelé infimum. Le terme `s`-opérateur vient de sort car il peut-être utilisé pour ordonner les éléments de $\mathbb{S}$ [Duc99] (*cf.* proposition III.1).

[2] Relation binaire réflexive ($\forall x \in \mathbb{S}, x \preccurlyeq_\oplus x$), antisymétrique ($\forall x\ y \in \mathbb{S}, x \preccurlyeq_\oplus y$ et $y \preccurlyeq_\oplus x \Rightarrow x = y$) et transitive ($\forall x\ y\ z \in \mathbb{S}, x \preccurlyeq_\oplus y$ et $y \preccurlyeq_\oplus z \Rightarrow x \preccurlyeq_\oplus z$), où il est possible que ni $x \preccurlyeq_\oplus y$, ni $y \preccurlyeq_\oplus x$ soit vérifié.

Proposition III.3 (Contrainte du réseau pour $\diamond$ non idempotent). *Si le sous-réseau n'a pas de cycles*[1]*, la terminaison du calcul global défini comme l'exécution concurrente des calculs locaux* $\diamond$`-direct-association`*, prend fin au bout d'un temps fini quel que soit l'opérateur* $\diamond$*.*

Par exemple pour une variable parallèle f, le calcul de la somme des $f[v]$ sur les nœuds v de V requiert une structure acyclique du graphe partiel G' car l'opérateur $+$ n'est pas idempotent.

Les principales notions utiles à la validation de l'algorithme désormais présentées, étudions le flux des données dans le réseau. Nous proposons une reformulation de l'algorithme de segmentation présenté à la section III.2.2 page 36. Le processus est décrit par une propagation des données dans un graphe où le degré de connectivité de chaque nœud est dynamique (variable). À l'aide des concepts présentés à l'instant, nous validons l'algorithme de segmentation par ascension de colline réordonnancé en montrant la convergence de la construction du graphe partiel G' (*i.e.* net_G) et la convergence des calculs.

Cette validation comporte trois partie : la première (§III.3.2) commune initialise la machine à trois états, la seconde (§III.3.3) décrit le comportement de l'algorithme sans faire d'hypothèses sur les temps de propagation puis, les instants de réception des étiquettes décrivant les régions sont formalisés (§III.3.4).

III.3.2 Variables, opérateurs et état initial du réseau

Soit $G = (V, A, f, \Omega)$ l'image digitale pondérée de dimensions $C \times L$, *i.e.* un graphe orienté symétrique où $f : V \to \mathbb{S}$ est une variable parallèle du réseau G (fonction de valuation) et $\Omega : A \to \mathbb{R} \times \overline{\mathbb{R}}^+$ la fonction de pondération du graphe.

Soit $\mathbb{S} = [0, N] \times [0, C * L]$ où le premier élément de chaque couple correspond à la luminance du pixel (majorée par N, la luminance maximale) et le deuxième son étiquette (étiquette maximale égale au nombre de pixels de l'image de dimensions $C \times L$).

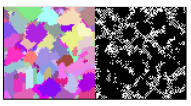
Remarque: Comme l'image gradient représente un relief topographique, l'altitude d'un pixel fait toujours référence à sa luminance (niveau de gris) et inversement.

Les notations suivantes sont adoptées :
- $G = (V, A, f, \Omega)$: graphe initial (image digitale) ;
- $G' = (V, A', f, \Omega)$: graphe partiel en cour de construction ;
- $G^\infty = (V, A^\infty, f, \Omega)$: graphe partiel au point de convergence ;
- $G^* = (V^*, A^*, f, \Omega)$: sous-graphe de G.

Soit les fonctions $h : V \to [0, N]$ l'altitude de chaque pixel et $l : V \to [0, C * L]$ son étiquette.

Pour tout arc $a = (u, v)$ de A, $\Omega(a) = (\text{cost}(a), t^x_{prop}(a))$ représente le coût de passage (définition B.31 page 193) du nœud u vers v, et les temps de propagation t_{prop}

[1] Il n'existe pas de chemin tel que ses deux extrémités soit un même nœud (définition A.19 page 179).



de la donnée $x \in \mathbb{S}$ entre deux nœuds voisins (uniquement utilisés lors de l'étude du comportement temporel du réseau, section III.3.4 page 57).

La variable parallèle net_G fixant la connectivité des nœuds évolue au cours du processus de segmentation.

Initialement, chaque nœud du graphe possède sa propre altitude et une étiquette unique. Les étiquettes peuvent être, par exemple, la position du pixel suivant un balayage direct vidéo, ou bien aléatoires (§IV.4.1 page 95). De plus, chaque nœud communique avec tous ses voisins :

$$\forall v \in V \text{ et } \forall i \in [0, d_G^- - 1], \quad net_G[v][i] = 1 \tag{III.15}$$

avec d_G^- le degré de connexité du graphe G.

Nous définissons l'opérateur associatif d'unification $\circledcirc$ sur le sous-graphe G^* consistant à calculer le `min` sur les étiquettes :[1]

$$\forall u \text{ et } v \in \mathbb{S}, \quad u \circledcirc v = \big(u_1, \min(u_2, v_2)\big) \tag{III.16}$$

$\circledcirc$ est un `s`-opérateur car les opérateurs `min` et identité sont des `s`-opérateur, et :

$$\circledcirc\texttt{-direct-association}(net_{G^*}, f) = \begin{cases} \texttt{Id-direct-association}(net_{G^*}, h) \\ \texttt{min-direct-association}(net_{G^*}, l) \end{cases} \tag{III.17}$$

où `Id` est l'opérateur identité. On rappelle que la variable parallèle f correspond au couple (h,l) des altitudes et étiquettes.

Nous définissons sur un sous-graphe G^* l'opérateur d'inondation $\lhd$ consistant à copier les données de son[2] voisin d'inondation :

$$\forall u \text{ et } v \in \mathbb{S},\ u \lhd v = v \tag{III.18}$$

Cet opérateur est défini par :

$$\begin{array}{rccc} \lhd\texttt{-step}(net_{G^*}, f) \ : & \mathbb{S}^{|V|} & \rightarrow & \mathbb{S}^{|V|} \\ & f & \mapsto & g \end{array} \tag{III.19}$$

avec

$$g[v] = \underset{u \in \mathcal{N}^1(v)}{\lhd} f[u] \tag{III.20}$$

L'opérateur $\lhd$ est associatif mais n'est pas commutatif, donc il n'appartient pas à l'ensemble des opérateurs associatifs (définition III.12 page 40). C'est pourquoi, le terme `association` est omis.

Par abus de notation, $u \lhd v$ indique également le processus d'inondation du nœud u par le nœud v ($\forall u, v \in V^*$).

[1] $\forall u \in \mathbb{S}$, on note $u = (u_1, u_2)$.

[2] Unicité présentée à la section III.3.3.1.

Proposition III.4 (Convergence de $\lhd$). *L'opérateur $\lhd$ converge sur toute forêt d'arborescences.*

Preuve: Montrons que la proposition III.3 (convergence de tout opérateur associatif $\diamond$ sur un réseau acyclique) est également vérifiée pour $\lhd$ sur une forêt d'arborescences.

Soit G^* une forêt d'arborescences de racines $\{u_0^k\}$. Pour tout k, comme la racine u_0^k n'a pas de père, elle est stable juste après l'émission de ses données vers ses fils. Tout nœud $u \notin \{u_0^k\}$ n'a qu'un seul père, donc l'opération de copie $\lhd$ est déterministe et se termine localement au bout d'un temps fini (hypothèse de temps de communication fini entre deux nœuds voisins). Par récurrence sur la plus grande branche parmi toutes les arborescences, on montre que tous les nœuds finissent par avoir un calcul stable au bout d'un temps fini. Le calcul est donc globalement stable après un délai fini. CQFD

III.3.3 Comportement atemporel

Dans toute cette partie, nous faisons volontairement aucune hypothèse sur les temps de propagation des données. La fonction de pondération Ω est uniquement le coût de déplacement topographique «cost» (définition B.31 page 193) :

$$\forall (u,v) \in A, \Omega(u,v) = \text{cost}(u,v) \tag{III.21}$$

Ceci permet de lever toute contrainte de synchronisation entre les nœuds du réseau. Nous supposons seulement, afin qu'une propagation soit possible, que tous les nœuds voisins peuvent communiquer, c'est-à-dire que les temps de communication sont finis. Nous verrons alors que la segmentation a lieu **sans aucune connaissance de l'ordonnancement** des calculs effectués par les nœuds du réseau.

III.3.3.1 Phase d'initialisation d'un nœud

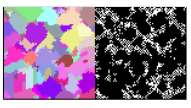
Cette phase constitue le point de départ de l'algorithme où un premier échange de données est effectué entre pixels voisins. L'ensemble des calculs présentés s'effectue de façon localement synchronisé au niveau de chaque nœud.

Soient v un nœud quelconque de V et $\Gamma(v)$ ses voisins de plus grand pente[1] :

$$\forall v \in V, \quad \Gamma(v) = \{u \in \mathcal{N}(v) \mid pente(v,u) = \max_{w \in \mathcal{N}(v)} pente(v,w)\} \tag{III.22}$$

Mise à jour de f durant la phase d'initialisation. Durant cette phase de l'algorithme propre à chaque pixel v, une première mise à jour des variables internes ($f[v]$) est réalisée en fonction de la topographie locale de v (§III.2.2 page 36). Afin de simplifier les notations, on note $f[v] = \diamond_u f[u]$ la mise à jour de la donnée $f[v]$ par un opérateur $\diamond$ sur les données d'entrée $f[u]$. Soit $\overline{\mathcal{N}}(v) = \mathcal{N}(v) \cup \{v\}$.

[1] Voisins maximisant la pente topographique (définition B.37 page 194).



Si $\Gamma(v)$ est
- vide : $f[v] = \underset{u \in \overline{\mathcal{N}^{=}}(v)}{\circledcirc} f[u]$: v est un pixel intérieur.
- un singleton : $f[v] = f[v] \lhd f[\Gamma(v)]$.
- multiple : $f[v] = f[v] \lhd f\left[\mathcal{N}^I(v)\right]$: un **unique voisin** de plus grande pente est choisi arbitrairement parmi $\Gamma(v)$ et fixe $f[v]$.

Comme $net_G[v][i]$ vaut 1 pour tout v et pour tout i, la mise à jour de $f[v]$ s'effectue sur **toutes** les données issues des voisins de v.

Mise à jour de net_G durant la phase d'initialisation. Au cours du processus d'initialisation, le voisinage de chaque nœud est modifié en fonction de sa topographie locale.

Si $\Gamma(v)$ est
- vide : Seuls les bits du masque $net_G[v]$ associés aux arcs dont l'origine est de même altitude sont fixés à un (non masqué). Les autres, associés aux arcs dont l'origine est d'altitude strictement supérieure, sont fixés à zéro (masqué).
- un singleton : Seul l'arc entrant issu de $\Gamma(v)$ est non masqué.
- multiple : v est un **pixel pivot** car il est le lieu de rencontre de plusieurs lignes de plus grande pente. Seul l'arc entrant du voisin choisi arbitrairement $\mathcal{N}^I(v)$ est non masqué.

Comportement du graphe partiel $G' = (V, A', f, \Omega)$. L'évolution de la variable parallèle net_G crée, à chaque mise à jour du masque d'un nœud v, un nouveau graphe partiel G' de G où certains des arcs entrants de v sont supprimés.

Proposition III.5. *Dans le graphe G', les données de f ne peuvent se propager de l'amont vers l'aval du relief topographique.*

Preuve: Montrons que pour tout nœud v de V, l'origine de tout arc entrant de v est un nœud d'altitude inférieure ou égale.

Si $\Gamma(v) \neq \emptyset$, alors le choix de $\mathcal{N}^I(v)$ implique que le degré $d^-_{G'}(v)$ est égal à un. Une fois initialisé, le pixel passe à l'état NM et y reste, et le masque $net_G[v]$ n'est plus jamais modifié et autorise uniquement la lecture des données issues de ce voisin d'altitude strictement inférieure.

Si $\Gamma(v) = \emptyset$, alors il n'existe pas de voisin de plus grande pente, v n'écoute que ses voisins $\mathcal{N}^{=}(v)$ (équation III.6 page 36). Pour tout nœud $u \in V$ voisin de v dans G tel que $h(u) > h(v)$, l'arc arc $a = (u, v)$ n'appartient pas à A'. Donc toute donnée issue d'un voisin de v d'altitude strictement supérieure ne peut-être lue. CQFD

Remarque: Pour tout pixel v minimum local, $\Gamma(v)$ et $\mathcal{N}^{=}(v)$ sont vides, donc $net_G[v]$ est nul : l'état de v est invariant puisqu'il ne peut recevoir de données.

III.3.3.2 Phase de relaxation d'un nœud

À présent, montrons que l'algorithme converge et qu'à l'équilibre, chaque minimum correspond à une région connexe de l'image. Dans un premier temps, une hiérarchie d'hypothèses classifie l'ensemble des topographies possibles d'une image. Ensuite pour chaque classe, la convergence et la justesse des traitements sont démontrées.

Soit α le nombre de minima contenus dans l'image. Soit ${G^*_{\mathcal{M}}}^i = ({V^*_{\mathcal{M}}}^i, {A^*_{\mathcal{M}}}^i, f, \Omega)$ le sous-graphe de G associé au $i^{\text{ème}}$ plateau minimum d'altitude ${h_{\mathcal{M}}}^i$. Soit $G^*_{\mathcal{M}} = (V^*_{\mathcal{M}}, A^*_{\mathcal{M}}, f, \Omega) = \bigcup_{i\in[1,\alpha]} {G^*_{\mathcal{M}}}^i$ les plateaux minima de G (ici, un minimum local est également considéré comme un plateau). $G^*_{\mathcal{M}}$ est non vide car il existe au moins un pixel d'altitude minimale dans l'image.

Hypothèse I : L'image n'est composée que de minima locaux.

$$\forall v \in V^*_{\mathcal{M}}, |\mathcal{N}^{=}(v)| = 0 \tag{III.23}$$

Sous cette hypothèse, chaque minimum de l'image ainsi que l'étiquette de chaque région sont parfaitement déterminés. En effet, tout nœud minimum local est invariant ; il ne change ni d'altitude, ni d'étiquette. Nous montrons (propositions III.15 et III.17) qu'au point de convergence, G^∞ est une forêt d'arborescences recouvrante (définitions A.28 et A.32 page 180) où l'étiquette de chaque racine de $V^*_{\mathcal{M}}$ s'est propagée sur la totalité de l'arborescence.

Hypothèse I.1 : L'image est semi-complète inférieurement.

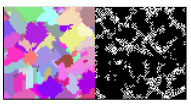
$$\forall v \in V \setminus V^*_{\mathcal{M}}, |\Gamma(v)| > 0 \tag{III.24}$$

Sous cette hypothèse, tout pixel non-minimum possède au moins un voisin d'altitude strictement inférieure. Nous montrons (proposition III.7) que la topologie du graphe partiel G' est entièrement figée dès que tous les nœuds ont fini leur phase d'initialisation (net_G a convergé).

Hypothèse I.1.a : L'image est exempte de pixels pivots.

$$\forall v \in V \setminus V^*_{\mathcal{M}}, |\Gamma(v)| = 1 \tag{III.25}$$

Sous cette hypothèse, le voisin de plus grande pente est déterminé de façon unique.



Hypothèse I.1.b : L'image comporte des pixels pivots.

$$\exists v \in V \setminus V^*_{\mathcal{M}} \quad | \quad |\Gamma(v)| > 1 \tag{III.26}$$

Sous cette hypothèse, chaque pixel pivot v a déterminé durant sa phase d'initialisation un unique voisin d'inondation $\mathcal{N}^I(v)$ parmi ses voisins de plus grande pente $\Gamma(v)$. Nous montrons que la topologie du graphe partiel est entièrement déterminée, et que le comportement de la propagation est identique à celle décrite sous l'hypothèse I.1.a. Le comportement des pixels pivots entraîne uniquement un indéterminisme sur la position des LPE.

Hypothèse I.2 : L'image n'est pas semi-complète inférieurement.

$$\exists v \in V \setminus V^*_{\mathcal{M}} \quad | \quad |\Gamma(v)| = 0 \tag{III.27}$$

Sous cette hypothèse, les pixels intérieurs[1] d'un plateau non-minimum ne peuvent déterminer leur voisin d'inondation durant leur phase d'initialisation. Nous montrons (proposition III.18) que si deux minima sont séparés par un plateau non-minimum, la LPE est située sur ce plateau. Nous montrons également que tous les nœuds de ce plateau sont inondés et que l'évolution de la variable parallèle net_G converge (proposition III.16), et enfin que la phase de relaxation transforme un plateau non-minimum en une forêt d'arborescences recouvrante (proposition III.17).

Hypothèse II : L'image comporte des plateaux minima.

$$\exists v \in V^*_{\mathcal{M}}, |\mathcal{N}^{=}(v)| > 0 \tag{III.28}$$

Nous montrons que la phase d'initialisation isole les plateaux minima (proposition III.20) et que la phase de relaxation conservent la propriété fortement connexe des sous-graphes formés par les nœuds d'un plateau minimum et leurs arcs : un plateau minimum ne peut-être inondé. Nous montrons également que l'étiquette minimale d'un plateau se propage sur sa totalité et converge (proposition III.21). La substitution du plateau par un unique nœud de connectivité adéquate nous permet de nous ramener à une topologie du réseau où l'hypothèse I s'applique.

III.3.3.3 Propagation déterministe des étiquettes suivant les lignes de plus grande pente

Supposons l'hypothèse I.1.a vérifiée : tout minimum est local, et $\Gamma(v)$ est un singleton pour tout nœud v non minimum (équation III.25). Montrons que A' est fixé ($A' = A^\infty$) dès la phase d'initialisation globalement terminée, que le mouvement ascensionnel des données le long des reliefs pentus est déterministe, converge, et que G^∞ est une forêt d'arborescences de moindre coût recouvrante.

Proposition III.6. *La phase d'initialisation construit $G' = (V, A', f, \Omega)$ tel que :*

$$\forall u, v \in V, \quad (u, v) \in A' \Leftrightarrow \Gamma(v) = \{u\} \tag{III.29}$$

[1] Pixel d'un plateau où tous ses voisins sont d'altitudes supérieures ou égales (définition B.22 page 191).

Preuve: Montrons la double implication III.29.

Par hypothèse, tout nœud v non minimum a un unique voisin u de plus grande pente $\Gamma(v)$. Comme l'initialisation (§III.3.3.1 page 45) ne masque pas l'arc issu du voisin le plus bas, $(u, v) \in A'$. Si u est le $i^{\text{ème}}$ voisin de v, alors $net_G[v][i] = 1$.

Comme l'initialisation (§III.3.3.1) masque tout arc entrant dont l'origine n'est pas $\Gamma(v)$:

$$\forall w \in \mathcal{N}(v) \setminus \Gamma(v), \quad (w, v) \notin A' \tag{III.30}$$

Si $w \in \mathcal{N}(v) \setminus \Gamma(v)$ est le $i^{\text{ème}}$ voisin de v, alors $net_G[v][i] = 0$. CQFD

Proposition III.7. *L'initialisation terminée, la variable parallèle net_G est invariante.*

Preuve: Montrons que la phase de relaxation ne peut modifier net_G.

Pour tout nœud minimum $v \in V$ et pour tout $i \in [0, d_G^-]$, $net_G[v][i] = 0$: le nœud est invariant (aucune mise à jour n'est possible).

Pour tout nœud non-minimum v, l'initialisation fixe un unique voisin : la machine à état transite vers l'état NM. Comme dans cet état, la machine associée à v copie uniquement les données issues de l'origine de l'unique arc entrant de v, $net_G[v]$ n'est jamais modifié. CQFD

La construction de G' est déterministe car le voisin d'inondation est déterminé de façon unique pour tout nœud non-minimum, et pour tout minimum v, tous les arcs entrants de v sont supprimés.

Proposition III.8. *Le graphe partiel G^∞ suivant la variable parallèle net_G au point de convergence est une forêt d'arborescences recouvrante de racines, tous les minima de G^∞.*

Preuve: Montrons que chaque minimum local est la racine d'une arborescence.

Pour tout minimum local u du graphe partiel G^∞, le degré de connectivité est nul : u n'a pas de père. Tout nœud v non-minimum ne possède qu'un seul voisin d'inondation : v possède un unique père. Par ailleurs, si u est père de v, alors $h(u) < h(v)$. Donc tout chemin dans G^∞ est strictement ascendant (*i.e.* strictement croissant selon h). Il ne peut donc pas exister de cycles, G^∞ est donc une forêt d'arborescences.

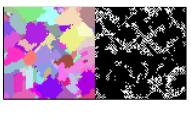
La phase d'initialisation ne supprimant aucun nœud, G^∞ recouvre G. CQFD

Proposition III.9. *Pour toute racine u d'une arborescence, l'étiquette $l(u)$ se propage vers tous les nœuds v de l'arborescence.*

Preuve: Montrons que l'algorithme de relaxation converge vers $l(u)$.

Itérons successivement une $\triangleleft-$`step` en tout nœud de V (tout nœud minimum étant isolé, $\triangleleft$ n'a aucune influence sur lui). La proposition III.4 assure la convergence du calcul global.

Comme le pixel initiateur de tout chemin d'une arborescence est un minimum local, l'étiquette de ce dernier se propage ($\triangleleft$ est un opérateur de copie) sur la totalité de cette arborescence. CQFD



Proposition III.10. *La forêt $G^\infty = (V, A^\infty, f, \Omega)$ est de coût minimum.*

Preuve: Montrons que toute branche de l'arborescence est une ligne de plus grande pente (définition B.38 page 194), alors suivant [Mey94], chacune de ces branches minimise la π-distance topographique (définition B.32 page 193).

Tout nœud u non-minimum, appartenant à l'arborescence de racine v, est connecté à son unique voisin $\Gamma(u)$ de plus grande pente : le chemin $\pi = v \rightsquigarrow u$ est une ligne de plus grande pente. Il est démontré dans [Mey94] que la π-distance topographique est minimale si π est une ligne de plus grande pente. Donc chaque branche d'une arborescence quelconque de G' minimise la fonction de coût Ω. CQFD

Conclusion. Sous l'hypothèse I.1.a, l'algorithme réordonnancé segmente l'image suivant la distance topographique car pour toute arborescence de G^∞, l'étiquette du minimum se propage sur toute l'arborescence, et toute branche minimise la distance topographique.

III.3.3.4 Propagation des étiquettes suivant des lignes de plus grande pente composées de pixels pivot

Dans cette partie, nous supposons que l'hypothèse I.1.b est vérifiée : $\Gamma(v)$ est multiple pour certains nœuds v (équation III.26 page 48). Nous démontrons que la phase d'initialisation transforme la topographie de cette image en un graphe équivalent à celui obtenu sous l'hypothèse I.1.a.

Les nœuds v, tel que $\Gamma(v)$ est multiple, sont appelés **pivots**. La phase d'initialisation choisi aléatoirement un unique voisin d'inondation pour ces nœuds, d'où **un indéterminisme** des segments obtenus **si les voisins** $\Gamma(v)$ **appartiennent à des bassins d'attraction distincts.** Leur vision étant limitée à leurs voisins $\mathcal{N}(v)$, ils sont incapables de déterminer à l'initialisation quel est leur bassin d'attraction : un chemin joignant un minimum local à v peut ne pas être minimal.

Pour illustrer ces propos, observons l'exemple de la figure III.5. Soit une image 4-connexe constituée de deux minima $v_\star$ et $v_\diamondsuit$ d'altitude 0 et 3 respectivement. Pour notre algorithme, l'image est composée d'un pixel pivot v (pixel en gras) d'altitude 4. Ses voisins inférieurs $\Gamma(v) = \{v_S, v_O\}$ sont ses voisins Sud et Ouest. Localement, il ne peut savoir que le minima $v_\diamondsuit$ est plus proche (suivant la distance topographique D_h) que le minima $v_\star$. Cependant la convergence de $\lhd$ n'est assurée que si tout nœud a au plus un père. Nous avons donc choisi d'affecter arbitrairement **un unique père** d'un pixel pivot v : le voisin d'inondation $\mathcal{N}^I(v)$ est le premier voisin de plus grande pente parmi $\Gamma(v)$ qui lui envoie son étiquette.

1. Si $v \lhd v_O$, on obtient la figure III.5b.
2. Si $v \lhd v_S$, on obtient la figure III.5c.

0	5	5	6
1	3	**4**	5
2	5	3	5

(a) Image 4-connexe composée de deux minima : pixel(1,1) d'altitude $h = 0$ (étiquette $\star$) et pixel(3,3) d'altitude $h = 3$ (étiquette $\diamondsuit$). Le pixel pivot est représenté en gras ($h = 4$)

$\star$	$\star$	$\star$	$\star$
$\star$	$\star$	$\star$	$\star$
$\star$	$\star$	$\diamondsuit$	$\diamondsuit$

(b) Cas où le pixel pivot est inondé par l'ouest

$\star$	$\star$	$\diamondsuit$	$\diamondsuit$
$\star$	$\star$	$\diamondsuit$	$\diamondsuit$
$\star$	$\star$	$\diamondsuit$	$\diamondsuit$

(c) Cas où le pixel pivot est inondé par le sud

FIG. III.5 – Illustration du comportement d'un pixel pivot : cas où Γ est multiple.

3. Si les messages arrivent simultanément, on choisi aléatoirement un voisin d'inondation.

Ainsi pour tout nœud u non-minimum, la phase d'initialisation fixe $net_G[u]$ tel que l'origine du **seul** arc entrant de u est le voisin de plus grande pente ($|\Gamma(u)| = 1$), ou d'inondation ($\mathcal{N}^I(u)$).

À la fin de la phase d'initialisation, le réseau G' possède les mêmes caractéristiques que sous l'hypothèse I.1.a, et les propriétés III.7 à III.9 sont vérifiées.

Proposition III.11. *Sous l'hypothèse I.1.b, l'algorithme réordonnancé construit les LPE suivant les lignes de plus grande pente.*

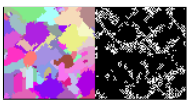
Preuve: Montrons que l'algorithme vérifie bien la définition de l'algorithme de LPE suivant les lignes de plus grande pente présentée dans [MB00] (définition B.42 page 197).

Par hypothèse tout minimum local u possède une étiquette unique $l(u)$. Une fois la phase d'initialisation terminée, le graphe partiel G' est une forêt d'arborescences recouvrante car tout nœud non-minimum a un unique père dont le niveau de gris est strictement inférieur. Or, d'après la proposition III.9, l'étiquette de chaque minimum local se propage sur la totalité de l'arborescence :

$$\forall v \in V \setminus V^*_{\mathcal{M}}, \quad l\left(\mathcal{N}^I(v)\right) = l(v) \tag{III.31}$$

La définition B.42 est donc vérifiée. CQFD

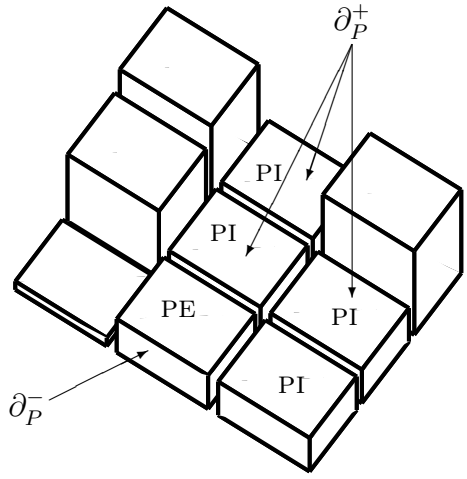
Par ailleurs, il faut souligner que le choix arbitraire sur les pixels pivot ne constitue pas un biais pour notre algorithme car **le choix** parmi tous les voisins de $\Gamma(v)$ est **équiprobable**. Comparativement, les algorithmes séquentiels (par inondation suivant une FAH, par balayages successifs...) font un choix déterministe, ce qui entraine un biais.



III.3.3.5 Propagation des valeurs sur un plateau non minimum

Dans cette partie, les contraintes sur la topographie des voisins de nœuds non-minima sont relâchées : l'hypothèse I.2 s'applique (équation III.27 page 48). L'image n'est pas semi-complète inférieurement, elle comprend des pixels intérieurs.

La pente inférieure (définition B.30 page 192) étant indéfinie pour de tels pixels, ils sont incapables de déterminer leur pixel d'inondation durant la phase d'initialisation car leur vision est réduite à leurs proches voisins. Nous montrons que le processus d'inondation des pixels intérieurs, lorsque la machine passe de l'état MP à l'état NM, permet de construire une arborescence même sur les plateaux non-minima, ce qui assure la convergence de l'algorithme.



PE : Pixel extérieur.
PI : Pixel intérieur.
∂_P^+ : Frontière supérieure.
∂_P^- : Frontière inférieure.

FIG. III.6 – Classification des pixels d'un plateau non-minimum (image 4-connexe).

Soit $P^* = \bigcup_{i\in[1,\beta]} P_i^*$ le sous-graphe des β plateaux non minima $P_i^* = (V_i^*, A_i^*, f, \Omega)$, de $G = (V, A, f, \Omega)$, où pour tout i il existe au moins un nœud intérieur et extérieur dans V_i^* :

$$\forall i \in [1,\beta], \quad \exists v \in V_i^* \mid \Gamma(v) = \emptyset \qquad \exists v \in V_i^* \mid \Gamma(v) \neq \emptyset \tag{III.32}$$

Proposition III.12. *Pour tout plateau non-minimum P_i^*, et pour tout nœud u de G frontière inférieure ($u \in \partial_{P_i^*}^-$), u passe à l'état* NM *dès la phase d'initialisation terminée et n'écoute aucun nœud du plateau.*

Preuve: Comme $\Gamma(u) \neq \emptyset$, u est inondé durant la phase d'initialisation. CQFD

Proposition III.13. *Tout pixel intérieur est inondé par un pixel frontière inférieure.*

Preuve: Montrons par l'absurde qu'un nœud intérieur ne peut rester à l'état MP.

Soit $P_i^* = (V_i^*, A_i^*, f, \Omega)$ un plateau non-minimum quelconque.La phase d'initialisation une fois terminée, V_i^* est composé de deux sous-ensembles :

$$V_i^{\mathrm{NM}} = \partial_{P_i^*}^- \tag{III.33}$$

$$V_i^{\mathrm{MP}} = V_i^* \setminus V_i^{\mathrm{NM}} \tag{III.34}$$

Au cours de la relaxation, seuls les nœuds à l'état MP peuvent passer à l'état NM : l'ensemble V_i^{MP} est de cardinal décroissant et converge vers $V_i^{\text{MP}\infty}$. V_i^{NM} et V_i^{MP} étant complémentaires sur V_i^*, V_i^{NM} converge vers $V_i^{\text{NM}\infty} = V_i^* \setminus V_i^{\text{MP}\infty}$.
Supposons que $V_i^{\text{MP}\infty}$ soit non vide : $|V_i^{\text{MP}\infty}| \neq 0$. Soit $\mathcal{N}^{P_i^*}(v)$ le voisinage de v dans le plateau P_i^*, alors :

$$\forall v \in V_i^{\text{MP}\infty}, \quad \mathcal{N}^{P_i^*}(v) \subset V_i^{\text{MP}\infty} \tag{III.35}$$

car tout nœud MP conserve sa connexité originale, et aucun nœud ne peut se faire inonder.
Soient $u \in V_i^{\text{NM}\infty}$ (non vide par construction, proposition III.12) et $v \in V_i^{\text{MP}\infty}$. P_i^* étant un plateau, il existe au moins un chemin $\pi = (u_0 = u, u_1, \ldots, u_N = v)$ sur P_i^*, d'où :

$$\exists n \in]0, N] \mid \forall j \geqslant n, u_j \in V_i^{\text{MP}\infty} \tag{III.36}$$

Soit n_0 le minorant vérifiant cette propriété, alors $u_{n_0} \in V_i^{\text{MP}\infty}$, et $u_{n_0-1} \in V_i^{\text{NM}\infty}$. Or $u_{n_0-1} \in \mathcal{N}^{P_i^*}(u_{n_0})$, donc contredit la relation III.35. L'ensemble $V_i^{\text{MP}\infty}$ est donc vide. CQFD

Définition III.15 ($\lhd$, la relation «a été inondé par»). *Soit la relation d'antériorité* ***«a été inondé par»*** *réalisée par $\lhd$. Cette relation d'antériorité est* ***transitive*** *($u \lhd v$ et $v \lhd w \Rightarrow u \lhd w$) et* ***antiréflexive*** *($u \lhd u$ est faux).*

Remarque: Pour passer à l'état NM, un nœud doit se faire inonder par un nœud étant déjà dans cet état.

Proposition III.14. *Tout sous-graphe $P_i^{*\infty}$ est acyclique.*

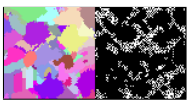
Preuve: Démontrons par l'absurde qu'il n'existe pas de chemin formant un cycle.
Supposons qu'il existe un chemin $\pi_{\circlearrowleft} = \{u_1, u_2, \ldots, u_N\}$ de cardinal $n > 1$ contenu dans $P_i^{*\infty}$ tel que $u_1 = u_N$. Selon la proposition III.13, tous les nœuds u_n sont à l'état NM, et selon la remarque précédente, ils vérifient :

$$\forall n \in [2, N], \quad u_n \lhd u_{n-1} \tag{III.37}$$

Par application de la transitivité, $u_N \lhd u_1$. Ce qui contredit l'antiréflexivité. CQFD

Proposition III.15. *Chaque nœud de la frontière inférieure d'un plateau non-minimum est la racine d'une arborescence sur ce plateau.*

Preuve: Montrons que pour tout sous-graphe $P_i^{*\infty}$, tout nœud de la frontière inférieure n'a pas de père dans $P_i^{*\infty}$, et que tout nœud intérieur en a un et un seul.
Selon les propositions III.13 et III.14, le graphe converge et tous les nœuds sont à l'état NM. Tous les nœuds de $P_i^{*\infty}$ n'ayant qu'un père au plus, le sous-graphe $P_i^{*\infty}$ ne comportant pas de cycle, et les pixels frontière inférieure n'ayant pas de père dans $P_i^{*\infty}$ (proposition III.12), $P_i^{*\infty}$ est une forêt d'arborescence de racines les pixels frontières inférieure $\partial_{P_i^*}^-$. CQFD



Proposition III.16. *Le processus d'inondation d'un plateau non-minimum converge.*

Preuve: Le sous-graphe $P^{*\infty}$ étant une forêt d'arborescences, la propagation des données selon l'opérateur $\lhd$ se termine (proposition III.4 page 45). CQFD

Proposition III.17. *Sous l'hypothèse I.2, l'algorithme converge.*

Preuve: Montrons par l'absurde que le raccord des arborescences créees sur les plateaux non-minima et celles de plus grande pente ne génère pas de cycle.

Par construction de l'algorithme, lorsque tous les pixels non-minima ont été inondés, il existe une relation d'ordre sur l'altitude des nœuds de tout chemin :

$$\forall \pi = \{u_0, u_1, \ldots, u_N\} \mid \forall n \in [0, N-1], u_n \in V \text{ et } u_n = \mathcal{N}^I(u_{n+1}) \quad \text{(III.38)}$$

alors

$$\forall n \in [0, N-1], \quad h(u_n) \leqslant h(u_{n+1}) \quad \text{(III.39)}$$

De plus, l'égalité est uniquement vérifiée sur les plateaux.

Supposons qu'il existe un cycle $\pi_{\circlearrowleft} = \{u_0, u_1, \ldots, u_N\}$ avec $u_0 = u_N$ et $N > 1$, alors $h(u_0) = h(u_N)$ et d'après l'équation III.39, $\forall n \in [0, N], h(u_n) = h(u_0)$. Donc $\pi_{\circlearrowleft}$ est strictement contenu dans un plateau, en contradiction avec la proposition III.14.

Le graphe G^∞ est une forêt d'arborescences car tout nœud non-minimum n'a qu'un père et un seul (comportement de net_G §III.3.3.1 page 45 et proposition III.13), seuls les minima n'ont pas de père, et le graphe est acyclique. La convergence de l'opérateur d'inondation $\lhd$ (proposition III.4) assure la convergence de la propagation des données dans G^∞. CQFD

Proposition III.18. *Si au moins deux pixels sur la frontière inférieure d'un même plateau non-minimum ne sont pas attirés par le même minimum, alors la ligne de partage des eaux déterminée par l'algorithme est située sur ce plateau.*

Preuve: Montrons qu'une étiquette venant du plateau non-minimum ou d'un autre bassin d'attraction ne peut se propager sur la totalité du plateau.

Pour tout i, soit $\partial^-_{P^{*i}}$ la frontière inférieure de P^{*i}. Suivant la proposition III.15, $P^{*\infty}_i$ est une forêt d'arborescences de racines les nœuds de $\partial^-_{P^{*i}}$. Soit u et v deux nœuds de $\partial^-_{P^{*i}}$ attirés par deux minima différents. À la convergence, tout nœud de l'arborescence ayant pour racine u (resp. v) aura l'étiquette de u (resp. v). Le plateau comprendra donc deux arbres d'étiquette différente, la LPE est alors placée entre deux nœuds d'arborescences différentes. CQFD

Remarque: Une interprétation ou corollaire de la proposition III.18 consiste à dire que la ligne de partage des eaux ne peut redescendre le relief.

Conclusion. L'algorithme construit des lignes de plus grande pente hors des plateaux non-minima (hypothèse I.1), des arborescences sur les plateaux non-minima (proposition III.15), et la connection des lignes de plus grande pente aux nœuds frontière inférieure et supérieure ne crée pas de cyle (proposition III.17). Seuls les minima locaux n'ont pas de père, donc l'algorithme construit une forêt d'arborescences de racine ces minima.

La LPE étant placée sur ces plateaux si deux pixels frontière inférieure sont attirés par deux minima différents (proposition III.18), l'algorithme effectue bien la segmentation de l'image sous l'hypothèse I.2.

III.3.3.6 Unification des étiquettes d'un plateau minimum

Dans cette partie, nous supposons que l'hypothèse II est vérifiée : les minima de l'image peuvent être des plateaux (équation III.28 page 48).

Afin que l'algorithme réordonnancé respecte la définition d'un algorithme de LPE suivant les lignes de plus grande pente (définition B.42 page 197), chaque minimum de l'image doit correspondre à une étiquette unique, et tout pixel non-minimum doit posséder la même étiquette que celle de son voisin de plus grande pente (suivant un voisinage généralisé, équation B.39 page 195). Nous montrons que les minima ne sont jamais inondés (chaque région correspond à un unique minimum de l'image), que tout pixel d'un plateau minimum possède au point de convergence une même étiquette (tout minimum possède une étiquette unique), et que ce processus d'unification converge (assurance d'un résultat).

Soit ${G^*_{\mathcal{M}}}^i = ({V^*_{\mathcal{M}}}^i, {A^*_{\mathcal{M}}}^i, f, \Omega)$ le $i^{\text{ème}}$ plateau minimum de G d'altitude $h_{{G^*_{\mathcal{M}}}^i}$.

Proposition III.19. *Tous les nœuds d'un plateau minimum après initialisation sont à l'état* MP *et écoutent uniquement les nœuds de ce plateau.*

Preuve: Montrons que pour tout nœud minimum, l'origine de tous ses arcs entrants sont tous et uniquement ses voisins du plateau.

Pour tout i et tout $u \in {V^*_{\mathcal{M}}}^i$, soit $v \in \mathcal{N}(u)$ le $j^{\text{ème}}$ voisin de u. Par construction de l'algorithme (comportement de net_G §III.3.3.1 page 45) :

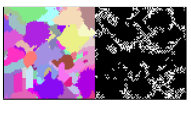
- si $h(v) = h(u)$, alors $net_G[u][j] = 1$.
- si $h(v) > h(u)$, alors $net_G[u][j] = 0$.
- le cas $h(v) < h(u)$ est impossible car u est minimum, u ne peut pas passer à l'état NM.

CQFD

Proposition III.20. *Les sous-graphes ${G^*_{\mathcal{M}}}^i$ après initialisation (i.e. tout nœud de ${G^*_{\mathcal{M}}}^i$ n'est plus à l'état* INIT*) sont des sous-graphes isolés dans G et fortement connexes.*

Preuve: Montrons que seules les données du plateau peuvent circuler, et qu'aucun arc n'est supprimé.

Comme tous les arcs d'origine un pixel voisin non-minimum de la frontière sont supprimés, ${G^*_{\mathcal{M}}}^i$ est isolé : il ne peut pas recevoir de données extérieures. Comme



tout nœud u de ${G^*_{\mathcal{M}}}^i$ peut échanger que des données "d'altitude" $h_{{G^*_{\mathcal{M}}}^i}$, il ne peut pas être inondé : $net_G[u]$ après initialisation est invariant.
Donc (proposition III.19) aucun arc dans ${A^*_{\mathcal{M}}}^i$ n'est supprimé. Donc pour tous nœuds u et v de ${V^*_{\mathcal{M}}}^i$, il existe un chemin $u \rightsquigarrow v$ dans ${G^*_{\mathcal{M}}}^i$. CQFD

Conséquences:
- Comme tout nœud minimum ne peut pas recevoir de données "d'altitude" strictement inférieure, il ne peut pas passer à l'état NM. Donc tout au long du processus de relaxation du réseau, les plateaux minima ${P^*_{\mathcal{M}}}^i$ gardent leur état MP et leur caractère fortement connexe.
- Par hypothèse, tout pixel possède une étiquette unique à l'état initial. Comme un plateau minimum ne peut-être inondé par un autre minimum, toute **fusion** de deux régions (attribution d'une même étiquette à deux bassins d'attraction différents) est **impossible**.

Pour qu'une région ne soit pas subdivisée, tout pixel d'un plateau minimum doit posséder une étiquette identique : l'étiquette du plateau. La proposition suivante assure l'existence d'une étiquette par plateau.

Proposition III.21. *Tout pixel minimum est étiqueté par l'étiquette minimale du plateau auquel il appartient.*

Preuve: La justification découle directement du caractère fortement connexe du plateau minimum (proposition III.20) et par application de la ⊚`-direct-association` (équation III.17 et proposition III.2 page 42) sur tout nœud à l'état MP. CQFD

À présent, généralisons les minima locaux vers des plateaux de minima. Comme l'étiquette d'un plateau minimum est unifiée au point de convergence, construisons un graphe équivalent à G où les nœuds d'un plateau minimum sont substitués par un unique nœud.

Pour tout plateau minimum ou minimum local ${G^*_{\mathcal{M}}}^i$ d'altitude $h_{{G^*_{\mathcal{M}}}^i}$, soit $\mathcal{V}^i_{\mathcal{M}}$ le nœud de même altitude représentant le $i^{\text{ème}}$ minimum, et :

$$\mathcal{V}_{\mathcal{M}} = \bigcup_{i \in [1,\alpha]} \mathcal{V}^i_{\mathcal{M}}$$

l'ensemble des minima (notations utilisées §III.3.3.2).

Pour tout i, soient $A^i_{péri}$ les arcs entrants et sortants de la périphérie du $i^{\text{ème}}$ minimum :

$$A^i_{péri} = \Big\{(u,v) \text{ et } (v,u) \mid u \in \partial_{{G^*_{\mathcal{M}}}^i} \text{ et } v \in \mathcal{N}(u)\ ,\ h(v) > h(u)\Big\} \tag{III.40}$$

et $\mathcal{A}^i_{\mathcal{M}}$ ces mêmes arcs, mais ayant $\mathcal{V}^i_{\mathcal{M}}$ comme origine ou extrémité :

$$\mathcal{A}^i_{\mathcal{M}} = \Big\{(\mathcal{V}^i_{\mathcal{M}},v) \text{ et } (v,\mathcal{V}^i_{\mathcal{M}}) \mid \exists u \in \partial_{{G^*_{\mathcal{M}}}^i} \text{ et } v \in \mathcal{N}(u)\ ,\ h(v) > h(u)\Big\} \tag{III.41}$$

et de même, $\mathcal{A}_{\mathcal{M}}$ les arcs de tous les plateaux :

$$\mathcal{A}_{\mathcal{M}} = \bigcup_{i\in[1,\alpha]} \mathcal{A}^i_{\mathcal{M}}$$

Soit $\mathcal{G} = (\mathcal{V}, \mathcal{A}, f, \Omega)$ le graphe équivalent à G où tous les minima sont substitués par un nœud seul, avec :

$$\mathcal{V} = \mathcal{V}_{\mathcal{M}} \cup \big(V \setminus V^*_{\mathcal{M}}\big) \tag{III.42}$$
$$\mathcal{A} = \mathcal{A}_{\mathcal{M}} \cup A \setminus (A^*_{\mathcal{M}} \cup A^i_{péri}) \tag{III.43}$$

Le graphe $\mathcal{G}$ est un graphe symétrique (tout arc et son opposé sont dans $\mathcal{A}$) composé uniquement de minima locaux $\mathcal{V}_{\mathcal{M}}$, il vérifie l'hypothèse I.

III.3.3.7 Conclusion

À partir d'une hiérarchie d'hypothèses, l'ensemble des topologies (minimum local, lignes de plus grande pente déterministe ou indéterministe, image semi-complète inférieure ou non, plateaux minima) est classifié. Pour chacune d'entre-elles, en relâchant progressivement les contraintes, nous avons montré indépendamment de l'ordonnancement des calculs que **l'algorithme de *Hill-climbing* réordonnancé est un algorithme de segmentation** par construction des LPE suivant les lignes de plus grande pente.

La section suivante complète la pondération Ω du graphe en associant, en plus du coût topographique cost, les temps de transfert t_{prop} entre nœuds voisins.

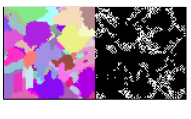
L'image pouvant contenir des pixels intérieurs et les contraintes de vitesse de propagation des pixels extérieurs étant relâchées, ni les voisins de plus grande pente (Γ) ni sa forme généralisée ($\widehat{\Gamma}$, définition B.39 page 195) ne permettent la détermination du voisin d'inondation pour notre algorithme. Nous construirons $\widetilde{\Gamma}$, le voisin de plus grande pente généralisé utilisé par l'algorithme de *Hill-Climbing* réordonnancé.

III.3.4 Comportement temporel

La section précédente démontrait la convergence de l'algorithme et introduisait, entre autres, la notion de «voisin le plus rapide» pour quelques topographies particulières (pixels pivots et plateaux non-minima). Afin d'expliciter cet aspect temporel de l'algorithme, nous formalisons ici les instants d'unification et d'inondation des nœuds du graphe. Ainsi en faisant des hypothèses sur les latences de propagation des données, il est possible d'estimer *a priori* le comportement temporel des nœuds du réseau. Le voisin de plus grande pente $\widetilde{\Gamma}$ adapté aux caractéristiques du réseau est alors défini.

Pondérons l'image digitale $G = (V, A, f, \Omega)$ par le coût topographique **et** par le temps de propagation des données et construisons la fonction de pondération temporelle t_{prop}. Soit Ω la fonction de pondération du réseau :

$$\forall (u,v) \in A \text{ et } x \in \mathbb{S}, \quad \Omega = \big(\text{cost}(u,v), t^x_{prop}(u,v)\big) \tag{III.44}$$



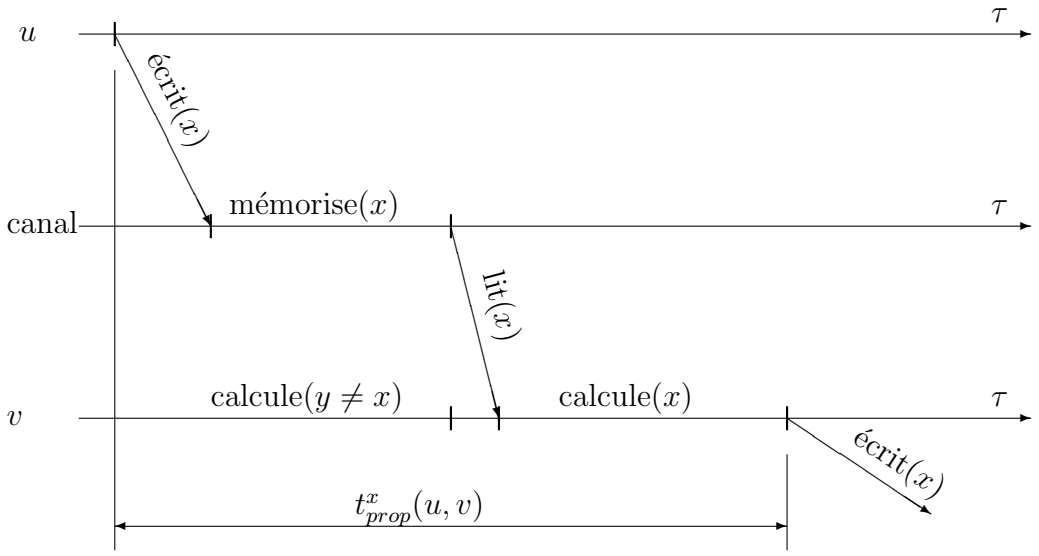


FIG. III.7 – Temps de propagation d'une donnée x.

Pour tout arc (u, v) et pour tout $x \in \mathbb{S}$, $t^x_{prop}(u, v)$ correspond à l'intervalle de temps entre l'écriture de la donnée x par le nœud voisin et la fin du traitement de cette donnée par le nœud courant (figure III.7). Si x ne traverse jamais le nœud v, ou si v n'est pas voisin de u, alors on pose $t^x_{prop}(u, v) = +\infty$. Le nœud v absorbe la donnée x s'il n'écrit jamais x vers ses voisins.

Cette fonction modélise les temps de traitement que nous simulerons lors du chapitre IV. En perspective de l'architecture utilisée, supposons que chaque nœud communique via un canal (l'arc) doté d'une file d'attente de taille infinie de type FIFO (toute donnée écrite est mémorisée et restituée chronologiquement).

Le temps de propagation de la donnée x entre deux nœuds u et v éloignés est noté $t^x_{prop}(u \rightsquigarrow v)$:

$$\forall x \in \mathbb{S}, \forall u \text{ et } v \in V, \quad t^x_{prop}(u \rightsquigarrow v) = \min_{\pi \in \{u \rightsquigarrow v\}} \left(\sum_{n=0}^{|\pi|-1} t^x_{prop}(u_n, u_{n+1}) \right) \tag{III.45}$$

Tout chemin vérifiant cette égalité est appelé **chemin le plus rapide.**

III.3.4.1 Phase d'initialisation

Durant cette phase, chaque nœud envoie ses données à tous ses voisins et attend une donnée de leur part (§III.3.3.1). La durée d'initialisation d'un nœud quelconque v s'exprime par :

$$\forall v \in V, \quad T_{\text{INIT}}(v) = \max_{u \in \mathcal{N}(v)} t^{f[u]}_{prop}(u, v) \tag{III.46}$$

À supposer que le majorant de la fonction t_{prop} soit connu pour tout nœud du réseau, cette relation peut-être utilisée pour déterminer l'instant où tous les nœuds ont fini leur phase d'initialisation.

III.3.4.2 Propagation des données sur une image semi-complète inférieure

Nous supposons l'hypothèse I.1 vérifiée (équation III.24 page 47) : tout pixel non-minimum a au moins un voisin d'altitude strictement inférieure.

Le chemin de convergence de l'algorithme est entièrement défini par la topographie de l'image et par le comportement des pixels pivots. Le voisin d'inondation $\mathcal{N}^I(v)$ de tout pixel pivot v est celui qui vérifie :

$$t_{prop}^{f[\mathcal{N}^I(v)]}(\mathcal{N}^I(v), v) = \min_{u \in \Gamma(v)} t_{prop}^{f[u]}(u, v) \tag{III.47}$$

Si plusieurs nœuds vérifient l'équation III.47, alors un **arbitre** choisit aléatoirement un unique voisin d'inondation.

Soient v un nœud quelconque de la $i^{\text{ème}}$ arborescence de racine $m_i \in V_{\mathcal{M}}^i$ et $\pi = (u_0 = m_i, u_1, \ldots, u_N = v)$ la ligne de plus grande pente joignant ces deux sommets. La date de convergence de v, *i.e.* lorsque le traitement de la donnée $f[m_i]$ est terminé, est :

$$T_{\triangleleft}(v) = \sum_{n=0}^{N-1} t_{prop}^{f[m_i]}(u_n, u_{n+1}) \tag{III.48}$$

Cette équation nous permet de déterminer l'instant où tout nœud a fini de calculer. Si l'on suppose qu'il existe un majorant sur les latences et une longueur moyenne des chemins, il est alors possible d'estimer le temps moyen de segmentation d'une image.

III.3.4.3 Propagation des valeurs sur un plateau non-minimum

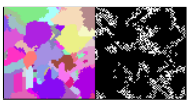
Ici, l'image vérifie l'hypothèse I.2 : des plateaux non-minima composés d'au moins un pixel intérieur existent. À partir de la définition des voisins de plus grande pente généralisé $\widehat{\Gamma}$ (équation B.39 page 195), nous proposons une version adaptée aux réseaux pondérés notée $\widetilde{\Gamma}$.

Soit $P_i^* = (V_i^*, A_i^*, f, \Omega)$ un plateau non-minimum composé d'au moins un pixel intérieur. Le temps de parcours des données d'une frontière inférieure à tout pixel intérieur v est défini par :

$$\forall v \in V_i^* \setminus \partial_{P_i^*}^-, \quad t_{prop}^{f[\partial_{P_i^*}^-]}(\partial_{P_i^*}^- \rightsquigarrow v) = \min_{u \in \partial_{P_i^*}^-} \left(t_{prop}^{f[u]}(u \rightsquigarrow v) \right) \tag{III.49}$$

Nous avons vu que tout pixel intérieur se fait inonder (proposition III.13 page 52) par le premier (§III.2.2.2 page 37) de ses voisins qui lui envoie une donnée où l'altitude est strictement inférieure à celle du plateau. L'équation III.49 formalise l'instant d'inondation de tout pixel intérieur v.

Ce comportement exhibe la notion de «voisin de plus grande pente généralisé»



(définition B.39 page 195) initialement définie par :

$$\widehat{\Gamma}(v) = \left\{ \bigcup_{\substack{u \in \partial_P^- \\ D_P^g(v,u) = D_P^g(v,\partial_P^-)}} \Gamma(u) \right\} \tag{III.50}$$

mais où la distance entre un pixel intérieur et sa frontière est basée sur une "métrique élastique" (la distance entre un nœud extérieur et intérieur varie suivant les expériences et elle n'est pas fondée sur les coordonnées des nœuds).

L'équation III.51 formalise ce nouveau concept où nous substituons la distance géodésique D_P^g contrainte au plateau P par le temps de parcours des données sur ce plateau (équation III.49). L'adaptation de $\widehat{\Gamma}$ à un réseau pondéré est alors définie par $\widetilde{\Gamma}$:

$$\widetilde{\Gamma}(v) = \left\{ \bigcup_{\substack{u \in \partial_{P_i^*}^- \\ t_{prop}^{f[u]}(u,v) = t_{prop}^{f[\partial_{P_i^*}^-]}\left(\partial_{P_i^*}^-, v\right)}} \Gamma(u) \right\} \tag{III.51}$$

Pour tout pixel intérieur v, $\widetilde{\Gamma}(v)$ est un de ses ascendants, c'est-à-dire le voisin de plus grande pente du pixel frontière inférieure $u \in \partial_{P_i^*}^-$ situé sur le chemin d'inondation le plus rapide.

À partir de l'équation III.48, la date de convergence de tout pixel intérieur d'un plateau non-minimum P_i^* est :

$$\forall v \in V_i^*, \quad T_{\triangleleft}(v) = T_{\triangleleft}\left(\widetilde{\Gamma}(v)\right) + t_{prop}^{f[m_j])}\left(\widetilde{\Gamma}(v), v\right) \tag{III.52}$$

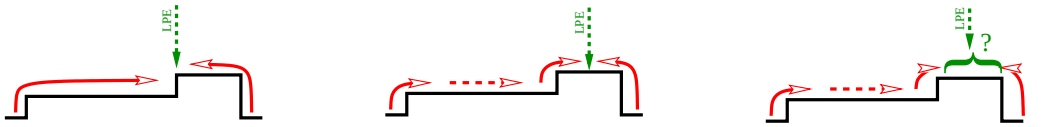
avec m_j le minimum d'attraction de $\widetilde{\Gamma}(v)$.

Remarques:

- Si pour tous nœuds voisins u et v de P_i^* la latence $t_{prop}^{f[u]}(u,v)$ est constante et égale à 1, on a $\widetilde{\Gamma} = \widehat{\Gamma}$ car le temps de propagation et la distance géodésique sont équivalents.
- Si $\widetilde{\Gamma}(v)$ n'est pas un singleton, alors v se comporte comme un pixel pivot. L'algorithme choisit arbitrairement un unique voisin d'inondation.
- L'ébranchage d'un plateau non-minimum fait apparaître des points d'articulation (définition A.22 page 179). La ligne de partage des eaux d'un plateau non-minimum peut-être vue comme la ligne joignant tous ces points.

Nous présentons le comportement de quelques mises en œuvres de l'algorithme de *Hill-Climbing* et d'inondation uniforme pour une image composée de pixels intérieurs.

Dans l'appproche séquentielle de l'algorithme de *Hill Climbing* (figure III.8a), si l'image n'est pas auparavant transformée en une image semi-complète inférieure, la



(a) *Hill-Climbing* séquentiel (§II.4.1.4 page 25)

(b) Inondation uniforme (§II.4.2.1 page 27)

(c) *Hill-Climbing* réordonnancé

FIG. III.8 – Propragation des étiquettes suivant différents processus.

LPE sur un plateau non-minimum tente de s'approcher du SKIZ binaire (figure B.7 page 194) où les marqueurs sont les minima, et ce bien sûr dans les limites du plateau. En effet, tous les voisins des pixels frontière des minima **et uniquement ceux-la** initialisent la propagation des données dans l'image. Le processus de segmentation consiste à déterminer séquentiellement pour tout nœud v initiateur ses voisins de plus grande pente non-traités, à leurs attribuer l'étiquette $l(v)$, à les marquer à leur tour initiateur, et à mettre au repos v. La vitesse de propagation des données est indépendante du relief, un plateau non-minimum séparé par plusieurs minima est étiqueté majoritairement par le minimum le plus proche.

L'algorithme par inondation uniforme (figure III.8b) place les ligne de partage des eaux au milieu du plateau. Comme les pixels sont traités dans l'ordre croissant de leur altitude, le front d'inondation d'un plateau non-minimum s'effectue à vitesse constante à partir de tous les pixels de la frontière inférieure. Donc la LPE est située à équidistance (suivant la distance géodésique) des pixels extérieurs.

Pour la version réordonnancée que nous proposons (figure III.8c), l'inondation d'un plateau non-minimum s'effectue à partir de tous les pixels de la frontière inférieure comme ci-dessus, mais **la vitesse de propagation est variable** et inconnue. C'est le pixel frontière le plus proche (suivant la métrique t_{prop}) qui influe la position de la LPE. Quant à la localisation de la LPE, nous pouvons seulement affirmer qu'elle est située sur ce plateau (proposition III.18).

III.3.4.4 Unification des étiquettes d'un plateau de minima

L'hypothèse II (équation III.28 page 48) est vérifiée. Le réseau étant pondéré par un temps variable de transfert des données, le chemin le plus court (pour la distance géodésique) ne correspond pas forcément au chemin le plus rapide. Déterminons le temps nécessaire pour unifier les étiquettes d'un plateau minimum.

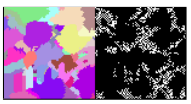
Soit $u^i_{\mathcal{M}}$ le nœud d'étiquette minimale $l^i_{\mathcal{M}}$ du plateau minimum $G^{*\,i}_{\mathcal{M}} = (V^{*\,i}_{\mathcal{M}}, A^{*\,i}_{\mathcal{M}}, f, \Omega)$. Le temps de convergence de tout nœud v de ce plateau est :

$$\forall i \in [1,\alpha] \text{ et } \forall v \in V^{*\,i}_{\mathcal{M}}, \quad T_{\text{MINI}}(v) = t^{f[u^i_{\mathcal{M}}]}_{prop}(u^i_{\mathcal{M}}, v) \tag{III.53}$$

Le temps d'unification des étiquettes de ce plateau est alors :

$$\forall i \in [1,\alpha], \quad T^i_{\text{MINI}} = \max_{u \in V^{*\,i}_{\mathcal{M}}} (T_{\text{MINI}}(u)) \tag{III.54}$$

L'équation III.55 correspond au temps nécessaire pour étiqueter tous les plateaux



minima :

$$T_{\text{MINI}} = \max_{i \in [1,\alpha]} \left(T^i_{\text{MINI}} \right) \tag{III.55}$$

À cet instant il est possible que de nombreuses régions soient déjà segmentées, comme par exemple tous les bassins d'attraction ayant une surface inférieure au plateau de minima de surface la plus grande (on suppose ici des latences de propagation homogènes).

L'équation III.55 peut être vue comme le temps nécessaire à la détection et étiquetage des minima pour un algorithme classique. Pour ce dernier, la phase de propagation des étiquettes ne peut commencer qu'à cet instant.

Cet aspect de l'algorithme réordonnancé montre l'intérêt de désynchroniser les traitements et de supprimer les points de synchronisation globaux.

III.3.4.5 Conclusion sur le comportement temporel

La présentation de l'aspect temporel des propagations cherche à formaliser les notions telles que «le chemin le plus rapide est...», «le premier voisin qui...», «dès que le pixel a fini...», «le plateau minimum est unifié quand...» *etc.*

Le chronogramme du transit d'une donnée dans un nœud (figure III.7) est une première approche du comportement des processeurs que nous modéliserons au chapitre suivant. Ne faisant aucune hypothèse sur les temps de communication et de traitement des pixels, cette présentation ne peut pas prétendre être complète. Toutefois, si l'architecture cible nous permet d'imposer des contraintes temporelles sur ces latences, l'ensemble des équations proposées dans cette section sont des éléments d'analyse sur le comportement des pixels et sur les temps de segmentation d'une image.

Nous avons traité l'ensemble des configurations possibles du relief topographique. Chaque chemin d'inondation traverse soit un plateau minimum, soit une suite de pixels suivant une ligne de plus grande pente, soit des pixels pivots ou soit un plateau non-minimum. Ainsi en fonction du chemin emprunté entre la racine et une feuille d'une arborescence, la spécification temporelle de la propagation des données sur ce chemin est déterminée par les différentes relations proposées.

III.4 Présentation d'un exemple complet.

Afin d'expliciter l'ensemble des processus énoncés sections III.2 page 34 et III.3 page 39, cette partie présente le résultat de plusieurs segmentations d'une même image de dimensions 5×5 (figure III.9).

La figure III.10 présente un exemple d'évolution possible du graphe au cours de l'inondation, la latence de propagation des données étant aléatoire. Pour chaque nœud v, la donnée $f[v]$ est représentée par deux nombres : l'étiquette et en indice l'altitude. La couleur de ces nombres correspond à l'état de chaque pixel : le bleu correspond à l'état INIT, le rouge à l'état MP et le noir à l'état NM. La variable parallèle net_G est symbolisée par les arcs du graphe. Comme $net_G[v]$ représente les

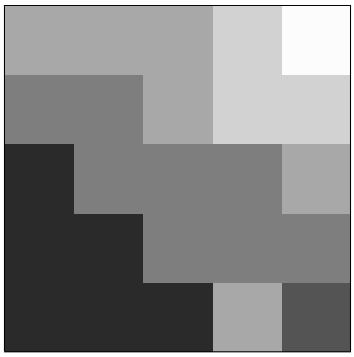


FIG. III.9 – Image exemple constituée d'un plateau minimum (en bas à gauche) et d'un minimum local (coin en bas à droite).

masques des ports **entrants** du nœud v, la destination d'une flèche indique l'état à 1 du $i^{\text{ème}}$ bit de $net_G[v]$.

À l'état initial (figure III.10a), chaque nœud possède une étiquette unique (l'adresse du nœud pour un balayage vidéo direct) et le niveau de gris (en indice) correspondant à l'image de la figure III.9. Un plateau de nœuds minima est situé dans le coin en bas à gauche (altitude 1) et un minimum local en bas à droite (étiquette 25, altitude 2).

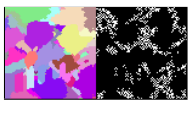
Le graphe est une grille 4-connexe : à cet instant tous les nœuds communiquent avec tous ses voisins, la phase d'initialisation est amorcée (tous les nœuds sont bleus).

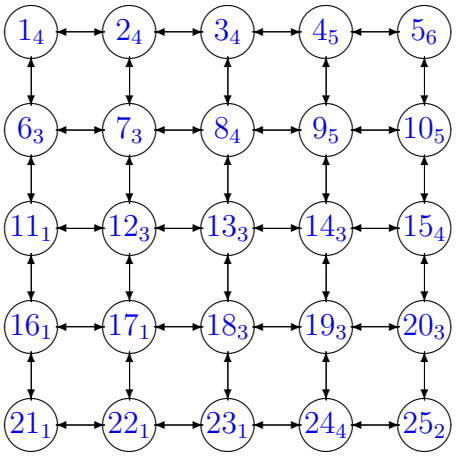
La figure III.10b présente l'état du réseau après quelques échanges de données. On remarque que quelques nœuds sont dans une phase de relaxation (pixels non-minima en noir et pixels à l'état MP en rouge), alors que d'autres sont toujours dans une phase d'initialisation (pour eux, tous les bits de net_G sont encore à 1).

À la figure III.10c, tous les nœuds ont fini de s'initialiser et l'unification des étiquettes du plateau minimum est quasiment terminée. Le plateau non-minimum d'altitude 3 (encore à l'état MP) au centre du réseau est en cours d'inondation, les pixels extérieurs sont à cet instant les nœuds d'altitude 3 et d'étiquettes 17 et 25. Le pixel (4,4) d'étiquette 13 va bientôt recevoir deux fronts d'altitude 1 (par sa gauche) et d'altitude 2 (par sa droite). Si les fronts arrivent simultanément, c'est un pixel intérieur et pivot.

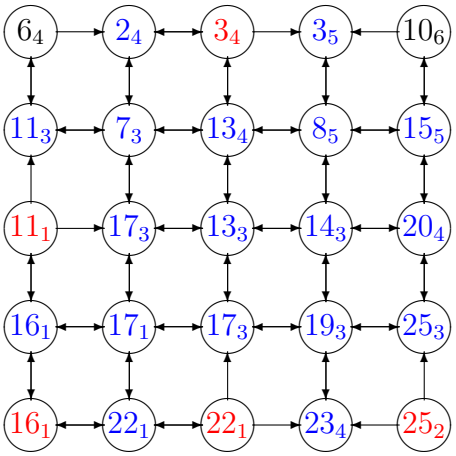
La figure III.10d indique que le front d'inondation est arrivé en premier par la droite (ou choix aléatoire ?) : le pixel de coordonnées (4,4) n'écoute que son voisin d'inondation situé à l'est. Une explication possible mais non justifiée est que le nœud de coordonnée (5,4) (dernière colonne, avant dernière ligne) propage plus rapidement les signaux étant donné que sa charge de calculs est allégée (les pixels situé sur les bords ont moins de voisins).

Le pixel de coordonnées (4,3) illustre bien le comportement du pixel pivot. Il est sur le point de recevoir à sa gauche une étiquette 11 et altitude 1, et au sud une étiquette 25 et altitude 2. Suivant son choix, tous ses nœuds enfants (ici un seul de coordonnées $(4, 2)$) s'attribuerons les données d'inondation choisies.

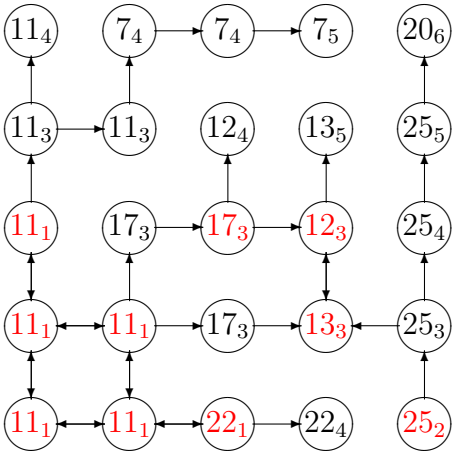




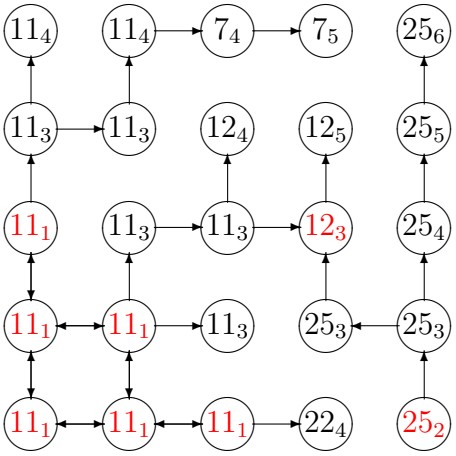
(a) Réseau à l'état initial



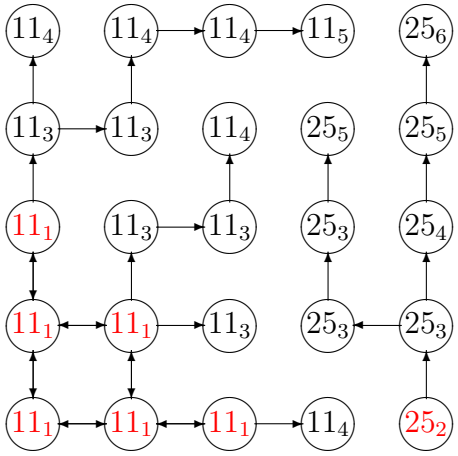
(b) Initialisation en cours



(c) Pixel pivot d'étiquette 12 (en rouge) potentiellement inondable par les minima d'étiquettes 11 et 25



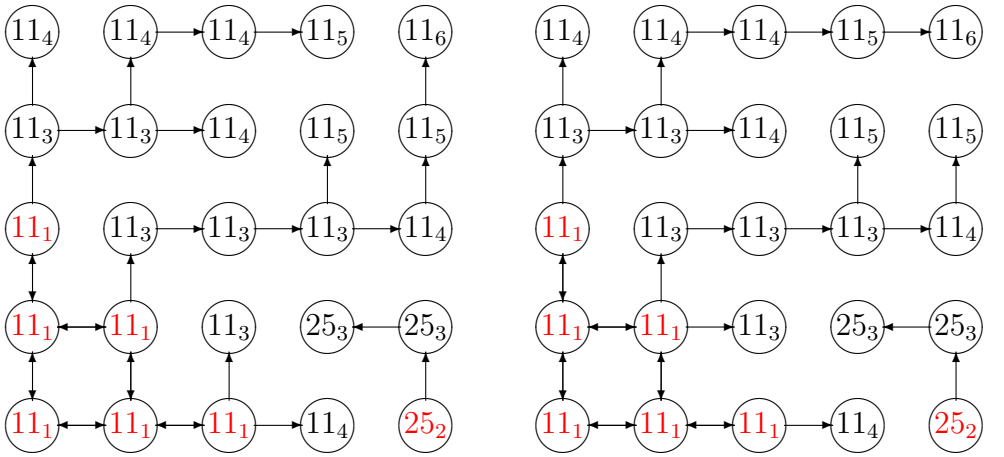
(d) L'inondation du plateau non-minimum d'altitude 3 est quasiment terminée



(e) Réseau à l'état final : le pixel pivot de coordonnées (4,3) a choisi le minimum d'étiquette 25 (premier arrivé ou choix arbitraire)

FIG. III.10 – Exemple d'inondation d'une image 5×5. Étiquettes en large police et altitudes en indice, connectivité représentée par une flèche entrante, et état de la machine en couleur (bleu↔INIT, rouge↔MP et noir↔NM).

Enfin, la figure III.10e présente le réseau au point de convergence, les LPE correspondantes sont celles présentées à la figure III.13a. Nous observons qu'un plateau minimum est bien une composante fortement connexe où tous les nœuds sont à l'état MP. À partir de chaque nœud frontière supérieure des minima, une branche acyclique suivant le relief topographique et suivant $\widetilde{\Gamma}$ est construit. Si les deux minima sont remplacés par deux nœuds, le graphe équivalent est bien une forêt recouvrante de deux arborescences.



(a) les pivots (5,1) et (3,4) sont inondés par le sud

(b) les pivots (5,1) et (3,4) sont inondés par l'ouest

FIG. III.11 – Exemples de partitions finales identiques pour deux constructions différentes des chemins (seul le pixel pivot (5,1) a choisi un voisin d'inondation différent).

La figures III.11 présente deux graphes différents au point de convergence. Particulièrement pour le pixel de coordonnées (5,1), l'indéterminisme des temps de propagation des données implique des chemins de convergence différents. Toutefois les régions obtenues sont identiques (figure III.13b), et la variation du comportement du pixel pivot de coordonnées (5,1) n'a eu aucune influence.

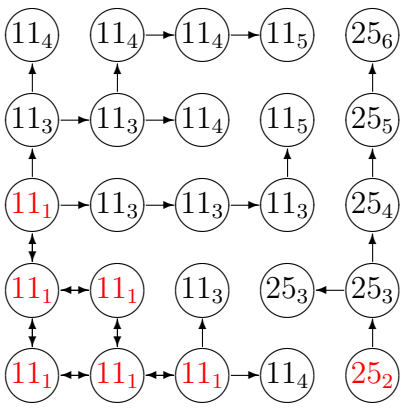
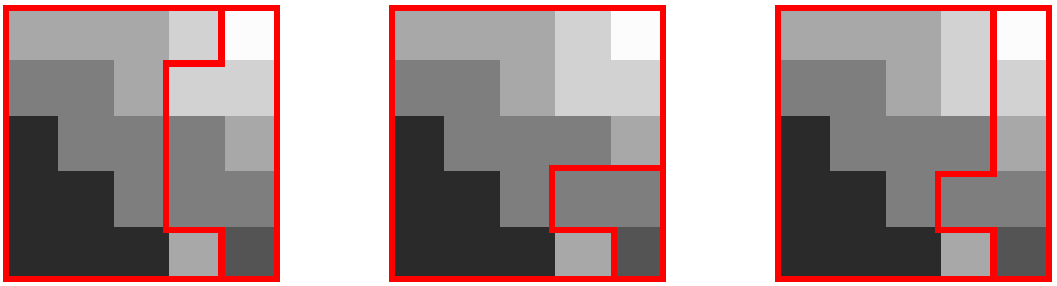


FIG. III.12 – Cas où le nœud (5,3) est initialisé par le pixel sud en premier.

Par contre pour le graphe de la figure III.12, la partition finale est différente. Nous voyons ici l'influence du pixel pivot de coordonnées (5,3) : ses voisins de plus grande pente sont à l'ouest et au sud, et pour cette simulation il choisi de se faire inonder par son voisin du sud.



(a) LPE associée au graphe de la figure III.10

(b) LPE associée au graphe de la figure III.11

(c) LPE associée au graphe de la figure III.12

FIG. III.13 – Lignes de Partage des Eaux correspondant à différentes simulations.

La figure III.13 montre comment sont construites les lignes de partage des eaux "d'épaisseur nulle" correspondant aux trois simulations présentées précédemment (figures III.10 à III.12) : elles sont situées entre deux pixels d'étiquette différente. Ces images nous permettent également de visualiser plus aisément les bassins d'attractions de l'image.

III.5 Conclusion

Dans ce chapitre, nous avons déterminé que l'algorithme de *Hill-Climbing* permet d'allier faisabilité d'implantation dans une chaîne de codage vidéo orienté objet et mise en œuvre parallèle efficace. L'état de l'art sur les implantations parallèles de cet algorithme ou équivalent montre une inadéquation entre l'architecture (implantation délicate des files d'attentes) et l'algorithme (points de synchronisation globaux pénalisants). C'est pourquoi, nous proposons un algorithme aux traitements locaux et désynchronisés où ne subsiste qu'un seul point de synchronisation globale : le point de convergence. La recherche des minima, étiquetage des plateaux minima et processus de segmentation sont réalisés simultanément au niveau de chaque pixel.

L'introduction de ce nouvel algorithme exibe une machine à trois états composant chaque nœud d'une grille digitale. Nous décrivons le comportement des pixels suivant leur topographie locale : minimum local, pixel intérieur minimum ou non-minimum, pixel possédant un unique voisin de plus grande pente et pixel pivot.

Ensuite, nous démontrons la convergence de l'algorithme afin d'assurer l'existance d'un résultat quelque soit l'image à segmenter. À partir d'un modèle mathématique de calcul cellulaire, les réseaux associatifs, et d'une hiérarchie d'hypothèses caractérisant la topographie de l'image, nous démontrons que l'algorithme de *Hill-Climbing* proposé construit un sous-graphe acyclique sur tout plateau non-minimum constitué de pixels intérieurs, et une forêt d'arborescences minimisant la distance topographique sur les reliefs pentus. L'unification des étiquettes sur un plateau minimum est garantie. Au

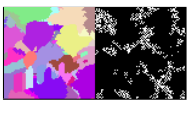
point de convergence, le graphe est une forêt d'arborescences où chaque racine est un nœud modélisant le plateau minimum associé.

Seul le comportement aléatoire des pixels pivot, nœud ayant au moins deux voisins de plus grande pente, empêche la minimisation de la distance topographique. Toutefois, de par le choix équiprobable du voisin d'inondation, ce comportement n'introduit pas de biais dans la détermination des LPE, contrairement aux algorithmes séquentiels sensibles à ces mêmes points où le déterminisme de l'ordre de traitement des pixels implique un écart constant de positionnement des LPE.

Il est important de noter que cette démonstration est indépendante de l'ordre de traitement des pixels. **C'est un algorithme asynchrone** [Rob97] car le chemin de convergence est inconnu et chaque donnée est mise à jour indépendamment des autres. De bonnes performances en termes de temps de calcul sont donc attendues.

Si des plateaux non-minima existent, l'affirmation «pour un nœud intérieur, le nœud frontière le plus rapide est le nœud frontière le plus proche suivant la distance géodésique» est fausse en général car les chemins d'inondation sur un plateau non-minimum sont soumis à des comportements locaux propres au réseau. Toutefois, plus les vitesses de propagation sont homogènes, plus la métrique temporelle t_{prop} se rapproche, à un facteur près, de celle géodésique D^g. Nous le confirmerons par simulation (§IV.4.1 page 95).

L'algorithme de *Hill-Climbing* réordonnancé étant désormais validé, le chapitre suivant démontre sa faisabilité d'implantation par une architecture parallèle.



Chapitre IV

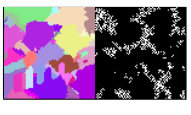
Modélisation pour la validation par simulation et l'exploration d'architectures

A partir d'un état de l'art sur les algorithmes parallèles de segmentation d'images, le chapitre III a présenté l'intérêt de réordonnancer l'algorithme de segmentation par ascension de colline (*Hill-Climbing*). La modélisation théorique des flux de données dans un graphe représentant le réseau dynamique de processeurs, a montré la convergence de l'algorithme et la justesse des régions obtenues. À présent, nous disposons d'un algorithme de segmentation où les traitements sont localisés.

Dans ce chapitre, nous présentons un aspect outil de l'étude : nous établissons un environnement de travail pour la validation du couple algorithme-architecture par simulation. Nous explorons les architectures capables d'exécuter efficacement et correctement cet l'algorithme. Pour cela, un environnement de simulation est établi, nous permettant de contrôler le flux des données dans l'architecture, et de vérifier les résultats de segmentation.

Le couple algorithme-architecture une fois validé, nous aborderons au chapitre V l'aspect implantation matérielle du circuit.

Une première exploration des grands types d'architectures parallèles existants nous permet à la section IV.1, de déterminer celle qui est la mieux adaptée à notre algorithme. Ensuite à la section IV.2, nous détaillons les caractéristiques des architectures choisies. Dans des perspectives de faisabilité d'implantation matérielle, les granularités allant de la plus fine à la plus grosse sont présentées. À partir de ces modèles, la section IV.3 présente un environnement de prototypage *hardware* rapide : SystemC™. Les modèles, une fois établis dans cet environnement, sont validés dans la section IV.4 par simulation du flux des données dans l'architecture. La section IV.5 présente une analyse des résultats de simulation pour l'ensemble des architectures étudiées. À partir de ces résultats, nous estimons leurs caractéristiques afin de déterminer les meilleures architectures alliant temps de traitement et charge de calcul. Enfin, la conclusion de ce chapitre présentée dans la section IV.6 fournit une synthèse sur l'environnement de simulation ainsi que sur les caractéristiques des architectures étudiées.



IV.1 Les architectures de machines parallèles

Le traitement d'images est souvent caractérisé par une faible complexité algorithmique au niveau pixel (souvent limitée à des accumulations ou recherches de maximum ou minimum) et par une grande quantité de données à traiter. Dans le cadre de ces études, nous cherchons une architecture parallèle adaptée à ces caractéristiques. Au cours de la présentation de la taxinomie des différents types d'architectures parallèles, nous évaluons leurs principaux atouts et faiblesses en vue d'une telle application. Nous justifions l'implantation SPMD *Single Program Multiple Data stream* de l'algorithme de *Hill-Climbing* sur une grille régulière de processeurs élémentaires.

La parallélisation des traitements est motivée par l'augmentation du débit de calcul (en faire plus en un temps donné) ou la réduction du temps de calcul (pour une même taille du problème, déterminer un résultat le plus rapidement possible) [Cap98]. Pour cela, trois sources de parallélisme sont possibles :

1. **Le parallélisme de données :** Présence de groupes de données devant subir le même calcul ;
2. **Le parallélisme de contrôle :** Présence de parties (séquences d'opérations, tâches) indépendantes dans un programme ;
3. **Le parallélisme de flux :** Traitement d'un flux séquentiel de données, chacune traitée par une suite de sous-traitements (travail à la chaîne).

La granularité est un terme souvent employé pour préciser si le degré de parallélisation d'un algorithme est faible (proche d'une implantation séquentielle) ou fort (proche d'une implantation cellulaire). Pour une taille fixée d'un problème, la **granularité** correspond à la proportion de traitements attribués à chaque élément de calcul pour résoudre ce problème. Un système parallèle sera dit à **fine granularité** si un grand nombre de processus combinatoires de petite taille, parfois désignés par le terme **grain**, participe à la résolution du problème. Par exemple, une implantation parallèle à fine granularité pour le traitement d'images consiste à associer à chaque pixel, un processeur. À l'opposé, une **grosse granularité** correspond à une faible division de la quantité de traitements dont l'asymptote est un système séquentiel (un seul processeur).

IV.1.1 Approche de la taxinomie des machines parallèles

Les caractéristiques d'une architecture parallèle dépendent de ses ressources. Les ressources fondamentales sont les unités de contrôle (gestion du flux d'instructions), les mémoires (stockage et émission de données), les unités de traitements (analyse et combinaison des données), et le réseau d'interconnexions (gestion du mouvement des données).

La classification proposée par Flynn [Fly72] dans les années 70 est à ce jour toujours utilisée pour caractériser les différents types d'architecture d'ordinateurs. Cette classification est fondée sur le nombre de flux de données (*Single Data stream* ou *Multiple Data stream*) et sur le nombre de flux d'instructions (*Single Instruction*

stream ou *Multiple Instruction stream*) présents sur une architecture donnée. De la combinaison de ces possibilités résulte quatre catégories :

SISD *Single Instruction stream, Single Data stream* : modèle de fonctionnement séquentiel où chaque instruction traite une seule donnée.

SIMD *Single Instruction stream, Multiple Data stream* : modèle où les unités de traitement sont répliquées, mais sont gérées par une seule unité de contrôle (processeurs Pentium utilisant des instructions MMX *MultiMedia eXtended*, machines cellulaires). Cette architecture permet le **parallélisme de données**.

MISD *Multiple Instruction stream, single Data stream* : modèle où plusieurs instructions sont exécutées sur un seul flux de données. Bien qu'aucune architecture ne corresponde à ce modèle [Bau01] (modèle théorique [CJ98]), les machines *pipeline* (travail à la chaîne [LS00]), superscalaires ou vectorielles sont classifiées dans ce modèle, car elles implantent le **parallélisme de flux**.

MIMD *Multiple Instruction stream, Multiple Data stream* : modèle où chaque processeur est libre d'exécuter les instructions qui lui sont propres sur un flux de données qui lui est propre (multiprocesseurs à mémoire partagée ou distribuée, §IV.1.2). Cette architecture permet tout type de parallélisme.

Dans notre contexte d'études où la segmentation d'images est intégrée dans un système complet de codage vidéo, les architectures SISD ne sont pas suffisamment puissantes pour supporter le traitement du flux des données à 25 images QCIF minimum par seconde en respectant les contraintes de consommation des terminaux mobiles (§IV.5.2.1 page 132).

Une architecture SIMD permet d'augmenter le débit des calculs, et une parallélisation efficace lorsque les traitements sont réguliers : chaque instruction est alimentée par des données utiles (données devant être mises à jour). Cependant, pour l'algorithme de segmentation considéré où le flux des données se propage des minima vers les maxima, le caractère des traitements est local et fortement dépendant de l'image d'entrée. Dans ce cadre, une implantation SIMD serait inefficace.

Le parallélisme de flux, réalisé par une architecture *pipeline*, est inadapté car les traitements au niveau pixel sont simples. Une architecture MISD est donc écartée.

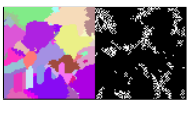
Nous nous sommes donc orientés vers la classe des machines MIMD.

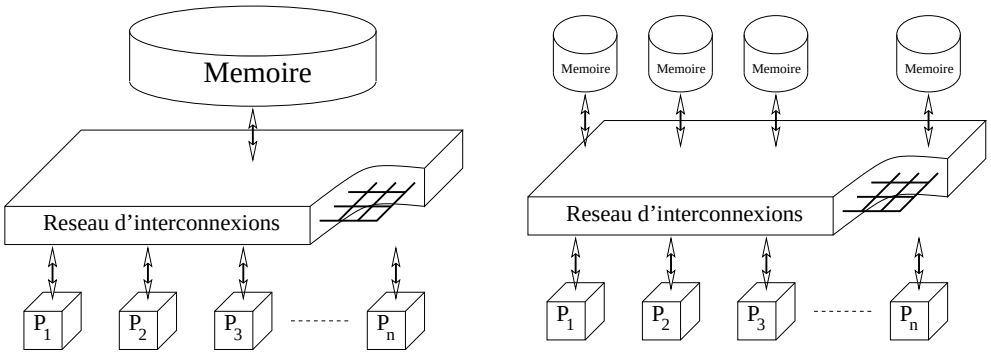
IV.1.2 Les architectures MIMD

Les architectures MIMD (systèmes distribués) ont bien évolué depuis l'établissement de cette taxinomie. Une classification plus fine [VdSD01] de ce modèle distingue deux groupes : les **réseaux fortement couplés** et les **réseaux faiblement couplés**. Le type d'interconnexion utilisé pour relier mémoires et processeurs, bus ou commutateur (*switch*), génère d'autres sous-divisions pour chacun de ces groupes [Car99].

IV.1.2.1 Réseau de processeurs fortement couplés

Ces architectures (*tightly coupled systems*) sont à espace d'adressage unique, c'est-à-dire que tous les processeurs accèdent à une mémoire physique **unique** (fi-





(a) à mémoire unique (b) à banc de mémoires

FIG. IV.1 – Réseau de processeurs fortement couplés (mémoire partagée).

gure IV.1a) ou à un banc de mémoires (figure IV.1b) via un réseau d'interconnexions. La mémoire est **partagée** par plusieurs ou tous les processeurs.

Les données circulent dans un réseau qui est généralement une matrice de commutation *Crossbar Switch* (n^2 commutateurs sont nécessaires pour connecter les n unités de calculs), ou un réseau maillé Ω [Tro94] (seulement $n.\log_2 n$ commutateurs disposés en étages sont utilisés, au prix de temps de transfert conséquents car les données doivent traverser plusieurs étages).

Ce type d'architecture est adapté à une granularité à gros grains (quelques processeurs complexes). Construire un système multiprocesseurs fortement couplé à mémoire partagée et constitué d'un grand nombre de processeurs est difficile et coûteux. Il n'est pas en adéquation avec les contraintes des terminaux portables.

Comme nous souhaitons une complexité et un coût d'implantation les plus faibles possibles, nous nous sommes dirigés vers les réseaux de processeurs faiblement couplés.

IV.1.2.2 Réseau de processeurs faiblement couplés

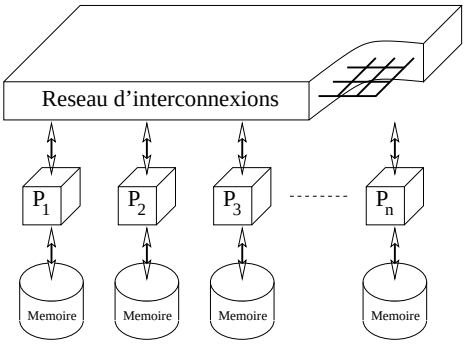


FIG. IV.2 – Réseau de processeurs faiblement couplés (mémoire distribuée).

Dans le groupe des **systèmes faiblement couplés** (*loosely coupled systems*), les nœuds qui définissent la machine parallèle sont indépendants les uns des autres. Chaque nœud est composé d'un processeur et d'une **mémoire locale** physiquement

distribuée (figure IV.2). La coopération des processeurs [Bau01] s'effectue grâce à des communications, des échanges de **messages**, car ces systèmes n'ont pas la capacité d'adressage globale : les mémoires sont **privées**, seul le processeur local peut y accéder.

La croissance de la complexité du réseau en fonction du nombre de processeurs est proportionnelle au degré[1] de chaque nœud (un processeur et sa mémoire privée) du réseau. Si le degré est faible, alors cette topologie est bien disposée au parallélisme massif (nombre important de processeurs). Ce type d'architectures est adapté à des traitements locaux car les temps de communication entre deux processeurs distants sont approximativement proportionnels à la distance qui les sépare, *i.e.* le nombre de processeurs intermédiaires à traverser. Si le degré de connectivité de chaque nœud est constant (resp. variable) au cours du temps, alors la topologie du réseau d'interconnexions est dite **statique** (resp. **dynamique**).

Le chapitre III présentait un algorithme local au niveau pixel construisant, à partir d'une grille symétrique, une forêt d'arborescences. Pour implanter sous forme parallèle l'algorithme de *Hill-Climbing* réordonnancé, nous préconisons un réseau de processeurs faiblement couplés car le nombre de pixels et le flux des données sont importants, les calculs sont simples et locaux, et le réseau est dynamique. De plus, nous verrons (§IV.2 page 80) que cette architecture permet d'envisager toute taille de grain, du plus fin (un processeur par pixel) au plus gros (un processeurs pour l'image). C'est pourquoi, nous approfondissons l'étude de cette architecture MIMD à mémoire distribuée.

Modèle SPMD *Single Program Multiple Data stream*. Ce modèle est un cas particulier du modèle MIMD où le groupe d'instructions est identique pour tous les processeurs, sans pour autant qu'ils exécutent les mêmes instructions au même moment.

Dans le cadre de l'algorithme de *Hill-Climbing*, nous choisissons ce modèle d'implantation parallèle car le comportement de tous les pixels est identique. Le programme implante la machine à trois états décrite par la figure III.2 page 37.

Les processeurs et la répartition de la mémoire du système à présent fixés, il reste à définir le réseau d'interconnexion entre les grains.

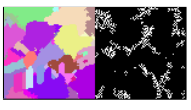
IV.1.3 La topologie du réseau de communication

Dans cette partie, nous présentons les principales topologies de réseau appliquées aux systèmes distribués [Dul96] [Rob97]. L'évaluation des points forts et points faibles de chacun d'eux nous permet de retenir (§IV.1.3.5) la topologie en forme de grille et de tore pour l'implantation de notre algorithme.

IV.1.3.1 Les grilles

Une grille (figure IV.3) est un réseau où le degré de connexité de chaque nœud est constant. Les grilles les plus usitées sont 4, 6 et 8 connexes (voir figures II.2 et II.3

[1] Nombre de "liens" en un nœud (définition A.15 page 179).



page 10).

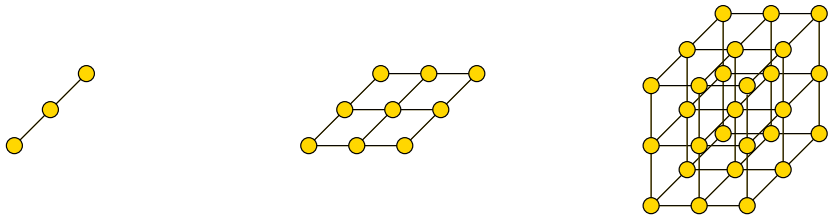


FIG. IV.3 – Topologie en grille.

Le principal avantage de la grille réside dans sa simplicité : le graphe 2D est planaire. La conception physique d'un tel réseau est simple car les liens de communication sont approximativement de même longueur et ne se croisent pas.

Cependant, le diamètre[1] du réseau est relativement élevé, impliquant un nombre de nœuds important à franchir lorsque deux processeurs éloignés veulent communiquer.

IV.1.3.2 Les tores

L'inconvénient des grilles est la particularisation des processeurs situés au bord du réseau : leur degré de connexité est plus faible. Si le voisinage de chaque processeur du bord est important, la topologie en forme de tore (figure IV.4) permet de compléter leur voisinage par rebouclage des canaux de communication.

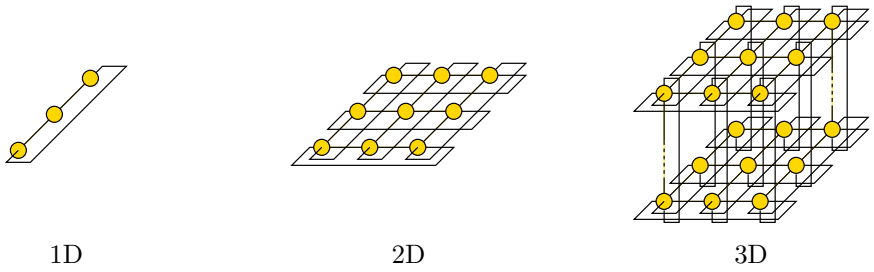


FIG. IV.4 – Topologie en tore.

Le principal avantage de cette topologie est que le degré de connexité est constant même pour les nœuds du bord. Le principal inconvénient est, comme pour la grille, son diamètre élevé. Nous verrons §IV.1.3.5 que la variante repliée de cette topologie permet d'homogénéiser la longueur des interconnexions.

IV.1.3.3 Les pyramides

Cette topologie est construite par superposition de grilles 2D (§IV.1.3.1) de taille décroissante (figure IV.5). Le principal avantage de cette topologie est la hiérarchisation des traitements et la centralisation d'informations, inférant un diamètre du graphe réduit. Cependant, cette topologie n'est pas planaire donc délicate à implanter physiquement. Cette topologie n'est pas retenue car la possibilité d'effectuer des traitements régionaux n'est pas exploitée.

[1]Distance de deux nœuds les plus éloignés (définition A.21 page 179).

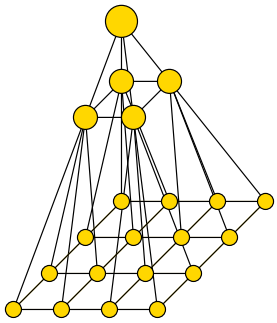


FIG. IV.5 – Topologie en pyramide.

IV.1.3.4 Les hypercubes

À partir de deux processeurs connectés, cette topologie est construite récursivement à partir d'hypercubes de dimensions inférieures (figure IV.6). Les hypercubes sont utilisés entre autres par les *Connection Machine* [Thi94].

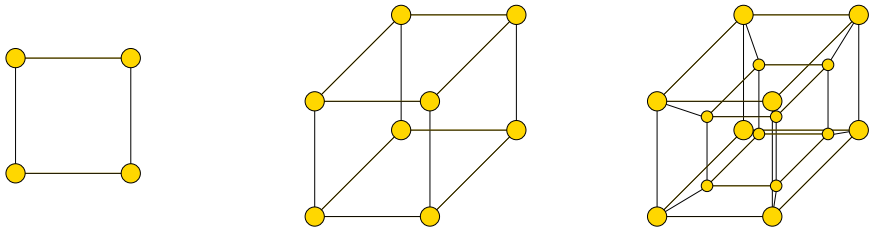


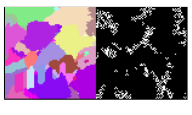
FIG. IV.6 – Topologie en hypercube.

Le principal avantage de cette topologie est sa régularité, ce qui permet d'homogénéiser les longueurs des canaux de communication entre sites (groupement de processeurs) de même dimension, et son faible diamètre (si n est la dimension de l'hypercube, alors une données traverse au maximum n routeurs pour atteindre le processeur cible [Fau95, ann.3]). Par contre, le degré de chaque sommet croît en fonction de la taille du réseau, d'où une topologie multi-dimensionnelle complexe à réaliser.

IV.1.3.5 Conclusion

Les topologies du réseau de communication les mieux adaptées à l'implantation parallèle de l'algorithme de *Hill-Climbing* réordonnancé sont la grille et le tore, car elles respectent la topologie de l'image (connexité des pixels), et exploitent la propriété de localité de l'algorithme.

L'algorithme de *Hill-Climbing* est basé sur la grille 2D pour modéliser les interactions entre pixels. Si la topologie du réseau d'interconnexion et de l'image sont identiques, alors le modèle d'exécution des processeurs est en adéquation avec le comportement des pixels (décrit par l'algorithme). C'est pourquoi, nous avons choisi une grille 2D comme topologie de l'architecture cible. Pour des raisons de faisabilité microélectronique que nous présenterons au chapitre V, une grille 2D 4-connexe est



choisie pour l'implantation parallèle à granularité la plus fine de cet algorithme.

Pour les granularités augmentées (§IV.2.3), la topologie en forme de tore est utilisée. Cependant par soucis de faisabilité microélectronique, l'intégration directe de ce schéma (figure IV.4) au niveau masque de fabrication (*layout*) ne respecte pas le principe de localité, puisque la longueur des liens de rebouclage augmente avec la taille du réseau. Afin de rétablir cette localité, c'est-à-dire obtenir des liens de taille minimale et indépendante de la taille du réseau, F. Robin [Rob97, chap.5] propose une topologie **torique repliée**.

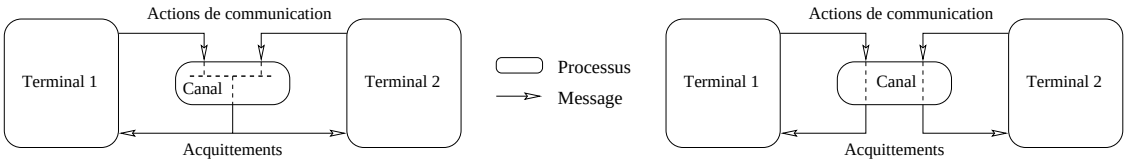
Afin de compléter la spécification du réseau d'interconnexion, étudions maintenant les différents modèles de communication.

IV.1.4 Les modèles de communication

Ce paragraphe présente les principaux modèles de communication par passage de messages, leurs avantages et inconvénients. La description détaillée des modèles de communication (par variable partagée, par méthode ou par événements), du point de vue général, va au-delà du contexte de ce mémoire. Une description plus complète peut être consultée dans [Fas01].

Les quatre qualificatifs couramment employés pour caractériser les communications par passage de messages sont **synchrone**, **asynchrone**, **bloquant** et **non-bloquant**. Ces termes sont parfois employés pour caractériser l'ordonnancement de deux processus communicants [Tro94] (synchrone si la mémoire du canal est nulle, asynchrone sinon, et non-bloquante si le canal acquitte aussitôt le processus émetteur). Cependant ils ne caractérisent pas la communication en elle-même, c'est pourquoi nous adoptons une définition radicalement différente.

Ici, seuls les termes «bloquant» et «non-bloquant» sont utilisés pour spécifier les communications. Nous n'utiliserons jamais les termes «synchrone» et «asynchrone» pour ce contexte.



(a) Communication bloquante. (b) Communication non-bloquante.

FIG. IV.7 – Caractérisation des communications entre un processus et son canal.

Ici, nous caractérisons l'ordonnancement des actions de communication (figure IV.7) entre **un** processus demandeur (demande d'écriture ou de lecture) et le canal. La formalisation de ces communications par des ordres d'entrée/sortie explicités en langage CHP (*Communicating Hardware Processes*, §V.3.2 page 153) est décrite dans [Rob97].

Le canal de communication est un processus au même titre que les processus effectuant des demandes de communication (les terminaux). Le canal peut-être com-

posé d'une mémoire de taille quelconque. **La spécification d'une communication s'effectue entre le demandeur et le canal.**

Définition IV.1 (Demande de communication). *Nous appelons «demande de communication», l'action d'un terminal sur le canal. Une demande de communication se décline en une demande de lecture, ou une demande d'écriture.*

Nous dirons qu'une demande de communication est «contentée» si et seulement si le canal est capable de transmettre une donnée valide, *i.e.* une donnée encore jamais transmise.

Définition IV.2 (Communication bloquante). *Une communication est dite bloquante si le canal acquitte une demande de communication seulement si elle peut être contentée.*

Cette communication fournit un mécanisme de transmission sans perte de données.

Par exemple (figure IV.7a), si le terminal 1 demande une donnée au canal (action de lecture) alors que ce dernier est vide (aucune donnée n'est présente), ce processus reste bloqué (le trait horizontal en pointillés symbolise une barrière de synchronisation entre les deux actions de communication) tant qu'aucune demande d'écriture de la part du terminal 2 n'est effectuée.

La mémoire d'un canal (FIFO) est un processus à part entière, il constitue une interface supplémentaire entre les deux terminaux. Il est chargé de mémoriser une donnée lors d'une demande d'écriture ou de fournir une donnée lors d'une demande de lecture. Quelque soit la taille de cette mémoire, nous dirons que la communication entre le terminal et le canal est bloquante car le processus lié à la FIFO arrête (par l'absence d'un message d'acquittement) le terminal demandeur si sa demande ne peut pas être contentée (canal vide si demande de lecture, ou canal saturé si demande d'écriture). Le seul effet de cette FIFO est de **désynchroniser les processus des terminaux**, elle n'a aucun effet sur la caractérisation de la communication. Si le canal ne possède pas de mémoire, cette communication force la synchronisation des actions de communication des deux terminaux.

Définition IV.3 (Communication non-bloquante). *Une communication est dite non-bloquante si le canal acquitte aussitôt une demande de communication même si elle ne peut pas être contentée.*

Par exemple (figure IV.7b), si le terminal demande une donnée au canal alors que ce dernier est vide, un signal d'acquittement est **aussitôt** retourné. L'acquittement retourné par le canal dépend de l'application et peut prendre différentes formes : retourner une copie de la dernière donnée envoyée, une valeur nulle, un message indiquant l'échec de l'action de communication, *etc.* C'est ce que nous appelons dans ce manuscrit **les stratégies de communication**, elles sous-entendent qu'une mémoire de taille au moins égale à un est utilisée. Par exemple, les stratégies liées à une demande d'écriture peuvent être :

- **conserver l'ancienne donnée :** si le canal est saturé, la dernière donnée écrite dans la FIFO ne peut pas être écrasée, et le canal retourne un message d'échec

au terminal demandeur. Le "destin" de la donnée la plus récente (celle qui fait l'objet de l'action de communication) est laissée à la charge du processus émetteur ;
- **conserver la nouvelle donnée :** si le canal est saturé, la dernière donnée écrite dans la FIFO est remplacée par la nouvelle donnée.

Si le canal est exempt de mémoire, une action de communication non-bloquante effectuée par un premier terminal ne peut-être contentée que si le second terminal effectue une action de communication duale bloquante (ici, le dual d'une lecture est une écriture).

Les communications non-bloquantes fournissent un mécanisme de transmission avec perte de données. Leur utilisation nécessite un algorithme robuste face à ce comportement.

Remarques:
- Les stratégies de communication liées à une demande de lecture ne sont pas décrites ici.

- Dans le cadre de l'implantation de l'algorithme de *Hill-Climbing*, nous verrons que des écritures non-bloquantes conservant la nouvelle donnée sont employées pour la granularité la plus fine (§IV.2.2), et des écritures non-bloquantes conservant l'ancienne donnée pour les granularités intermédiaires (§IV.2.3).

IV.1.5 Modèle de conception des processeurs

Afin de compléter la spécification de l'architecture, nous précisons la technologie, synchrone ou asynchrone, des processeurs élémentaires.

IV.1.5.1 Processeurs synchrones

Ces circuits VLSI (*Very Large Scale Integration*) sont cadencés par une horloge globale. Les unités combinatoires élémentaires le constituant sont séparées par des **registres** permettant d'échantillonner à intervalles de temps réguliers les résultats supposés stabilisés de la logique combinatoire.

Le principal avantage de ces circuits est qu'une logique combinatoire simple est utilisée, et que les outils de développement sont éprouvés.

Cependant, la synchronisation de tous les registres impose une approche au pire cas : la chaîne critique la plus longue parmi toutes les unités combinatoires fixe la fréquence maximum de l'horloge. De plus, les raies spectrales dues à l'arbre d'horloge peuvent bruiter le système de radiocommunication.

IV.1.5.2 Processeurs asynchrones

Ces circuits se distinguent par l'absence d'horloge [ND97]. Les unités combinatoires élémentaires se synchronisent localement grâce à l'échange de leurs données

via des communications bloquantes (définition IV.2 page 77) [Ren00] [Viv01].

L'interconnexion de modules élémentaires localement synchronisés forme un système où les modules de haut niveau sont désynchronisés. Les principaux avantages de ces circuits sont :

- **Un calcul au plus tôt :** le circuit étant localement synchronisé, dès qu'une unité a terminé un calcul, le résultat est aussitôt envoyé à ses unités voisines. De même, l'arrivée d'une nouvelle donnée aux interfaces d'une unité inactive déclenche aussitôt une phase de calcul ;
- **Un processeur inactif consomme quasiment rien :** par construction des unités, la logique est bloquée tant qu'aucune donnée n'est en entrée ;
- **La robustesse** face aux variations des éléments physiques (énergie électrique, température, paramètres de fabrication variables...) ;
- **Une bonne scalabilité technologique :** un circuit asynchrone s'adapte plus facilement aux contraintes imposées par l'évolution rapide des générations technologiques ;
- **La faible émission électromagnétique :** la désynchronisation globale des fronts des signaux implique un étalement du spectre des émissions ;
- **Le support des modèles de calcul asynchrones :** algorithmes asynchrones [Rob97].

Cependant, les fonctions combinatoires doivent être sans aléa, et toute donnée est accompagnée de sa signalisation, d'où une logique combinatoire plus complexe. Cette technologie étant relativement récente, les outils de conception n'ont pas la même maturité que les outils de conception synchrone.

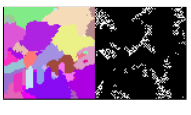
Ce style de conception microélectronique est en adéquation avec le contexte des études (terminaux portables : perspective basse consommation et faible bruit électromagnétique) et avec l'algorithme développé (localité des traitements et calcul au plus tôt). C'est pourquoi, nous orientons nos travaux vers la conception d'une architecture parallèle basée sur cette technologie.

IV.1.6 Conclusion

L'état de l'art sur les architectures parallèles nous a permis de déterminer le modèle d'architecture le mieux approprié au traitement d'images, et plus particulièrement à l'implantation parallèle de l'algorithme de *Hill-Climbing*. Cette étude oriente donc nos travaux vers une architecture SPMD (*Single Programm Multiple Data stream*) à mémoire distribuée composée de processeurs asynchrones. Les topologies du réseau d'interconnexion envisagées sont les grilles régulières et les tores qui permettent :

- d'implanter des traitements locaux de faible complexité nécessitant un flux important de données ;
- de respecter la topologie de l'image (connexité des pixels) et des traitements.

Lors de l'exploration des architectures (§IV.2), nous justifierons le choix d'un réseau d'interconnexion utilisant des communications non-bloquantes, où la topologie en grille 2D conserve la donnée la plus récente, et où la topologie torique 2D conserve les données les plus anciennes.



IV.2 Spécification et exploration des architectures

À partir des choix effectués sur le modèle d'architecture cible (§IV.1.6), nous présentons les caractéristiques de l'ensemble des architectures simulées. À partir de leurs spécificités communes (§IV.2.1), nous étudions, en perspective d'une faisabilité d'intégration, différentes architectures de granularité allant de la plus fine (§IV.2.2) à la plus grosse (§IV.2.3).

IV.2.1 Caractéristiques indépendantes de la granularité

Le modèle d'architecture choisi pour implanter l'algorithme de *Hill-Climbing* réordonnancé est de type SPMD, composé d'un réseau de processeurs asynchrones faiblement couplés.

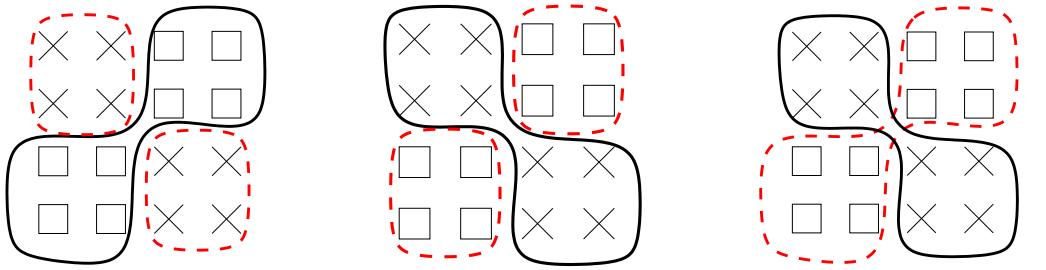
IV.2.1.1 Modèle SPMD

Le programme unique justifié par le comportement identique de tous les pixels dans l'algorithme réordonnancé, est entièrement spécifié par les calculs et transitions d'état de la machine à 3 états (figure III.2 page 37). Une description détaillé de ce programme est présentée à la section IV.5.1.3 page 119.

IV.2.1.2 Réseau d'interconnexion

Topologie régulière et choix du degré de connectivité. Comme les images d'entrée sont des images rectangulaires à deux dimensions, la topologie la mieux appropriée (§IV.1.6 page 79) est une grille régulière 2D (figure IV.3 page 74). Nous verrons (§IV.2.3) que la structure de tore est nécessaire pour les granularités intermédiaires.

Afin de simplifier l'implantation de l'algorithme, le degré de connectivité du réseau de processeurs est choisi identique à celui de l'image. Un circuit aussi simple que possible est souhaité, donc seul un voisinage réduit aux plus proches voisins est envisagé.



(a) Une région "□" et deux "×"

(b) Une région "×" et deux "□"

(c) Une région "□" et "×". Graphe non planaire !

FIG. IV.8 – Indétermination des régions sur une grille 8-connexe [Ser00].

Pour une grille régulière, les degrés de connectivité sont[1] 4, 6 ou 8. Comme le présente la figure IV.8, une image de dimension 4×4 composée de deux fois deux

[1]Pour des raisons d'homogénéité, seuls des voisinages symétriques sont envisagés.

blocs 2×2 de même étiquette, un voisinage 8-connexe implique une représentation non planaire des régions.

L'utilisation d'une grille 6-connexe (figure II.3 page 10) lève cette ambiguïté, cependant, construire un réseau où chaque processeur communique avec six voisins impliquerait un coût d'implantation des canaux de communication prohibitif. Nous confirmerons dans le chapitre V que ce point (le nombre de canaux interprocesseurs) est primordial pour l'implantation matérielle du réseau.

C'est pourquoi, nous avons choisi un réseau de processeurs sur une grille 4-connexe car il présente le meilleur rapport qualité de représentation des régions/complexité d'implantation, sachant que ce choix influe sur la partition finale de l'image.

Modèle de communication. Pour l'algorithme de *Hill-Climbing*, les calculs au niveau pixel sont fortement désynchronisés. Nous exploitons au mieux cette spécificité en utilisant des communications non-bloquantes en écriture (définition IV.3 page 77).

Par ailleurs, comme démontré dans la section III.3.3 page 45, le comportement atemporelle de l'algorithme segmente correctement les images. Le résultat de segmentation est donc faiblement dépendant (excepté pour les pixels pivots et intérieurs) de l'ordonnancement des calculs. Cette propriété est donc exploitée au profit de l'optimisation des coûts de communication et de synchronisation.

Les bus de communication sont unidirectionnels. Ce type de bus est, dans une première approche, facile à implanter car les interfaces de communication sont de complexité réduite. De plus, ceci permet à deux processeurs voisins d'émettre ou lire simultanément des données de façon totalement désynchronisée.

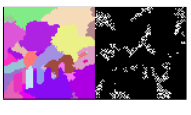
IV.2.1.3 Les unités de traitement

Processeur asynchrone. L'indépendance des traitements permet de relâcher les contraintes de synchronisation entre les processeurs. Les circuits asynchrones présentent de nombreux avantages (§IV.1.2.2 page 72) [Viv01, chap.1] [Ren00] [ND97]. C'est pourquoi nous mettons l'asynchronisme matériel au profit de l'asynchronisme algorithmique en utilisant un réseau de processeurs pilotés par les données (*Data Driven*).

La mémoire. Privée à chaque processeur, elle mémorise, entre autres, les luminances, les étiquettes et la topographie de l'image ($\mathcal{N}^I(p)$, $\mathcal{N}^=(p)\ldots$).

Modèle d'exécution. Si un processeur possède toutes les données nécessaires issues de ses voisins, alors il les traite sinon il devient inactif en attente d'informations complémentaires.

Afin de limiter les inondations multiples, et donc le nombre de communications, chaque processeur donne **la priorité aux lectures sur les écritures** : pour un même processus, aucun résultat d'un calcul n'est écrit vers des processus voisins tant qu'au moins une donnée est disponible en entrée [Rob97]. Le temps de convergence est réduit d'environ 10% à 20% car de nombreuses données transitoires ne sont pas traitées par les processeurs et propagées dans le réseau.



IV.2.1.4 Conclusion

L'exploration du spectre des architectures dédiées à l'algorithme de *Hill-Climbing* réordonnancé converge vers un modèle de parallélisme SPMD (*Single Program Multiple Data stream*), où le réseau d'interconnexion est une grille 2D 4-connexe. Chaque processeur possède une mémoire privée et communique avec ses voisins nord, est, sud et ouest via des bus de communication unidirectionnels où des communications non-bloquantes sont utilisées. L'ensemble des processeurs sont asynchrones : ils sont pilotés par le flux des données.

IV.2.2 Architecture parallèle à granularité la plus fine

Dans le cadre de l'algorithme de segmentation massivement parallèle, nous avons donné la priorité à la simplicité et à la rapidité du circuit. Nous avons choisi d'implanter une grille planaire régulière de processeurs élémentaires asynchrones (grille 2D, figure IV.3) communiquant à l'aide de bus unidirectionnels. Le réseau est faiblement couplé et chaque processeur est associé à un pixel.

IV.2.2.1 Modèle d'exécution

Un processeur devient actif dès l'arrivée d'une donnée sur au moins un de ses ports d'entrée. Tant que des données sont disponibles, ce dernier les consomment et met à jour ses variables internes. Si le canal d'entrée est vide à l'issu de ces traitements, alors les dernières données mise à jour sont écrites sur tous les canaux de sortie.

Afin de garantir la propagation de l'étiquette du minimum d'attraction sur l'ensemble de la région, chaque processeur assure la propagation de la dernière donnée.

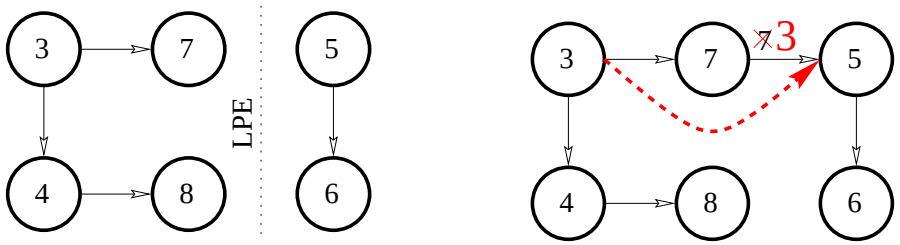
Le processeur devient alors inactif dès que toute lecture est impossible, et que toutes les écritures ont réussi.

IV.2.2.2 Modèle de communication

En perspective d'une diminution de l'activité des processeurs, nous avons choisi des communications non-bloquantes en lecture et en écriture via un canal ayant **une mémoire de taille un**. En effet, si la taille mémoire des canaux est nulle, alors tout processeur souhaitant émettre une donnée doit rester actif tant que les communications avec tous ses voisins ne sont pas contentées (assurance de la propagation de la dernière donnée, §IV.2.2.1). La taille des mémoires doit donc être non nulle.

Par ailleurs, la donnée la plus récente est la donnée la plus pertinente : utiliser des mémoires de tailles supérieures à un présenterait un coût d'implantation supplémentaire, et ralentirait l'architecture car un nombre plus important de données serait traité.

La particularité de cette architecture est que **le canal conserve la donnée la plus récente** (écrase la plus ancienne). Toutefois, si une telle stratégie est adoptée durant la phase d'initialisation, alors la topographie de l'image peut-être légèrement altérée (figure IV.9).



(a) Initialisation correcte. L'altitude est indiquée au centre de chaque processeur (pixel).

(b) Initialisation incorrecte, l'étiquette a franchi la ligne de crête.

FIG. IV.9 – Influence des communications non-bloquantes avec conservation de la donnée la plus récente sur le partitionnement final lors de la phase d'initialisation.

Par exemple, une donnée d'inondation d'un voisin rapide (pixel d'altitude 7) peut-être considérée comme une donnée d'initialisation pour un nœud plus lent (pixel d'altitude 5). Ceci provoque une distorsion partielle du relief pouvant provoquer un déplacement de la ligne de partage des eaux, voire même sa disparition.

Par contre, pour tout processeur initialisé (phase de relaxation), la perte d'une étiquette transitoire n'a aucune conséquence fonctionnelle sur le partitionnement final puisque seule la dernière (étiquette minimale du minimum régional) prime.

Au niveau de chaque canal, le meilleur compromis rapidité/justesse est obtenu en conservant la donnée la plus ancienne durant la phase d'initialisation, et en conservant la nouvelle donnée lors de la phase de relaxation. Cependant, ceci implique un coût de mémorisation supplémentaire : chaque processeur doit mémoriser sa donnée la plus récente afin de garantir son écriture dès la donnée d'initialisation consommée, sinon, un bassin d'attraction peut-être subdivisé.

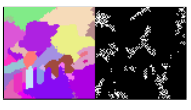
IV.2.3 Architecture parallèle à granularité augmentée

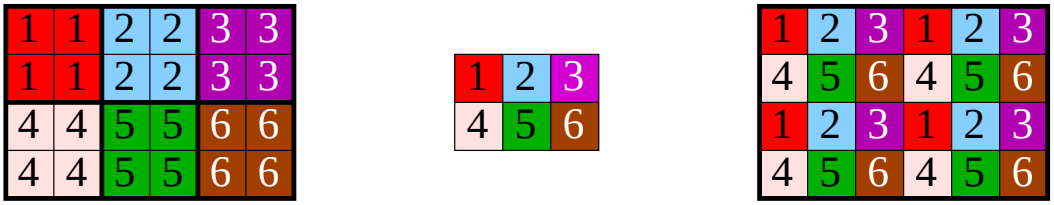
Une architecture à degré de parallélisme limité, *i.e.* à granularité intermédiaire, se justifie par l'apport du compromis entre optimisation des capacités de calculs, la surface d'implantation microélectronique, et les temps de traitement.

IV.2.3.1 Types de partitionnement des données et topologie du réseau

Les deux types de partitionnement classiques pour un parallélisme de données (§IV.1) sont les partitionnements **LSGP** (Localement Séquentiel Globalement Parallèle) (figure IV.10a) et **LPGS** (Localement Parallèle Globalement Séquentiel) (figure IV.10c) [Rob97].

Le partitionnement LSGP. Parfois désigné par 2D-rectilinéaire [Lau98], affecte à chaque processeur une région **compacte** de l'image : un domaine. Une zone de recouvrement des domaines peut-être utilisée afin d'accroître l'efficacité du système [Mog97], au prix d'une duplication des données.





(a) Partitionnement LSGP : chaque processeur est associé à un bloc **compact** 2×2 de données

(b) Réseau de 3 × 2 processeurs. Chacun d'eux possède 4 pixels

(c) Partitionnement LPGS : chaque processeur est associé à un bloc **distribué** 2×2 de données

FIG. IV.10 – Types de partitionnement LSGP et LPGS. Chaque processeur est identifié par une couleur (et numéro).

Les traitements locaux, internes à un même processeur, sont rapides. Cependant, chaque processeur calcule sur une **région** de l'image. Nous verrons lors de la validation de l'architecture (§IV.4), que les traitements de l'algorithme de *Hill-Climbing* sont rapidement très localisés. Opter pour un partitionnement LSGP impliquerait, pour l'implantation parallèle de cet algorithme, un déséquilibre des charges important : très peu de processeurs contribueraient à la convergence des calculs.

Le partitionnement LPGS. Les données, associées à un même processeur, sont équidistribuées sur la totalité du réseau.

Il est nécessaire d'utiliser une topologie **torique** 2D (figure IV.4 page 74) afin que les processeurs localisés sur le bord du réseau puissent communiquer avec leurs quatre voisins. Par exemple sur la figure IV.10c, un pixel du processeur 3 doit pouvoir envoyer une donnée à son pixel voisin est qui est contenu dans le processeur 1.

L'avantage de ce partitionnement est qu'il permet d'exploiter au mieux l'asynchronisme algorithmique et architectural car deux pixels voisins dans l'image (figure IV.10c) appartiennent à deux processeurs voisins dans le réseau (figure IV.10b).

Cependant, si la charge de calcul est répartie sur l'ensemble des données, le nombre des communications est plus important comparativement à un partitionnement LSGP.

L'étude d'opérateurs morphologiques désordonnancés montre de meilleurs performances pour un partitionnement LPGS par rapport à un partitionnement LSGP [Rob97] car l'asynchronisme algorithmique et architectural est mieux exploité. Comme notre algorithme de segmentation présente des caractéristiques de localité et de désordonnancement similaires, le partitionnement choisi pour l'étude de granularité intermédiaire est le partitionnement LPGS.

IV.2.3.2 Modèle d'exécution

Cette partie présente les particularités du comportement des processeurs impliquées par le partitionnement LPGS.

Adressage des pixels. Contrairement à la granularité la plus fine, le partitionnement LPGS nécessite l'envoie des coordonnées des pixels en plus des données à

transmettre. Un processeur souhaitant propager les données d'un de ses pixels vers son voisin, fournit l'adresse du pixel cible appartenant au processeur voisin. Ainsi, le pixel voisin récepteur (pour l'algorithme) peut être déterminé par le processeur voisin (pour l'architecture).

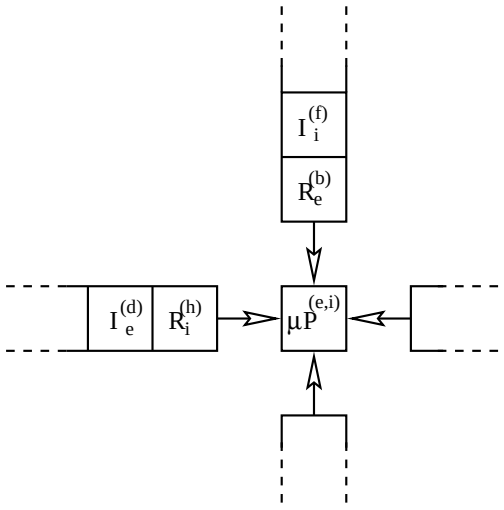
Ce surcoût est compensé par la meilleure répartition des charges entre processeurs, d'autant plus si les traitements sont irréguliers et localisés [Rob97].

Pour des raisons d'optimisation du coût d'implantation, l'adresse du pixel cible est déterminée pour chaque émission, c'est-à-dire que si l'écriture de la donnée dans le canal a échoué, l'adresse précédemment calculée n'est pas mémorisée. Ceci permet de réduire le nombre de points mémoire, au détriment des performances du processeur (tout échec d'écriture implique un surcoût de calcul).

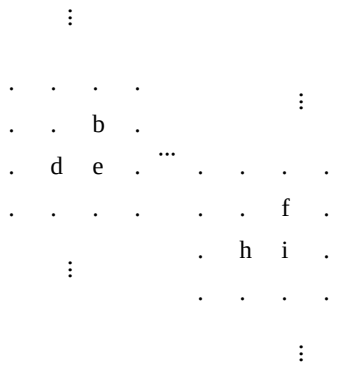
Synchronisation des machines à état. Dans cette partie, nous montrons que le comportement des pixels sans contrainte de synchronisation intraprocesseur interbloque les processeurs.

Dans le cas d'une granularité augmentée, l'ensemble des pixels contenus dans un processeur communiquent avec leurs pixels voisins via un **unique canal de communication** pour chaque direction : les huit canaux d'entrée/sortie du processeur sont partagés. Contrairement à la granularité la plus fine étudiée précédemment, chaque pixel ne dispose pas de canaux privés.

Comme nous le justifierons à la section IV.2.3.3, les communications non-bloquantes utilisées conserve les données les plus anciennes. Si le modèle d'exécution des pixels (§IV.2.2.1) n'est pas modifié, l'utilisation de cette stratégie de communication implique un interblocage des processeurs.



(a) Processeur associé aux pixels e et i bloqué : les données de relaxation $R_e^{(b)}$ et $R_i^{(h)}$ issues des processeurs associés aux pixels h et b bloquent les FIFO, car les pixels e et i ont déjà reçu auparavant une donnée d'initialisation $I_i^{(h)}$ et $I_e^{(b)}$



(b) Les pixels d'intérêt de l'image : e et i appartiennent au même processeur $\mu P^{(e,i)}$

Fig. IV.11 – Interblocage des canaux de communication.

La machine originelle de l'algorithme de *Hill-Climbing*, associée à un pixel, est bloquée à l'état INIT tant qu'une lecture sur **tous** les pixels voisins n'est pas effectuée. La phase d'initialisation une fois terminée, une donnée de relaxation (c'est-à-dire d'inondation ou d'unification) est aussitôt émise vers tous les voisins de ce pixel.

Dans le cas d'une granularité augmentée, un processeur intègre autant de machines à état qu'il doit traiter de pixels. Si pour deux pixels e et i (figure IV.11) en cours d'initialisation, une donnée de relaxation $R_e^{(b)}$ destinée à e bloque une donnée d'initialisation $I_i^{(f)}$ destinée à i, et de même sur un autre canal du processeur (l'ordre des pixels étant inversé), alors ces deux canaux restent infiniment bloqués car aucune des deux machines de e ou de i ne consomment les données $R_e^{(b)}$ et $R_i^{(h)}$ situées en tête des FIFO.

Cet interblocage est résolu en donnant la priorité aux données d'initialisation (elles seront obligatoirement consommées au plus tôt puisqu'elles sont attendues par les pixels) sur les données de relaxation. Chaque processeur impose un point de synchronisation supplémentaire pour tous ses pixels : le passage de l'état INIT à l'état MP ou NM ne s'effectue plus au niveau pixel comme précédemment, mais au niveau de tous les pixels d'un processeur. Une données de relaxation est émise par un processeur si et seulement si tous ses pixels ont réussi à émettre leurs données d'initialisation (luminance et étiquette originales) vers tous leurs pixels (=processeurs) voisins.

IV.2.3.3 Modèle de communication

Les communications utilisées sont non-bloquantes en lecture et en écriture, la stratégie adoptée en écriture conserve les données anciennes. En effet pour un même processeur, les données destinées à un pixel ne doivent pas écraser les données précédemment envoyées destinées à un autre pixel.

Par exemple, supposons que deux pixels u_α et v_α d'un processeur α soient les voisins suivant la même direction de deux pixels u_β et v_β d'un autre processeur β. Soit u_α un minimum local, si l'émission d'une donnée de v_α vers v_β écrase une donnée d'initialisation de u_α vers u_β, alors u_β reste infiniment bloqué puisqu'il ne peut s'initialiser : u_α étant un minimum local, il ne transmet qu'une donnée vers ses voisins. Donc, **pour garantir la propagation des données sur la totalité de sa région, nous conservons les données les plus anciennes.**

Comme la stratégie adoptée implique un accroissement du coût des communications (chaque propagation d'une donnée d'un pixel implique une communication entre processeurs voisins), un compromis entre relâchement des synchronisations des processeurs et coût d'implantation doit être déterminé. Plus la mémoire des canaux est étendue, plus la probabilité d'échec d'une écriture est petite, mais plus le coût d'implantation microélectronique est élevé.

Nous verrons dans la section IV.5.1.4 page 125, une estimation quantitative de ce compromis entre profondeur des canaux et échec des émissions.

IV.3 L'environnement de simulation

Grâce à la bibliothèque de prototypage matériel SystemC™ (§IV.3.1), un environnement de simulation est établi (§IV.3.2). Nous présentons la méthode utilisée pour simuler (§IV.3.3) une architecture exempte d'horloge (§IV.3.4). À partir d'éléments définis par cette bibliothèque, nous indiquons comment implanter les différentes communications (§IV.3.5).

IV.3.1 La bibliothèque SystemC™

Pour modéliser le réseau de processeurs asynchrones, nous avons développé une méthodologie de simulation et de validation s'appuyant sur un langage de haut niveau utilisé dans l'industrie. La bibliothèque SystemC™ [Sys] est un environnement de simulation et de prototypage développé en C++. Elle est composée des éléments de base essentiels à la description des systèmes matériels qui sont, principalement :

- des processus indépendants et leur liste de sensibilité ;
- des ports d'entrée/sortie ;
- des canaux de communication ;
- une horloge.

L'ensemble est piloté par l'ordonnanceur (*scheduler*) qui assure le mouvement des signaux dans le modèle et l'activation et exécution des processus, le tout en émulant le parallélisme intrinsèque d'un circuit matériel.

Le langage de haut niveau d'abstraction facilite le prototypage de systèmes car :

- son caractère générique permet de paramétrer facilement le modèle, d'où une expression modulaire des prototypes développés ;
- il permet des simulations rapides, comparativement aux modèles de simulation de bas niveaux développés en HDL (*Hardware Description Language*) ;
- la vérification fonctionnelle peut s'effectuer à un haut niveau d'information (fichiers image, de trace...) et non pas au niveau signal logique.

Dans le cadre de nos études, la recherche amont d'une architecture capable d'implanter efficacement l'algorithme de *Hill-Climbing* implique prioritairement une description de haut niveau d'abstraction plutôt qu'une description synthétisable du modèle. C'est pourquoi, la **généricité des architectures** ainsi modélisées est le point **fondamental** qui a orienté nos travaux vers ce style de prototypage. Par exemple, nous verrons au cours des sections suivantes que l'exploration d'une gamme étendue d'architectures à degré de granularité intermédiaire est simplement obtenue par le paramétrage des dimensions de la grille, de la taille des mémoires tampon et de la largeur des données en fonction de la taille de l'image d'entrée. Un seul modèle générique permet la simulation d'une vaste gamme d'architectures par un simple passage en ligne de paramètres d'entrée.

IV.3.2 Structure de l'environnement de simulation

Dans cette partie, l'environnement de simulation est vue dans son intégralité. Une vision plus fine des unités de traitement et de communication est présentée dans

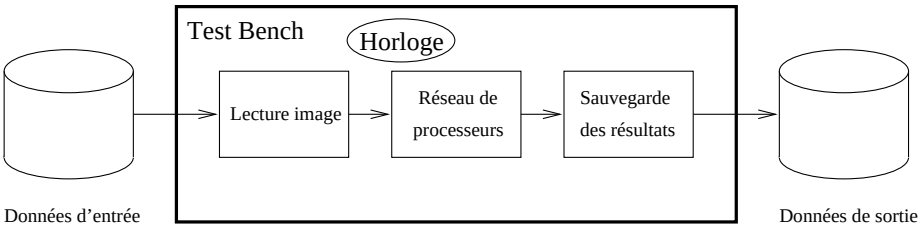


FIG. IV.12 – Architecture globale du système de simulation.

les sections IV.3.4 et IV.3.5.

L'ensemble des éléments du système de simulation (figure IV.12) sont les suivants.

L'architecture étudiée : Le réseau de processeurs est constitué de mémoires (variables locales), de processeurs (méthodes et procédures présentées §IV.3.4) et du réseau d'interconnexion (§IV.3.5).

Le banc de test (*test bench*) : Il fournit les *stimuli* (un fichier d'entrée par exemple) de l'architecture, et mémorise dans un fichier de sortie les signaux ou variables internes.

Les modules fonctionnels : (lecture image et sauvegarde des résultats) Ils interfacent le banc de test et les fichiers d'entrée/sortie. Destinés uniquement à visualiser le comportement de l'architecture modélisée, ils n'ont aucune influence sur son comportement.

L'horloge : Cet élément global de l'architecture est indispensable au fonctionnement de l'environnement de simulation SystemC™. Il est chargé, dans la conception de circuits synchrones, de synchroniser tous les registres (bascules, *latch*...) du circuit. Cependant, comment s'abstraire de cette contrainte pour concevoir un **circuit asynchrone** qui en est exempt ?

Nous verrons, au paragraphe IV.3.4, comment émuler le comportement d'un circuit asynchrone à partir d'un environnement adapté à la conception de circuits synchrones.

La simulation du système se déroule en quatre phases :

1. Instanciation de l'architecture (§IV.3.3).
2. Lancement de la simulation et initialisation.
3. Détection de fin.
4. Sauvegarde des mesures et destruction des modules.

L'architecture une fois créée, une commande lance l'ordonnanceur : tous les signaux s'initialisent et l'horloge effectue sa première transition. Comme il est impossible de déterminer *a priori* la date de fin des calculs, cette particularité des circuits asynchrones est implantée grâce à un signal dédié chargé de stimuler un module fonctionnel qui stoppe l'ordonnanceur. Enfin, les mesures globales sont sauvegardées juste avant la destruction de l'architecture modélisée.

À présent, étudions plus en détails la modélisation du réseau de processeurs, de son comportement et de ses communications.

IV.3.3 Instanciation de l'architecture

Durant la phase de construction de l'architecture, l'environnement de simulation crée les processeurs et leur mémoire privée (méthodes et variables locales), et les canaux de communication. La topologie du réseau étant une grille ou un tore, ces éléments forment des matrices de dimensions respectant un paramètre d'entrée externe au programme.

La connexion de tous ces canaux avec les ports de communication des processeurs finalise l'instanciation du réseau d'interconnexion : la topologie du système est complètement déterminée.

IV.3.4 Modèle d'exécution

Dans cette partie, nous présentons le comportement de chaque processeur au sein de l'environnement de simulation. Comme précisé précédemment, SystemC™ impose l'existence d'une horloge dans l'environnement. Nous présentons ici deux modèles d'exécution envisagés pour répondre à cet impératif :

1. Un module horloge inutile pour un modèle entièrement piloté par les données.
2. Un modèle cadencé par l'horloge, et fonctionnellement asynchrone.

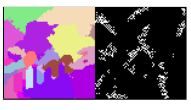
IV.3.4.1 Modèle piloté par les données

Un tel réseau asynchrone est caractérisé par l'absence d'horloge. Comme la bibliothèque SystemC™ est conçue pour le prototypage de systèmes synchrones, la présence d'une horloge globale est indispensable au fonctionnement de l'ordonnanceur. Nous avons donc intégré cette horloge dans un module fonctionnel (module en haut à gauche de la figure IV.13), sans interaction avec l'architecture modélisée. Ce module sert uniquement de base de temps pour le *scheduler*, et n'est jamais utilisé pour cadencer les unités de traitement.

Le principal avantage de ce modèle est sa capacité à reproduire fidèlement le comportement d'un circuit asynchrone. Il permet de simuler l'asynchronisme à un haut niveau d'abstraction. La figure IV.13 présente le modèle utilisé pour l'instanciation d'un pixel et de ses canaux de communication. Les communications bloquantes ou non-bloquantes sont explicitement modélisées (signaux *inDta/outDta*, *inAck/outAck*). Les éléments internes constituant le processeur (carrés noirs) sont stimulés par des signaux internes (signaux[1] *e_ReceivedD*, *e_Completed*...) indépendants de l'horloge du simulateur : le système est entièrement piloté par le flux des données (*Data driven*). Chacun de ces éléments constitue un processus parallèle pour le *scheduler*.

Le développement et l'utilisation de ce modèle a mis en évidence la nécessite d'une quantité importante de ressources logicielles : plus la granularité est fine, plus le modèle est complexe (plus de 63 000 tâches parallèles doivent être émulées pour une grille de 88×72 processeurs). La simulation d'un tel réseau nécessite plus de deux gigaoctets de mémoire vive. Pour des réseaux de tailles inférieures, le point de convergence est atteint au bout de plusieurs heures de simulation, et ce, avec une machine SUN UltraSparc bi-processeurs. L'intérêt de la simulation à un haut niveau

[1] Les petits rectangles en pointillés indiquent une latence de propagation non nulle.



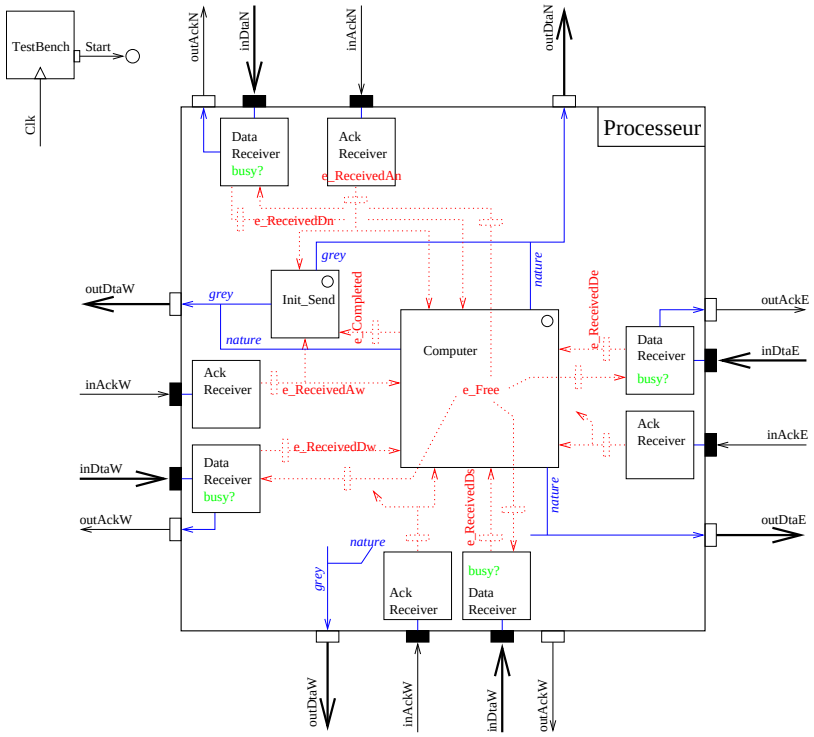


FIG. IV.13 – Synoptique du processeur 4-connexe (*data driven*).

d'abstraction est alors mis en défaut.

Ce modèle a cependant été utilisé au cours d'une préétude de l'algorithme de *Hill-Climbing* réordonnancé. Les résultats d'un algorithme de recherche de minima (corollaire aux algorithmes IV.2 et IV.3 page 112) implanté au niveau de granularité d'un processeur par pixel a montré la pertinence d'un tel modèle.

Afin de pouvoir simuler l'algorithme de segmentation complet sur des images de dimension QCIF (176 × 144 pixels) et d'explorer les architectures possibles, il est nécessaire de réduire la mémoire utilisée par le simulateur, ainsi que les temps de simulation. Nous nous sommes donc orientés vers un modèle de simulation plus simple, plus abstrait (*i.e.* moins prêt du *hardware*), et moins gourmand.

IV.3.4.2 Modèle fonctionnellement asynchrone

Cette partie présente la méthode adoptée pour réduire le nombre de processus parallèles tout en conservant un modèle d'exécution fonctionnellement asynchrone. Seule la granularité la plus fine est considérée ici, le modèle de simulation étant équivalent pour les granularités augmentées.

Comme les traitements sont locaux, l'idée est d'attribuer un unique fil d'exécution (*scheduler thread*) à chaque processeur. La complexité algorithmique est alors réduite d'un facteur dix environ.

La figure IV.14 présente la synoptique du processeur élémentaire : une machine à état fini composée d'un module purement algorithmique et d'un module cadencé. Ce dernier fournit en entrée du module combinatoire les résultats des calculs obtenus à

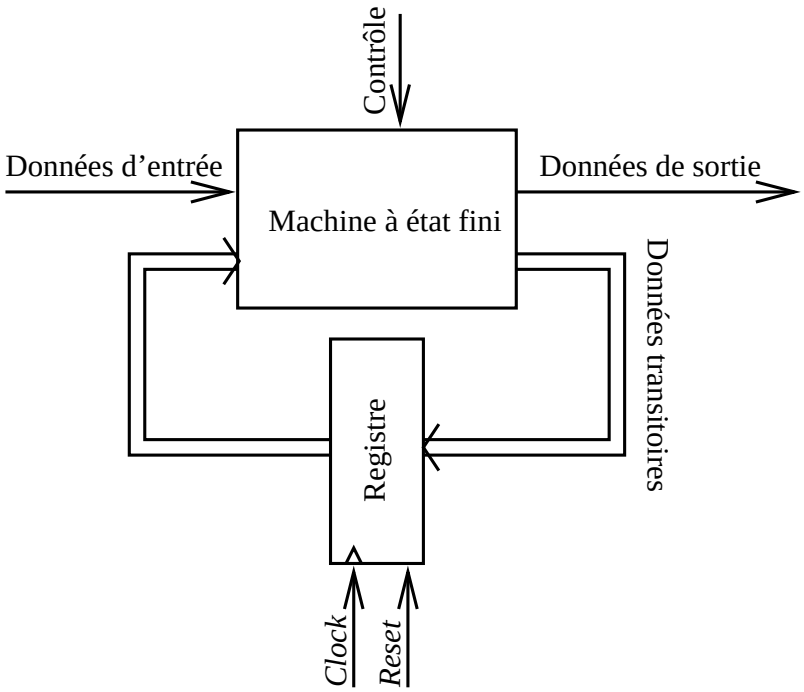


FIG. IV.14 – Présentation synoptique d'un processeur fonctionnellement asynchrone.

"l'étape" précédente.

L'espace temps du modèle. La notion "d'étape" fait référence ici à une date : un front d'horloge ascendant. L'intervalle de temps séparant deux étapes successives représente un *quantum* de temps indivisible entre deux états différents du réseau : c'est la résolution temporelle du modèle de simulation. Comme tous les processeurs sont immergés dans le même espace de temps (le temps est supposé incompressible), l'horloge les relie tous : elle est globale au système de **simulation**.

L'avantage de cette modélisation du comportement du processeur est qu'elle ne nécessite qu'un seul *thread* par processeur sensible uniquement à l'horloge. Cependant, elle impose que tout calcul intermédiaire s'effectue en une "étape" (*quantum* de temps). Sans modification de ce modèle, il est impossible de modéliser des latences variables suivant la complexité des calculs.

Afin d'intégrer des temps de traitement, la latence de chaque calcul ou communication est émulée par une mise en attente des fonctions combinatoire durant plusieurs cycles de l'horloge. Les estimations de ces latences sont issues des mesures de performances du processeur ASPRO [Viv01]. Étant de l'ordre de $7ns$ en moyenne, nous estimons que la latence d'une action est comprise entre 6 et $8ns$. Afin d'émuler l'indéterminisme des temps de calculs, la latence de chaque action est obtenue par tirage aléatoire équiprobable. Une modélisation plus fidèle de l'asynchronisme consisterait à utiliser une loi de Rayleigh [Pla94, chap.4]. Cependant, l'estimation de la latence d'une action étant déjà approximative, il ne nous a pas semblé utile d'alourdir le moteur de simulation en utilisant une telle loi. Dans le même ordre d'idées, afin de simplifier le processus d'écoulement du temps, la résolution est fixée

à une nanoseconde : la fréquence de l'horloge est donc fixée à 1 GigaHertz (période d'une nanoseconde).

Remarques:

- Il est important de noter que **cette horloge est fictive** et ne correspond à aucune caractéristique technologique. Son rôle est uniquement d'intégrer le temps dans le modèle (rôle de chronomètre), et non pas d'échantillonner les signaux entre blocs combinatoires d'un circuit (rôle d'ordonnancement).
- Les contraintes technologiques sont intégrées dans l'estimation des chaînes critiques des opérateurs, c'est-à-dire la mise en attente des fonctions combinatoires durant **un certain nombre de cycles** de cette horloge. La simulation est fonctionnelle car cette mise en attente n'est pas fonction des données à traiter, contrairement au fonctionnement matériel.

L'implantation de l'algorithme de *Hill-Climbing*. Ce paragraphe décrit la méthode utilisée pour implanter l'algorithme de segmentation dans le modèle présenté par la figure IV.14.

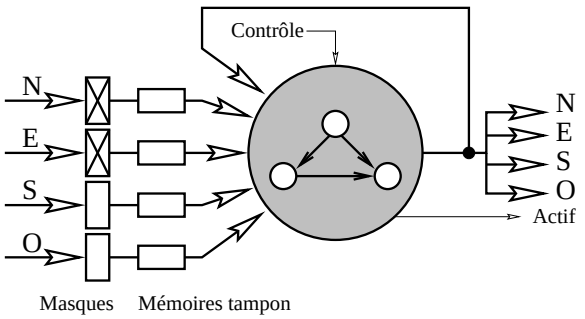


FIG. IV.15 – Processeur élémentaire doté de ses ports d'entrée masquables et son unité de traitement effectuant l'algorithme de *Hill-Climbing*.

La figure IV.15 est une présentation synoptique corollaire à celle de la figure IV.14, où les interfaces de communication interprocesseurs sont plus détaillées (cas où la grille est 4-connexe). Le cœur du processeur[1] (cercle gris contenant la machine à trois états) effectue cycliquement et séquentiellement une lecture, des calculs (une description précise du comportement est présentée par les algorithmes IV.7 à IV.10 page 119) et une écriture vers ses voisins.

Afin de modifier dynamiquement le degré de connectivité des nœuds du réseau, chaque processeur gère une variable de quatre bits indiquant si tel ou tel voisin doit être écouté (inhibition des ports nord et est sur la figure IV.15). Dans la section III.3.1 page 39, nous avons désigné cette variable par $net_G[v]$, où v correspond au processeur courant.

La détection de fin. Bien que le temps de calcul soit borné (l'algorithme ne boucle pas ni ne crée d'interblocages), il reste *a priori* inconnu. Il dépend en effet des données et des chemins de convergence de l'algorithme (construction aléatoire

[1]Unité combinatoire et registre

de la forêt d'arborescence). Ces derniers étant inconnus et dépendants de l'image, il est nécessaire de détecter le point de convergence puisqu'il ne peut être postulé[1]. La solution la plus simple et la plus efficace connue à ce jour est de combiner, par un OU logique, l'ensemble des états d'activité des processeurs [Dul96].

Un pixel est actif s'il reçoit de nouvelles données et les consomme. Il passe à l'état inactif, *i.e.* il s'endort, dès qu'il a fini l'envoi de ses résultats vers ses voisins.

L'utilisation de communications non-bloquantes entraînent potentiellement de fausses détection de fin. Sans aucune hypothèse sur les latences de transport des données entre deux pixels voisins, il est nécessaire d'établir un état d'activité sur les canaux de communication [Rob97]. Dans ce cas, le réseau est inactif si tous les processeurs sont inactifs **et si** tous les canaux de communication ne transportent pas de données.

Cette méthode accroît la complexité de l'architecture pour la simple détection de fin. Afin de la minimiser, nous supposerons que les latences de communications sont bornées, et qu'un filtre de *glitch* (filtre passe-bas sur le signal d'activité globale du réseau) est suffisant.

IV.3.5 Modèle de communication non-bloquante

Cette partie présente les éléments de la bibliothèque SystemC™ utilisés pour implanter des communications non-bloquantes. Seules ces communications sont détaillées ici puisqu'elles sont exclusivement utilisées par toutes les architectures étudiées (§IV.2 page 80).

Si la mémoire du canal est de taille unitaire, l'entité *signal* définie dans SystemC™ modélisant l'état logique d'un ou plusieurs fils est utilisée.

Si une mémoire de taille supérieure est souhaitée, alors des FIFO sont utilisées. L'écriture d'une donnée dans une FIFO étant bloquante par défaut (SystemC™ bloque le processus émetteur jusqu'à ce qu'une place dans le canal se soit libérée), **chaque processeur** élémentaire **sonde** la présence d'une donnée (resp. d'une place libre) lorsqu'il souhaite lire (resp. écrire) une donnée.

La bibliothèque SystemC™ n'autorise pas l'écriture simultanée d'une même valeur sur plusieurs FIFO : un cycle d'horloge est imposé pour chaque écriture. Étant en contradiction avec le modèle de communication souhaité, nous avons légèrement modifié cette bibliothèque afin que l'écriture simultanée d'une même donnée sur plusieurs canaux soit possible.

IV.3.6 Conclusion

Le caractère asynchrone du réseau ainsi que les communications non-bloquantes sont simulés grâce à un environnement de haut niveau d'abstraction : SystemC™.

Le modèle d'exécution et de communication de chaque processeur est émulé par une machine à état fonctionnellement asynchrone, cadencée par l'horloge du *scheduler* dont les temps de latence sont choisis aléatoirement. Cette alternative respecte les limitations de notre station de travail et nous permet de simuler le flux des données

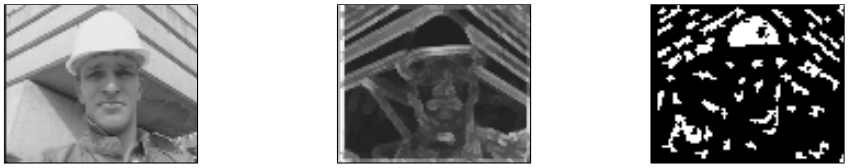
[1] Une majoration du chemin critique de propagation (par une spirale par exemple) et des latences de communication ne serait pas efficace.

dans un réseau de taille QCIF. L'environnement est alors suffisamment économe, en termes de ressources mémoire (environ 1 Go) et temps de simulation (environ une heure pour une image QCIF), pour reproduire les principales caractéristiques des architectures choisies (§IV.2).

IV.4 Validation des architectures

Les architectures désormais déterminées (§IV.1 et §IV.2) et l'environnement de simulation établi (§IV.3), cette partie est consacrée à la validation de ces architectures par la simulation. Les mesures qualitatives des résultats de segmentation montrent la pertinence de ces architectures. Les mesures quantitatives de complexité des modèles seront ensuite présentées dans la section IV.5.

Toutes les simulations présentées dans le reste de cette partie sont réalisées sur trois images gradient de dimensions standards pour les terminaux visiophoniques et simplifiées[1] pour des raisons de visibilité. Ainsi, la surface des bassins d'attraction est suffisamment grande pour observer la propagation des données dans le réseau.



(a) Image originale (b) Image gradient (c) Minima

FIG. IV.16 – Image de test : gradient simplifiée de *Foreman* SQCIF (88×72).

(a) Image originale (b) Image gradient (c) Minima

FIG. IV.17 – Image de test : gradient simplifiée de *Foreman* QCIF (176×144).

Les deux premières sont les images *Foreman* aux formats SQCIF (88×72 pixels, figure IV.16)[2] et QCIF (176×144 pixels, figure IV.17), et la troisième est *Susie* au format QCIF (figure IV.18).

Les images gradient sont ici éclaircies afin de mieux visualiser les zones de transitions à détecter.

Pour des raisons de reconnaissance des régions (SQCIF) et de visibilité (QCIF), les deux images *Foreman* n'ont pas subies le même niveau de simplification. Pour

[1]Utilisation de l'algorithme des cascades [Beu94] : premier niveau de hiérarchie pour l'image SQCIF et deuxième niveau de hiérarchie pour les images QCIF.

[2]Pour des raisons de visibilité, les images SQCIF sont légèrement agrandies par rapport aux images QCIF : les proportions ne sont pas respectées.

(a) Image originale (b) Image gradient (c) Minima

FIG. IV.18 – Image de test : gradient simplifiée de *Susie* QCIF (176×144).

l'image SQCIF, un seul niveau de hiérarchie de l'algorithme des cascades est utilisé, alors que pour l'image QCIF, deux niveaux de simplifications sont utilisés. L'image gradient SQCIF (figures IV.16b-c) comporte un nombre plus important de minima par rapport à l'image gradient QCIF (figures IV.17b-c). Un nombre plus important de régions pour l'image SQCIF est donc attendu.

IV.4.1 Influence de la répartition initiale des étiquettes

Dans cette partie, nous montrons l'effet de l'étiquetage initial sur le partitionnement final : des étiquettes ordonnées impliquent un résultat de segmentation différent (mais correspondant toutefois au même syntagme, §II.3.1 page 15). Par hypothèse, l'image d'entrée n'est pas semi-complète inférieurement (des pixels intérieurs non-minima existent) et nous considérons ici la granularité la plus fine de l'architecture (un processeur par pixel).

(a) Image synthétique 37×37 composée de quatre minima locaux situés au milieu de chaque bord et d'un unique plateau non-minimum

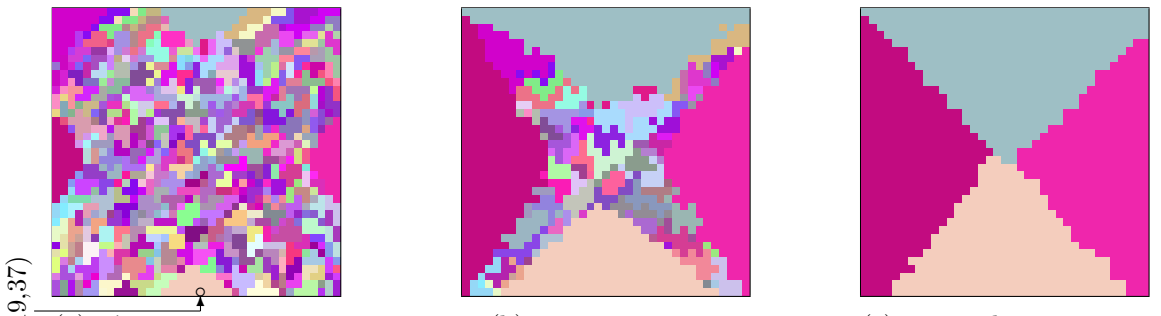
(b) LPE théoriques

FIG. IV.19 – Image test utilisée pour exhiber les effets de l'étiquetage initial.

Pour illustrer ceci, nous utilisons une image artificielle (figure IV.19) composée de quatre minima locaux situés au milieu de chaque côté du carré, et d'un plateau non-minimum. Théoriquement, les régions sont des triangles isocèles de base la largeur du carré (figure IV.19b). Les lignes de partage des eaux forment les diagonales du carré.

Afin d'étudier l'influence de l'étiquetage initial sur les partitions finales d'une image, deux répartitions sont expérimentées :

1. L'étiquette initiale correspond à l'adresse du pixel suivant un balayage direct vidéo (*raster scan*, figure IV.20).
2. L'étiquette initiale est attribuée de façon aléatoire (figure IV.22).



(a) étiquettes pour $t = t_1$ (b) pour $t = t_2 > t_1$ (c) point de convergence

FIG. IV.20 – Evolution des étiquettes au cours du temps pour une répartition initiale ordonnée.

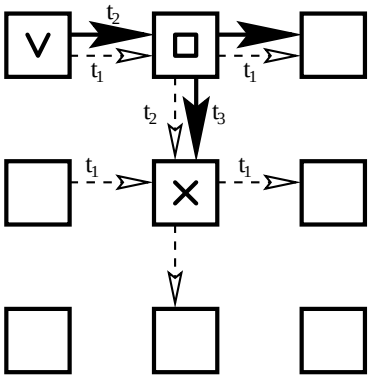
Afin d'expliciter le comportement du réseau, observons deux types de flux : le **flux d'unification** (un pixel situé à l'intérieur du plateau met à jour son étiquette, voir définition III.11 page 36) et le **flux d'inondation** (un pixel non-minimum transite de l'état MP à l'état NM, voir définition III.10 page 36). Le premier, de complexité élevée (lectures sur plusieurs ports et recherche de minima), est dépendant de l'étiquetage initial, alors que le second, de complexité réduite (lecture unique et simple copie), est dépendant du relief de l'image.

Grâce à la simulation de l'image de la figure IV.19, nous observons que plus les flux d'unification et d'inondation sont de directions similaires, plus la vitesse de propagation des étiquettes des minima est rapide. Un corollaire à cette remarque est : plus les directions sont similaires, plus la charge de calcul des processeurs en ces lieux est faible (allégement des traitements d'unification).

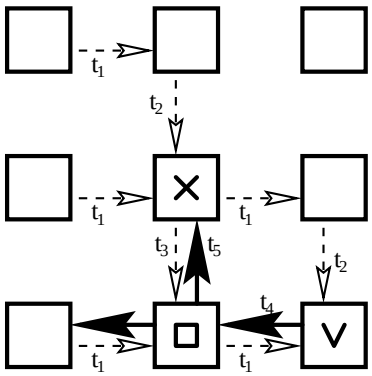
Suivant sa position dans le plateau, un processeur ajuste plus ou moins souvent son étiquette, d'où une charge de calcul variable.

Par exemple sur la figure IV.20, les étiquettes sont ordonnées suivant un balayage direct vidéo (de la gauche vers la droite et du haut vers le bas), donc l'étiquette la plus petite est dans le coin en haut à gauche, et la plus grande dans le coin diagonalement opposé. Le flux d'unification (propagation de l'étiquette minimale) s'effectue suivant la direction sud-est (figure IV.21) puisque pour tout pixel $v_\times$ du plateau, ses deux voisins ouest et nord sont d'étiquettes strictement inférieures ; $v_\times$ unifie son étiquette uniquement en lisant leurs données. La région sud (figure IV.20) issue du minimum local de coordonnées (19,37)[1] provoque un flux d'inondation de direction nord-est et nord-ouest (figure IV.21b). L'ensemble des pixels de cette région subissent deux flux (inondation et unification) de directions opposées : chaque pixel $v_\square$ (encore à l'état MP) situé sur un front d'inondation (un de ses voisins $v_\vee$ vient juste d'être inondé) unifie son étiquette jusqu'à deux fois **avant** d'être inondé, d'où une charge de calcul supérieure par rapport aux pixels situés aux endroits où les deux flux sont de même direction. L'étiquette du minimum local (19,37) (figure IV.20)

[1] Au milieu du bord sud de l'image.



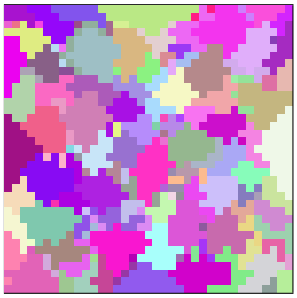
(a) Les flux sont de même direction. Les processeurs $v_{\vee}$ et $v_{\square}$ effectuent peu d'unifications avant de propager une donnée d'inondation à $v_{\times}$



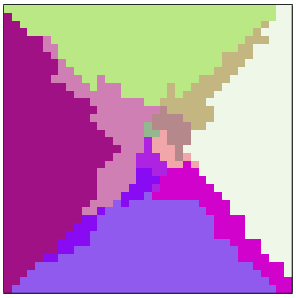
(b) Les flux sont de directions opposées. Plusieurs données d'unification (plus longues à traiter) chargent les ports des pixels : les données d'inondation se présentent aux interfaces des processeurs alors que ceux-ci sont en train de calculer des minima

FIG. IV.21 – Influence des flux d'unification (flèches en pointillées) et d'inondation (flèches pleines) sur les temps de traitements d'un processeur. Les pixels $v_{\vee}$, $v_{\square}$ et $v_{\times}$ sont respectivement un pixel inondé, un pixel sur le front d'inondation et un pixel en phase d'unification juste avant l'inondation.

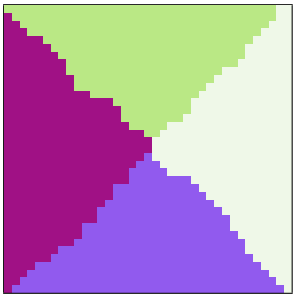
se diffuse donc plus lentement dans le réseau, impliquant un positionnement non symétrique des LPE.



(a) Étiquettes pour $t = t_1$



(b) pour $t = t_2 > t_1$



(c) point de convergence

FIG. IV.22 – Evolution des étiquettes au cours du temps pour une répartition initiale aléatoire. La propagation des données sur un plateau non-minimum est plus homogène.

Afin de rendre le flux d'unification indépendant de l'étiquetage initial, une étiquette aléatoire unique est attribuée à chaque processeur. On remarque une propagation plus uniforme et homogène —formation d'îlots dans la figure IV.22a— car tous les sites contribuent équitablement à la convergence du réseau. Les lignes de partage des eaux sont, dans ce cas, mieux positionnées (figure IV.22c).

Si le réseau est homogène en termes de capacité de calcul et de communication, nous pouvons affirmer que les fronts d'inondation se propagent sur un plateau

non-minimum, en moyenne, à vitesse constante suivant toutes les directions. Une répartition aléatoire des étiquettes est recommandée si des processeurs asynchrones sont utilisés.

Dans la suite du document, et en particulier au cours des sections IV.4.2 et IV.4.3, nous présentons uniquement les résultats de segmentation issus d'un étiquetage initial aléatoire.

IV.4.2 Partitionnement à fine granularité

Cette partie présente les résultats de différentes simulations du flux des données dans un réseau de processeurs asynchrones pour une granularité la plus fine. À l'état initial, chaque pixel possède son niveau de gris (image gradient à segmenter) et une étiquette unique aléatoire.

IV.4.2.1 Simulation du flux des données

Les figures IV.23 et IV.24 présentent l'évolution des étiquettes et les processeurs actifs contribuant à la convergence du réseau lors de la segmentation de l'image *Susie* QCIF. Chaque image des étiquettes (référencées par une couleur distincte) et des processeurs actifs (représentés par des pixels rouges) est séparée par un intervalle de temps régulier (ces intervalles ont été estimés par l'analyse de vitesse, §IV.5.2 page 131, à 700 ns environ).

Nous pouvons observer sur la figure IV.23 que la plupart des régions sont déterminées bien avant le point de convergence : dès la sixième image sur vingt, les segments obtenus sont pratiquement déterminés, à l'exception du fond qui est, à cet instant, encore divisé en cinq régions. Ceci est d'autant plus vrai qu'il existe de grandes différences entre les dimensions des régions.

Au niveau de l'activité du réseau, la figure IV.24 montre qu'un grand nombre de processeurs est actif en début de traitement, puis illustre que l'activité se concentre aux bords de régions n'ayant pas encore convergés. Une analyse quantitative de l'activité du réseau, présentée à la section IV.5.1 (figure IV.35a page 126), détaille ce comportement.

IV.4.2.2 Validation des résultats de segmentation

Afin d'apprécier plus aisément la cohérence de la segmentation, la figure IV.25 présente l'image originale (éclaircie afin de visualiser plus facilement les LPE) superposée avec les lignes de partage des eaux. Ces dernières étant d'épaisseur nulle, nous avons suréchantillonné l'image d'un rapport trois horizontalement et verticalement.

Comme mentionné précédemment, pour des raisons de visibilité, les LPE présentées dans la figure IV.25 sont issues d'images simplifiées, alors qu'elles ont été superposées sur cette figure aux images originales, ce qui explique la mauvaise position de certaines d'entre elles.

FIG. IV.23 – Evolution des étiquettes (*Susie* QCIF).

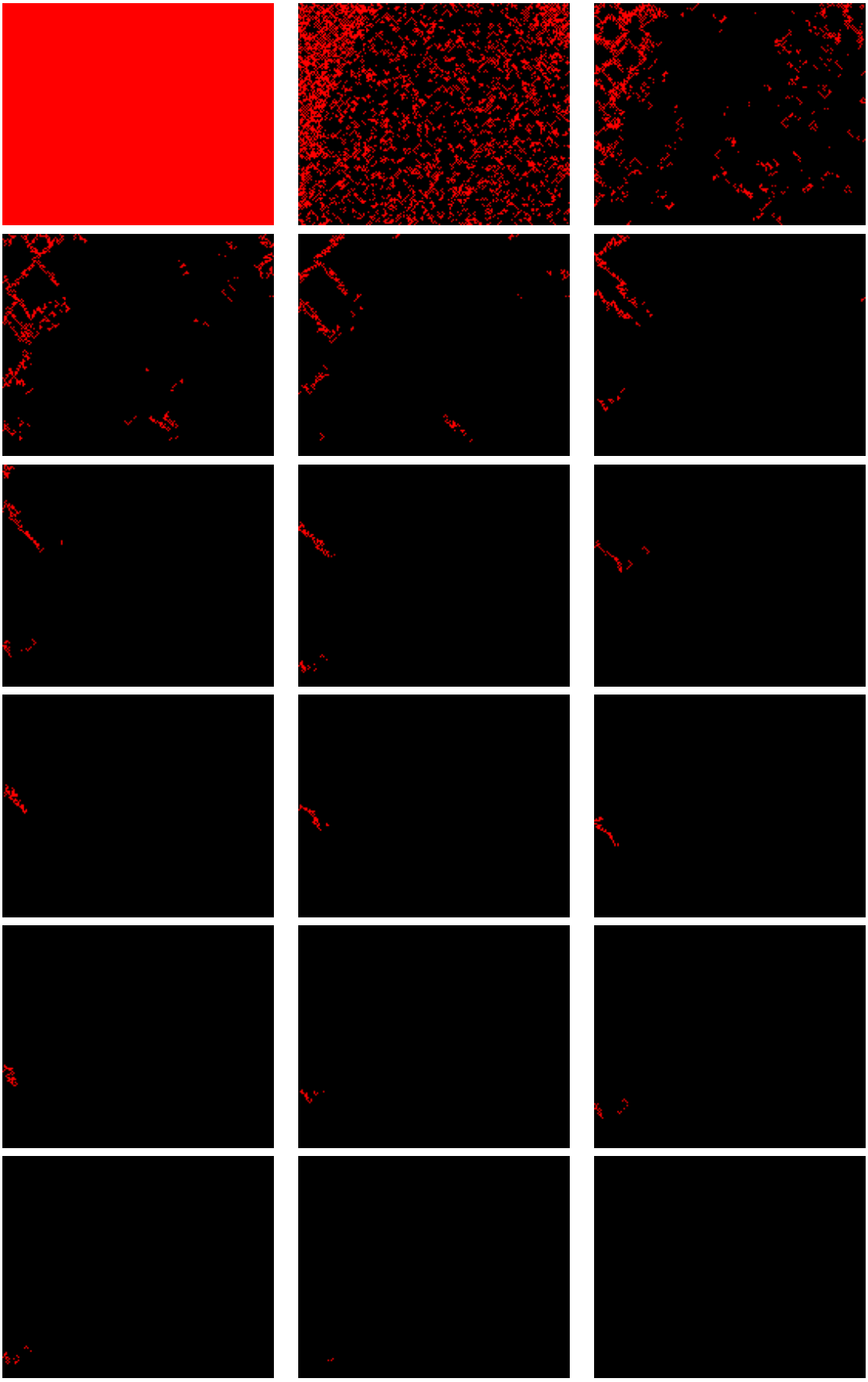
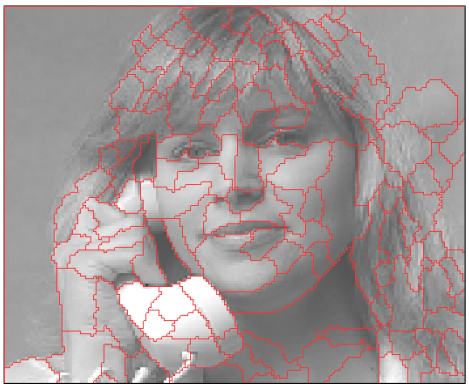


FIG. IV.24 – Processeurs actifs (*Susie* QCIF).



(a) *Susie* QCIF

(b) *Foreman* SQCIF

(c) *Foreman* QCIF

FIG. IV.25 – Image originale et ses lignes de partage des eaux déterminées par l'algorithme par ascension de colline réordonnancé.

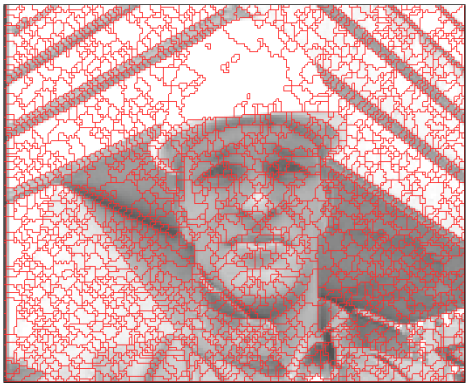
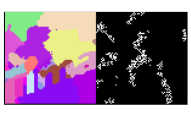


FIG. IV.26 – Segments obtenus sans simplification de l'image gradient. L'algorithme de *Hill-Climbing* désynchronisé place correctement les LPE.

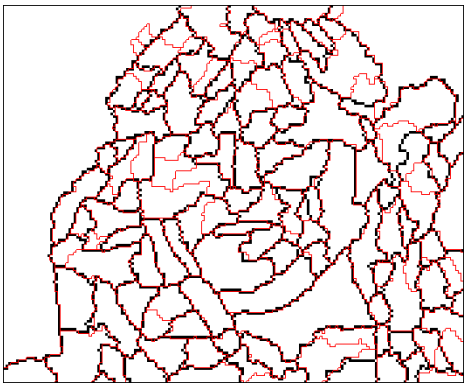


Par contre, la figure IV.26 présente le résultat de notre algorithme sur l'image gradient non simplifiée de *Foreman*, illustre la sursegmentation attendue, et surtout **le placement correct de la LPE**.

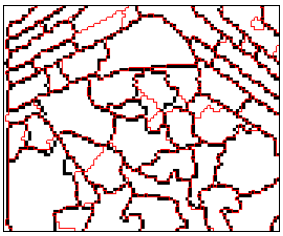
Afin de montrer la justesse de la segmentation, les partitions obtenues avec différents algorithmes classiques de segmentation sont comparées avec celles de notre algorithme.

Le premier algorithme considéré est la détermination des squelettes par zone d'influence sur une grille 6-connexe. Cet algorithme est considéré comme l'algorithme le moins biaisé. Le second est l'algorithme par inondation uniforme par file d'attente hiérarchique (§II.4.2.1 page 27).

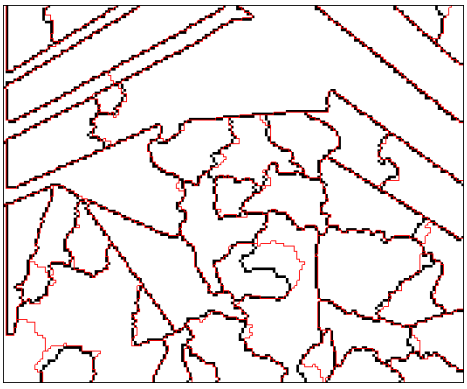
Contrairement à l'algorithme par inondation ou par ascension de colline, l'algorithme des squelettes par zone d'influence d'une image luminance détermine des pixels LPE : la ligne est d'épaisseur non-nulle. Afin de comparer les résultats obtenus par cette technique et celle proposée, l'image contenant les squelettes est suréchantillonnée d'un facteur trois horizontalement et verticalement, et les LPE d'épaisseur nulle (algorithme *Hill-Climbing*) lui sont superposées. Ainsi, la LPE d'épaisseur nulle est considérée comme "exacte" si elle est située sur le squelette.



(a) *Susie* QCIF



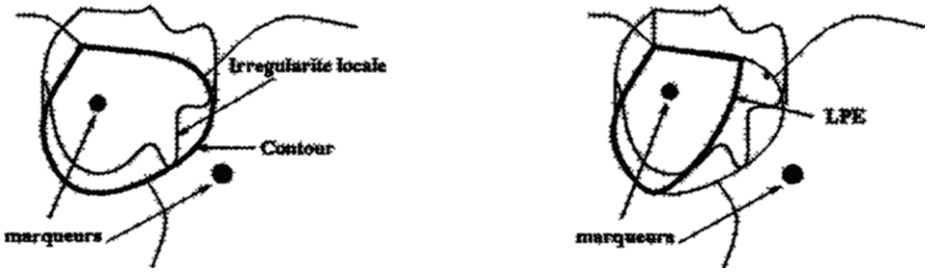
(b) *Foreman* SQCIF



(c) *Foreman* QCIF

FIG. IV.27 – Superposition des squelettes par zone d'influence (en noir) avec les LPE d'épaisseur nulle de l'algorithme parallèle à fine granularité (en rouge).

La figure IV.27 présente la comparaison entre les squelettes par zone d'influence et l'algorithme proposé. Nous pouvons observer certaines différences sur les trois images, la plus importante étant dans la figure IV.27c au niveau de la bouche du personnage. Cet artéfact bien connu [Vac95] est occasionné par l'utilisation d'un voisinage différent pour l'algorithme par SKIZ (6-connexe) et l'algorithme parallèle (4-connexe). Lors de l'utilisation d'un voisinage 6-connexe, le gradient présente une irrégularité locale (figure IV.28) qui n'existe pas pour un voisinage 4-connexe.



(a) Contour attendu (b) Contour obtenu

FIG. IV.28 – Influence de la qualité du gradient sur la segmentation : une irrégularité locale peut modifier notablement la position de la LPE (extrait de [Vac95]).

Nous observons sur la figure IV.27, pour une même image gradient, des régions supplémentaires générées par notre algorithme. En effet, il est également connu que la segmentation en 4-connexité crée toujours plus de régions qu'une segmentation en 6-connexité, car deux minima en 4-connexité peuvent "fusionner" en 6-connexité à travers les diagonales (figure II.3 page 10). C'est pourquoi, pour une même image gradient, nous observons sur la figure IV.27 des régions supplémentaires générées par notre algorithme.

La figure IV.29 présente la comparaison entre les LPE déterminées par l'algorithme séquentiel (inondation uniforme) et l'algorithme parallèle proposé. Les lignes noires correspondent aux lignes déterminées par l'algorithme séquentiel. Les écarts de partitions sont dus aux pixels pivots (*cf* la boutonnière, figure II.13 page 24) et aux plateaux non-minima.

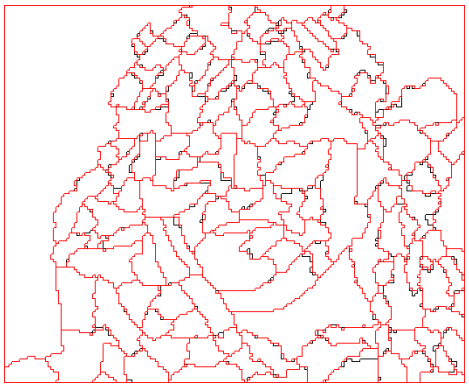
IV.4.2.3 Conclusion

Grâce à l'environnement de simulation SystemC^{TM}, nous avons vérifié que l'implantation à granularité la plus fine de l'algorithme de segmentation par ascension de colline se comporte bien tel que défini par sa formalisation (§III.3 page 39). La segmentation obtenue est cohérente pour les images naturelles testées.

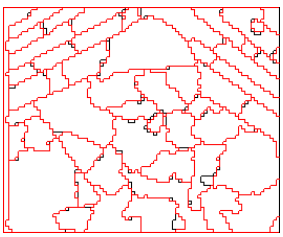
La simulation de l'activité du réseau illustre les zones d'activité du réseau. Nous étudierons ce point plus en détails lors de l'analyse des résultats (§IV.5.1.3 page 119).

IV.4.3 Partitionnement à granularité intermédiaire

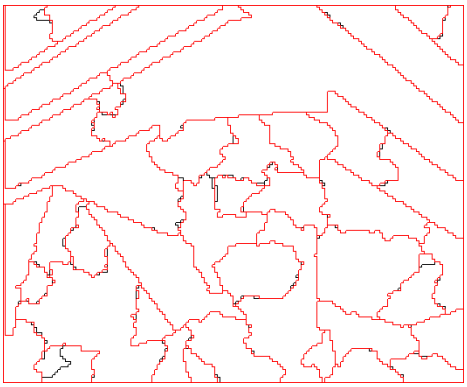
Dans cette partie, nous présentons les résultats de simulation (propagation des étiquettes et activité des processeurs) d'un réseau de 8×8 processeurs pour l'image *Foreman* QCIF (figure IV.17 page 94).



(a) *Susie* QCIF



(b) *Foreman* SQCIF



(c) *Foreman* QCIF

FIG. IV.29 – Comparaison des LPE d'épaisseur nulle entre l'algorithme séquentiel (en noir) et parallèle à fine granularité (en rouge).

IV.4.3.1 Simulation du flux des données

La figure IV.30 présente l'évolution des étiquettes pour un réseau de processeurs (partitionnement LPGS, §IV.2.3.1 page 83) de dimension 8×8 où la taille des FIFO est de quatre. Les images sont séparées par un intervalle de temps régulier estimé à 371 μs environ par l'analyse de vitesse (§IV.5.1.4 page 125).

Afin de déterminer quel pixel doit être mis à jour parmi le bloc alloué, chaque processeur possède un pointeur qui lui est propre. Arbitrairement, nous avons choisi de faire évoluer ce pointeur suivant un balayage direct vidéo.

Durant l'animation du flux des étiquettes, nous observons la caractéristique localement parallèle globalement séquentiel du partitionnement : à chaque intervalle de temps, on visualise une mise à jour terminée pour un bloc de l'image de la taille du réseau de processeurs, et cette mise à jour se déplace au début de la simulation en accord avec le déplacement du pointeur. En effet, au début du processus de segmentation, tous les pixels souhaitent une mise à jour, donc tous les processeurs se comportent de façon similaire. Comme les processeurs sont asynchrones et que la charge de calcul n'est pas identique pour tous, les propagations initialement locales s'étendent et rapidement nous observons des mises à jour réparties sur l'ensemble de l'image. Ceci s'explique par la faible proportion de pixels encore en activité par processeur, et par l'évolution indépendante de chaque processeur dans le réseau.

Bien sûr, le choix de faire évoluer ce pointeur de pixel suivant un balayage direct vidéo influence le partitionnement final puisque, localement, les premiers pixels traités durant la phase d'initialisation[1] seront mis à jour suivant des directions privilégiées. Afin de respecter un comportement plus homogène, il faudrait déplacer ce pointeur suivant un itinéraire aléatoire au sein de chaque processeur. Ce point n'a cependant pas été abordé et fait partie des perspectives.

La figure IV.31 présente l'activité du réseau, où son état est présenté pour une fréquence trois fois plus élevée que celle utilisée pour la figure IV.30. Quasiment tout les processeurs sont actifs jusqu'à la fin de la segmentation : les ressources du réseau sont mieux exploitées.

IV.4.3.2 Validation des résultats de segmentation

La figure IV.32 compare les lignes de partage des eaux obtenues avec un partitionnement à fine granularité et celles obtenues avec un partitionnement de 8×8 processeurs. Seules de faibles variations sont observées : environ un à deux pixels d'écart. Ces variations sont dues à l'indétermination du chemin de convergence. Par ailleurs, la région située en bas à gauche présente une différence due à la présence d'un pixel pivot. Cependant, nous remarquons que l'algorithme séquentiel par inondation (figure IV.29 page 104) et le partitionnement à granularité intermédiaire segmentent de façon similaire cette région.

IV.4.3.3 Conclusion

La propagation de l'étiquette des minima vers tous les pixels non-minima montre que les problèmes d'interblocage liés à ce partitionnement sont bien résolus. De plus,

[1] Détermination de la topographie locale.

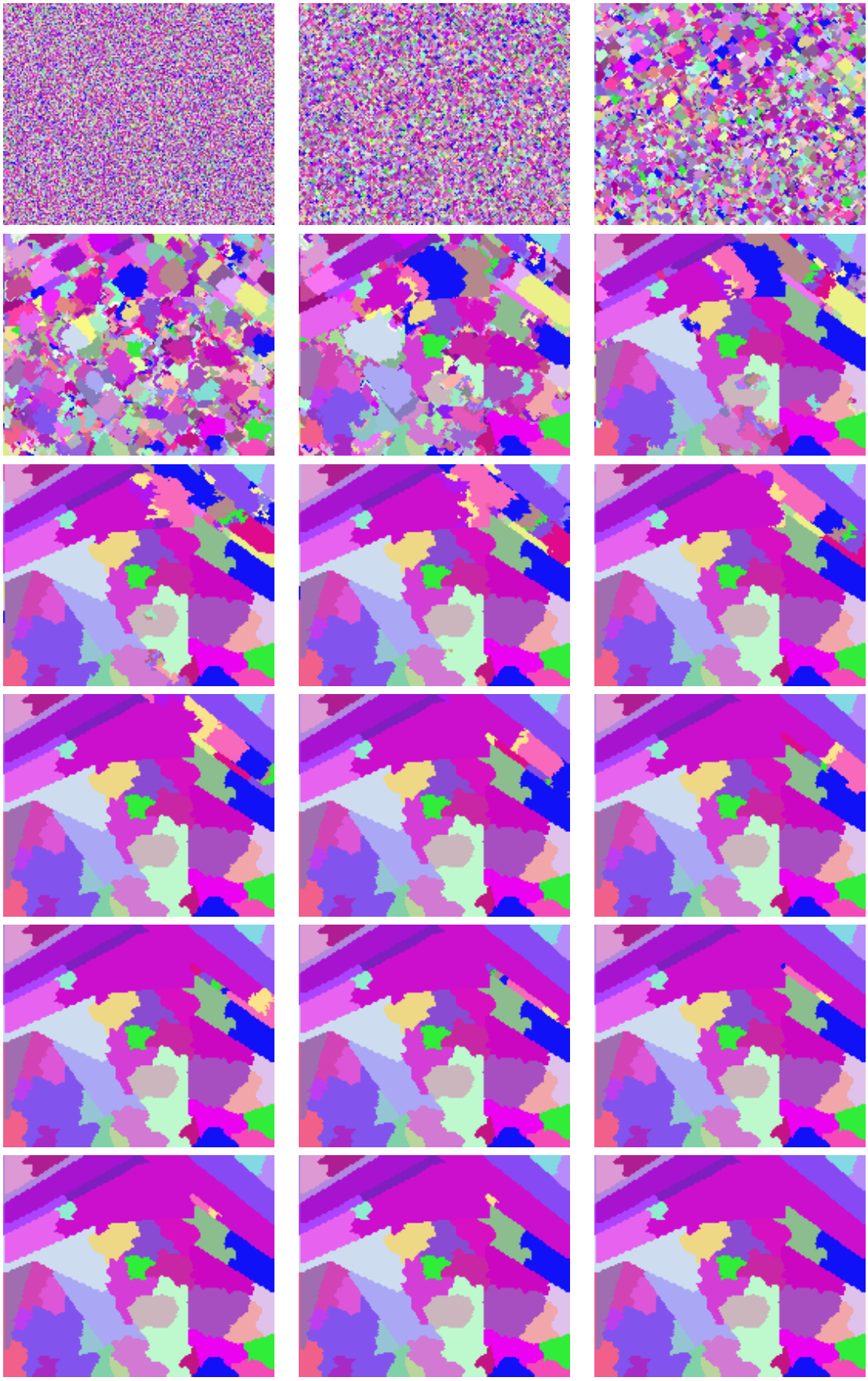


FIG. IV.30 – Evolution des étiquettes (*Foreman* QCIF).

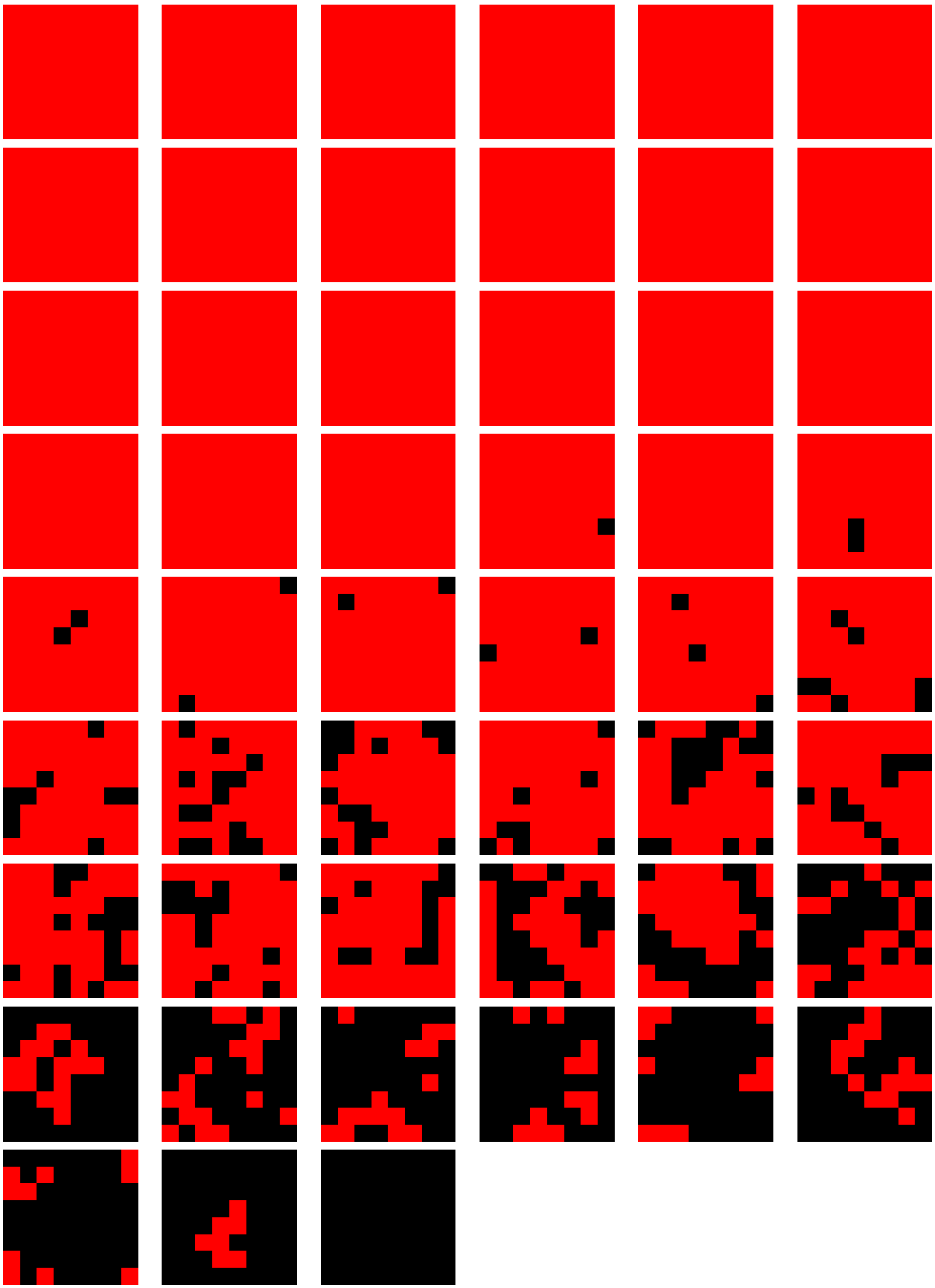


FIG. IV.31 – Activité du réseau 8×8 de processeurs (*Foreman* QCIF).

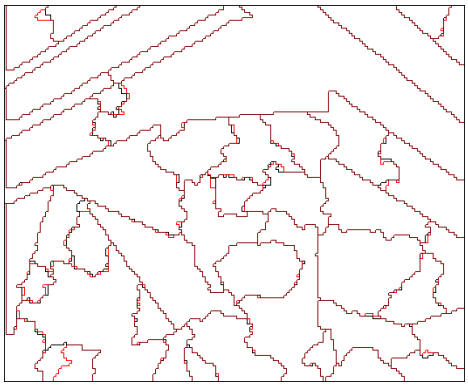


FIG. IV.32 – Comparaison entre les LPE obtenues avec l'algorithme à fine granularité (en noir) et à granularité intermédiaire (en rouge), pour un réseau de 8×8 processeurs.

la comparaison des segments obtenus entre les implantations à fine granularité et à granularité augmentée montre que les variations sont minimes : les partitions obtenues sont fidèles à l'image gradient.

Le partitionnement LPGS est une solution viable pour implanter l'algorithme de *Hill-Climbing* réordonnancé sur une architecture au nombre limité de processeurs. Nous observons une meilleure répartition des charges.

IV.4.4 Conclusion sur la validation par simulation

Nous avons mis en évidence l'influence de l'étiquetage initial sur le flux des données et sur le partitionnement final. Une répartition aléatoire des étiquettes homogénéise la propagation des données dans le réseau et permet un placement plus "naturel" (c'est-à-dire à équidistance des frontières inférieures) des LPE sur les plateaux non-minima.

Grâce à l'environnement de simulation SystemC™, nous avons vérifié l'adéquation de l'architecture avec l'algorithme réordonnancé de segmentation par ascension de colline et ce, quelque soit la granularité. Le comportement du réseau de processeurs est en accord avec le comportement du graphe défini lors de la présentation formelle de l'algorithme (§III.3 page 39). Pour les images naturelles testées, le couple algorithme-architecture génère des régions qui appartiennent au même syntagme (*cf.* paradigme de la segmentation, §II.3.1 page 15).

L'irrégularité et la localité des calculs est confirmée par l'observation d'une concentration de l'activité du réseau en début de processus de segmentation. Au bout de 30% du temps de convergence environ, plus de 90% des régions sont déjà déterminées.

Si les contraintes temporelles sont très fortes, un arrêt du processus de segmentation **avant** le point de convergence peut-être intéressant, même si certaines grandes régions sont encore subdivisées en plusieurs segments.

IV.5 Analyse des résultats de simulation

Cette étude est destinée à mesurer quantitativement, à l'aide d'une métrique adaptée, le comportement des architectures séquentielles et parallèles de la granularité la plus fine jusqu'à la plus grosse. Dans des perspectives de consommation et de temps de traitements, nous pourrons alors estimer plus précisément s'il est possible d'implanter un algorithme de segmentation dans un terminal portable en vue d'un codage vidéo orienté objet (restriction supplémentaire du budget temps alloué pour la segmentation).

Les performances d'un couple algorithme-architecture parallèle sont évaluées grâce à des indicateurs, comme par exemple le temps d'exécution, la scalabilité, la portabilité, la réutilisabilité et divers coûts induits par le matériel, la conception, l'implantation et la maintenance [Mog97] [Fos94]. Optimiser simultanément tous ces paramètres est utopique car des conflits apparaissent, comme par exemple le temps de calcul et le coût matériel (augmentation de la surface de silicium).

Cette section est construite suivant deux points de vue : le premier est orienté coût d'implantation (surface et consommation), alors que le second est orienté temps de segmentation (vitesse). Ainsi, suivant les contraintes, le choix d'une architecture pourra se porter plutôt vers une architecture plus ou moins coûteuse, ou plutôt vers une architecture segmentant plus ou moins rapidement.

Dans un premier temps, la section IV.5.1 présente l'activité de l'architecture : l'estimation de la complexité algorithmique, exprimée sous forme d'actions, permet d'évaluer la charge des processeurs ainsi que sa répartition dans l'architecture. Grâce à la généricité du modèle de simulation, nous étudions l'influence des spécificités des architectures à granularité intermédiaire (dimensions et taille des canaux) sur l'activité des processeurs et des canaux de communication. La simulation d'algorithmes séquentiels et parallèles ainsi instrumentés nous permet d'évaluer, suivant l'architecture, les coûts induit par la segmentation sur plusieurs exemples d'images.

Ensuite, la section IV.5.2 est dédiée au comportement dynamique de l'architecture : à partir d'une estimation de la latence de chaque action exécutée par les processeurs, un temps de traitement est estimé. La fréquence maximale de segmentation est ainsi déduite. De plus, grâce à l'environnement de simulation, l'évolution de la répartition des processeurs actifs au cours du temps est observé.

IV.5.1 Analyse de l'activité du réseau

Une des méthodes pratiques pour estimer la complexité des algorithmes est de mesurer la performance de son implantation en utilisant par exemple les outils de profilage tels que "gprof"[1] ou "tcov"[2]. Cependant, l'instrumentation d'un réseau de dimension 88×72 n'a pu aboutir : le profilage de la bibliothèque SystemC^{TM}nécessite trop de ressources pour notre station de travail[3]. Nous nous sommes donc orientés vers une instrumentation de nos programmes par ajouts d'espions.

[1] GNU Profiling.
[2] Code Coverage Tool : outil de la société SUN.
[3] SUN UltraSparc-10 IIi, 256 Mo de mémoire vive, 440 MHz.

IV.5.1.1 Métriques utilisées

Afin de mesurer les caractéristiques du réseau, le code modélisant l'architecture est instrumenté. L'algorithme de segmentation est décomposé sous forme d'**actions** et d'**itérations** pour chaque processeur.

Définition IV.4 (Action). *Une action est une opération simple exécutable par le processeur.*

Remarque: Suivant l'architecture du processeur (RISC (*Reduced Instruction Set Computer*), DSP (*Digital Signal Processor*)...), une action peut être composée de plusieurs instructions spécifiques au circuit.

Les actions sont classées en cinq grandes catégories :

1. **Lecture :** consommation d'une donnée (niveau de gris, nature, étiquette...) disponible dans la mémoire du canal ;
2. **Ecriture :** dépôt d'une donnée dans un canal insaturé (place vide) ;
3. **Affectation :** mise à jour d'une variable interne (drapeau, compteur...) ;
4. **Test :** évaluation de relations d'ordre ;
5. **Combinaison :** opération logique simple (ET, OU...).

L'action représente un coût élémentaire : suivant les capacités de l'architecture cible (processeur généraliste, DSP, ASIC...) en termes de latence et de consommation pour chaque action, il est possible, à l'aide des estimations de complexité, d'évaluer la faisabilité d'intégration d'un algorithme suivant les contraintes de temps et de coût.

La mesure du nombre d'itérations quantifie le nombre de mises à jour des variables internes d'une machine à état.

Définition IV.5 (Itération). *Une itération est un ensemble d'**actions** qui permettent de faire passer un processeur de l'état courant à l'état suivant. Elle représente un **pas élémentaire** vers le point de convergence.*

Une itération est composée d'une lecture des données d'entrée, des calculs internes et une écriture des résultats vers l'extérieur. La complexité d'une itération, en terme de nombre d'actions, est très variable car cette définition ne fournit aucune précision sur la quantité de calculs effectués.

Pour l'algorithme de segmentation séquentiel, nous présentons chaque méthode sous forme d'un programme symbolique, **au niveau image**. Ensuite, une table présente l'ensemble des indicateurs utilisés pour estimer la complexité. Le contenu d'une itération est alors détaillé **pour un pixel** sous forme d'actions où les lectures et écritures correspondent aux accès à des données d'un tableau. Le branchement des tests est symbolisé par "▷ok," (resp. "▶ko,") dans le cas où la condition est vraie (resp. fausse).

Lors de l'instrumentation de l'algorithme réordonnancé, chaque sous-algorithme et table des indicateurs associée sont présentés **au niveau pixel**.

Définition IV.6 (Taux d'occupation d'un processeur). *Le taux d'occupation d'un processeur est défini par le rapport entre l'accumulation des latences de chaque action effectuée, et le temps de convergence du réseau.*

Approximativement, pour une architecture parallèle asynchrone, ce taux est égal au rapport entre le nombre d'actions effectué par le processeur, et la chaîne critique exprimée sous forme d'actions cumulées.

Remarque: Pour un système monoprocesseur, ce taux est constant et égal à un.

Définition IV.7 (Taux d'activité d'un réseau). *Le taux d'activité d'un réseau est la proportion moyenne, sur le temps de convergence du réseau, du nombre de processeurs actifs par rapport au nombre total de processeurs.*

Pratiquement, ce taux α est déterminé par :

$$\alpha = \frac{\int_0^T n(t)\mathrm{d}t}{N.T} \tag{IV.1}$$

avec T le temps de convergence, n le nombre de processeurs actifs au cours du temps t et N le nombre total de processeurs.

IV.5.1.2 Algorithme séquentiel

Cette étude construit le point de référence des calculs de performances de l'algorithme parallèle proposé par rapport à un algorithme classique de segmentation : l'algorithme par inondation uniforme suivant une file d'attente hiérarchique (FAH, figure II.16 page 27) [Mey91].

L'algorithme séquentiel est décomposé en une suite d'étapes. La description détaillée de ces étapes permet d'introduire les traceurs utilisés, et de justifier la complexité mesurée.
L'algorithme présenté, bien qu'utilisant une file d'attente hiérarchique (FAH), pourrait probablement encore être optimisé. Cependant, l'approximation de la complexité sous forme de nombre d'actions est suffisante pour apprécier les performances de l'algorithme massivement parallèle proposé.

L'algorithme séquentiel classique est composé de cinq étapes :

Algorithme IV.1 Algorithme séquentiel de segmentation par construction des Lignes de Partage des Eaux.

1: Détermination de la nature (algorithme IV.2 page 112)
2: Détection des minima —*algorithme IV.3 page 113*—
3: Étiquetage des minima —*algorithme IV.4 page 114*—
4: Calcul d'histogramme —*algorithme IV.5 page 115, facultatif*—
5: Inondation —*algorithme IV.6 page 115*—

Lors de cette présentation, nous supposons que l'image à segmenter est une image luminance représentée par une structure de grille $G = (V, E, h)$ où, pour tout pixel $p \in V$, $h(p)$ correspond au niveau de gris du pixel et $\mathcal{N}(p)$ le voisinage de p.

Nature. La **nature** d'un pixel est un qualificatif spécifiant sa topographie locale. L'algorithme IV.2 détermine la nature (définition III.5 page 35) de chaque pixel p, où $n(p)$ précise si p est minimum local ML, un pixel intérieur PI, extérieur PE ou non-minimum NM. Cette taxinomie est présentée en détails section B.5.1 page 190.

Algorithme IV.2 Détermination de la nature.

⋆ h : image luminance d'entrée
⋆ n : image nature (ML, PI, PE ou NM)
pour tout $p \in V$ **faire**
IsML ← faux
$\text{Un}_<$ ← faux
$\text{Un}_=$ ← faux
si $\forall q \in \mathcal{N}(p), h(q) > h(p)$ **alors** IsML ← vrai
si $\exists q \in \mathcal{N}(p)$ tq $h(q) < h(p)$ **alors** $\text{Un}_<$ ← vrai
si $\exists q \in \mathcal{N}(p)$ tq $h(q) = h(p)$ **alors** $\text{Un}_=$ ← vrai
si IsML **alors**
n(p) ← ML
sinon si $\neg(\text{Un}_<)$ **alors**
n(p) ← PI
sinon si $\text{Un}_<$ **et** $\text{Un}_=$ **alors**
n(p) ← PE
sinon
n(p) ← NM

Le symbole "¬" correspond à l'inversion logique d'un bit. Le symbole "⋆" correspond aux données requises par l'algorithme.

La détection des minima. Cette étape consiste à localiser les plateaux minima. En effet, les pixels intérieurs PI requièrent un complément d'information afin de déterminer s'ils peuvent initier une inondation ou non du relief. Cette information contenue dans les pixels extérieurs PEest propagée sur la totalité des plateaux non-minima. Une méthode (algorithme IV.3) consiste à balayer l'image jusqu'à idempotence.

	NATURE			
1 it. ⇒	5 lectures	(gris des voisins et du pixel courant)		
	3 affectation	(Init drapeaux)		
(IsML)	4 tests (>)			
	3 comb (&)	▷ok, 1 affectation		
(Un$_<$)	4 tests (<)			
	3 comb (|)	▷ok, 1 affectation		
(Un$_{==}$)	4 tests (=)			
	3 comb (|)	▷ok, 1 affectation		
(Nature)	1 test (=)	▷ok, 1 écriture (ML)		
		▶ko, 1 test (=)	▷ok, 1 écriture (PI)	
			▶ko, 1 tests (=)	
			1 comb (&)	▷ok, 1 écriture (PE)
				▶ko, 1 écriture (NM)

TAB. IV.1 – Nature : répartition des actions effectuées.

Algorithme IV.3 Détection des minima par balayages successifs.

⋆ N_{PE} : le nombre de pixel extérieurs
tant que $N_{PE} \neq 0$ **faire**
  pour tout $p \in V$ **faire**
    si $n(p) =$ PE **alors**
      pour tout $q \in \mathcal{N}(p)$ **faire**
        si $n(q) =$ PI **et** $h(q) = h(p)$ **alors**
          $n(q) \leftarrow$ PE
          $N_{PE} \leftarrow N_{PE} + 1$
      $n(p) \leftarrow$ NM
      $N_{PE} \leftarrow N_{PE} - 1$

À ce stade du traitement, l'image **nature des pixels** ne contient que des pixels référencés ML pour les minima locaux, PI pour les pixels intérieurs situés sur les plateaux minima et NM pour tous les autres pixels. Les points sources sont donc déterminés par ces pixels ML et PI.

La table IV.2 présente l'ensemble des indicateurs utilisés pour estimer la complexité de l'algorithme IV.3.

Étiquetage. Afin d'identifier ces régions, une étiquette unique est attribuée pour chaque source : c'est le rôle de l'étiquetage (algorithme IV.4). Cette étape permet de supprimer la construction des **lignes de partage locales** [Beu90], lignes séparant deux eaux d'un même bassin.

Détection des minima	
1 itération ⇒	1 lecture (nature du pixel courant)
	1 test (=) ▷ok, 4 lectures ($n(p)$=PE)
	4*2 tests (=) (Nature des voisins)
	4*1 comb (&) ▷ok, 1 écriture ($n(q)$=PI)
	1 affectation
	1 écriture (NM)
	1 affectation ($N_{pe} \leftarrow N_{pe} - 1$)

Tab. IV.2 – Détection des minima : répartition des actions effectuées.

Algorithme IV.4 Étiquetage des minima par balayage vidéo.

⋆ n : image nature
⋆ l : étiquettes initialisées à 0 $\forall p \in V$
⋆ $Idem$ = faux —*drapeau binaire égal à "vrai" si l'idempotence est atteinte*—
⋆ $Etiq_{max}$ = largeur ∗ hauteur
⋆ $l_{tmp} = 0$
pour tout $p \in V$ **faire** —*Initialisation*—
  si $n(p) \neq$ NM **alors**
    $l(p) \leftarrow l_{tmp}$
    $l_{tmp} \leftarrow l_{tmp} + 1$
  sinon
    $l(p) \leftarrow Etiq_{max}$
tant que $\neg(Idem)$ **faire** —*Propagation des pixels extérieurs*—
  $Idem \leftarrow$ vrai
  pour tout $p \in V$ **faire**
    si $n(p) \neq$ NM **alors**
      pour tout $q \in \mathcal{N}(p)$ **faire**
        si $l(q) < l(p)$ **alors**
          $l(p) \leftarrow l(q)$
          $Idem \leftarrow$ faux

Durant la phase d'initialisation, l'image est parcourue ligne par ligne dans le sens vidéo, du pixel en haut à gauche vers celui en bas à droite. Une étiquette différente, un nombre entier positif, est attribuée à chaque pixel marqué ML ou PI. Le but étant d'attribuer l'étiquette minimale du plateau minimum à tous ses pixels, le majorant $Etiq_{max} = largeur * hauteur$ est utilisée pour initialiser tous les pixels de nature NM. La phase d'unification des étiquettes consiste à attribuer, au pixel en cours de traitement, l'étiquette la plus petite parmi ses voisins.

La table IV.3 présente l'ensemble des indicateurs utilisés pour estimer la complexité de la phase d'étiquetage (algorithme IV.4).

Cet algorithme procède uniquement par balayage direct vidéo. Cependant, une succession de balayages vidéo et inverse-vidéo permettrait d'atteindre le point de convergence plus rapidement [Vin93] mais ceci n'a pas été pris en compte ici. L'image est maintenant prête pour l'inondation.

	Initialisation de l'étiquetage
1 it ⇒	1 lecture (nature du pixel courant) 1 test ($\neq$) ▷ok, 1 affectation 1 écriture ▶ko, 1 écriture

	Propagation des pixels extérieurs
1 it ⇒	1 test ($\neq$) ▷ok, 4*1 test ($<$) ▷ok, 1 écriture (recherche de l'étiquette minimale) ($l(p) \neq Etiq_{max}$) 1 affectation

Tab. IV.3 – Étiquetage : répartition des actions effectuées.

Calcul de l'histogramme. Cette partie du traitement n'est pas spécifique à l'algorithme d'inondation mais permet de préparer l'organisation mémoire occupée par la file d'attente hiérarchique (FAH) [Nog98, chap.3]. En effet, comme nous le verrons dans le paragraphe suivant, chaque file est susceptible de contenir tous les pixels de même niveau de gris. L'histogramme (algorithme IV.5) nous permet donc de connaître, avant l'inondation, la taille maximale de chaque file d'attente.

Algorithme IV.5 Calcul d'histogramme.

⋆ H : histogramme de l'image initialisé à 0
$\forall p \in V$ **faire** $H\big[h(p)\big] \leftarrow H\big[h(p)\big] + 1$

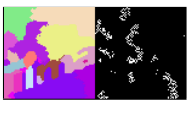
Ces calculs représentent un coût d'une lecture et d'une affectation (mise à jour de l'histogramme) par pixel. Ils peuvent être réalisés simultanément avec l'initialisation de l'étiquetage.

Inondation suivant une FAH. La priorité de chaque file de la FAH correspond au niveau de gris du jeton, le pixel. Cet algorithme, décrit initialement dans [Mey91] et repris dans [Nog98], simule l'inondation du relief par propagation des étiquettes. Cet algorithme est optimal dans le sens où chaque pixel n'est traité qu'une seule fois.

Algorithme IV.6 Inondation par File d'Attente Hiérarchique (FAH).

⋆ h : image de départ
⋆ l : étiquettes des marqueurs initiaux —les minima ici—
⋆ FAH : ensemble de FIFO dimensionnées suivant l'histogramme

```
                                        —Initialisation : recherche des points source—
pour tout p ∈ V faire          —Etiq_max correspondant à un pixel non marqué—
  si l(p) ≠ Etiq_max et ∃q ∈ 𝒩(p) tq l(q) = Etiq_max alors
    insérer {p} dans FAH[h(p)]
tant que FAH n'est pas vide faire                          —Inondation—
  extraire {p}
  pour tout q ∈ 𝒩(p) faire
    si l(q) = Etiq_max alors
      insérer {q} dans FAH[h(q)]
      l(q) = l(p)
```

La table IV.4 présente l'ensemble des indicateurs utilisés pour estimer la complexité de l'algorithme IV.6.

Initialisation de la FAH

1 it ⇒	1 lecture (lecture de l'étiquette du pixel courant)	
	1 test (≠) ▷ok, 4*1 test (=) ▷ok, (Est-ce un minimum ?)	1 affectation
		1 écriture (mise à jour étiquette)

Inondation suivant la FAH

1 it ⇒	1 lecture (extrait le pixel)	
	4*1 test (=) ▷ok,	1 affectation (insère le pixel dans FAH)
		1 écriture (attribue l'étiquette au pixel inondé)

Tab. IV.4 – Inondation par FAH : descriptif détaillé des indicateurs.

Complexité estimée. La table IV.5 présente les résultats de profilage obtenus pour la simulation de l'image gradient simplifiée de *Foreman* SQCIF.

Taille de l'image :	88×72 pixels			global	par pixel
Nombre d'itérations :		**Total**	=	**201 917**	**31.9**
	Minima	Nature	=	6 336	1.0
		Détection	=	76 032	12.0
	Label	Initialisation	=	6 336	1.0
		Propagation	=	101 376	16.0
	FAH	Initialisation	=	6 336	1.0
		Inondation	=	5 501	0.9
Nombre d'actions :		**Total**	=	**699 531**	**110.4**
		Lectures	=	148 902	23.5
		Ecritures	=	25 583	4.0
		Tests	=	402 360	63.5
		Affectations	=	48 387	7.6
		Combinaisons	=	74 299	11.7
Chaîne critique (actions) :				**699 531**	

Tab. IV.5 – Mesure de complexité pour l'algorithme séquentiel par FAH (*Foreman* SQCIF).

La table IV.6 présente la charge relative de chaque étape de l'algorithme séquentiel par rapport à la chaîne totale de segmentation.

L'inondation est optimale puisqu'une itération par pixel est nécessaire (le nombre moyen d'itérations pour l'inondation par FAH est inférieur à un puisque seuls les pixels non-minima sont traités dans cette partie).

Le pourcentage du nombre d'itérations réalisées par les différentes phases de l'algorithme montre que la détermination des minima et étiquetage des plateaux minima sont les plus coûteux. La table IV.7 présente les estimations de complexité effectuées pour l'image *Susie* (QCIF).

Étape	Nature	Détection des minima	Initialisation étiquetage	Histo-gram-me	Etique-tage	Initialisa-tion de la FAH	Inondation
Pro-por-tion	31.3%	30.3%	3%	1.9%	25.2%	2.8%	5.5%

TAB. IV.6 – Proportion de la charge de calcul, estimée à partir du nombre d'actions comptabilisé, nécessaire pour chaque étape (*Foreman* SQCIF).

Taille de l'image :	176 × 144 pixels			global	par pixel
Nombre d'itérations :		**Total**	=	**1 389 741**	**54.8**
	Minima	Nature	=	25 344	1.0
		Détection	=	405 504	16.0
	Label	Initialisation	=	25 344	1.0
		Propagation	=	887 040	35.0
	FAH	Initialisation	=	25 344	1.0
		Inondation	=	21 165	0.8
Nombre d'actions :		**Total**	=	**4 204 658**	**165.9**
		Lectures	=	694 678	27.41
		Ecritures	=	99 488	3.9
		Tests	=	2 931 540	115.7
		Affectations	=	192 926	7.6
		Combinaisons	=	286 026	11.3
Chaîne critique (actions) :				4 204 658	

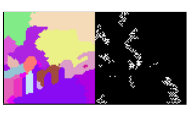
TAB. IV.7 – Mesure de complexité pour l'algorithme séquentiel par FAH (*Susie*).

Il est intéressant de noter que plus de 150 actions sont exécutées par pixel : sachant qu'une action peut-être composée de plusieurs instructions (au sens cœur de processeur), ce coût est élevé dans le cadre des processeurs dédiés à l'embarqué.

La table IV.8 présente des répartitions de charges similaires à celles de l'image *Foreman* SQCIF, sauf pour l'étiquetage (46.3% pour *Susie* contre 25.2% pour *Foreman*). Cette différence est en partie justifiée par la taille des plateaux minima (figures IV.16 et IV.18 page 94) : la propagation de l'étiquette minimale sur les plateaux nécessite un nombre de balayages d'autant plus important, que la surface de ces plateaux est grande (proportionnellement à la taille de l'image).

Étape	Nature	Détection des minima	Initialisation étiquetage	Histo-gram-me	Etique-tage	Initialisa-tion de la FAH	Inondation
Pro-por-tion	20.8%	23.8%	2.0%	1.2%	46.3%	2.3%	3.5%

TAB. IV.8 – Proportion de la charge de calcul nécessaire pour chaque étape, estimée à partir du nombre d'actions comptabilisé (*Susie*).



Activité de l'architecture L'architecture étant composé que d'un seul processeur, celui-ci calcule en permanence. Donc le taux d'occupation du processeur est maximum : 100% (figure IV.33).

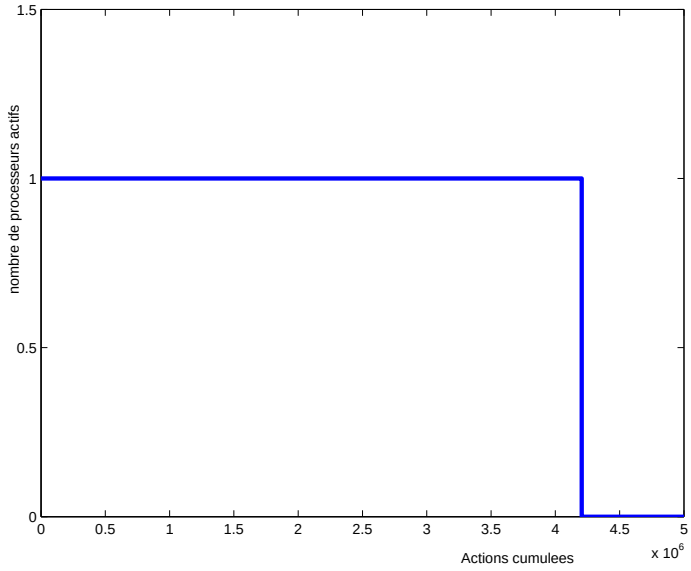


FIG. IV.33 – Taux d'occupation de l'architecture séquentielle : à 100% durant tout le processus de segmentation (*Susie* QCIF).

Remarque: L'abscisse de ces graphes est exprimée en temps, alors que cette section est dédiée à la quantification du coût en actions. Nous verrons à la section IV.5.2 que la moyenne de ces deux mesures sont proportionnelles.

Conclusion. À la vue de cette première estimation de complexité, nous observons une large proportion de la complexité induite par la recherche des minima et leur étiquetage, environ 89,8%[1] pour *Foreman* et 93% pour *Susie* : **les étapes de recherche des minima et d'étiquetage sont des opérations coûteuses**, par rapport à l'étape d'inondation.

Bien qu'aucune optimisation ne soit effectuée pour l'étiquetage des plateaux minima, ces mesures de complexité reflète toutefois approximativement la charge de calcul d'un algorithme de segmentation séquentiel **adapté à un terminal portable**. En effet, l'étape d'inondation est optimale, et cette limite ne pourrait être atteinte par une architecture ne pouvant supporter une gestion mémoire complexe induite par les files d'attentes : l'accès aléatoire à un espace mémoire induit des temps d'attente supplémentaires. Un algorithme d'inondation par balayage successifs de l'image présenterait un accès plus régulier à la mémoire, mais aussi une complexité beaucoup plus grande.

C'est pourquoi, nous estimons que la non-optimisation des post-traitements est compensée par la sous-estimation de la complexité du processus d'inondation induite par une FAH difficile à optimiser sur une architecture simple.

[1] Calcul d'histogramme, initialisation de la FAH et inondation.

Nous verrons, lors de l'analyse de vitesse présentée à la section IV.5.2, que ces coûts sont trop importants pour une implantation dans un terminal portable. Il est alors nécessaire, soit d'optimiser l'algorithme séquentiel, soit de rechercher une architecture dédiée optimale pour l'implantation d'un algorithme de segmentation.

Étant donné que le premier point a déjà fait l'objet de nombreuses recherches et a abouti sur l'utilisation de files d'attentes hiérarchiques (délicates à implanter par un ASIC [Lem96]), ces études ont confirmé notre orientation de recherche : une architecture de processeurs asynchrones faiblement couplés.

IV.5.1.3 Partitionnement à fine granularité

Nous présentons les étapes de l'algorithme de *Hill-Climbing* réordonnancé, implantées au sein de chaque pixel. Elles sont donc décrites par rapport au processeur courant, représenté par la variable p de l'ensemble des nœuds du réseau.

L'algorithme IV.7 décrit la machine à trois états introduite au paragraphe III.2 page 34. Il est composé de trois états :

1. L'initialisation (algorithme IV.8).
2. L'unification des étiquettes (algorithme IV.9).
3. L'inondation (algorithme IV.10).

Algorithme IV.7 Machine à trois états (variable d'état φ).

```
⋆ φ=INIT
  boucle∞
    Lit suivant 𝒩(p)
    si φ=INIT alors
      Envoie suivant 𝒩(p)
    sinon
      Envoie suivant 𝒩^≥(p)

    si φ=INIT alors
      si 4 données reçues alors
        Initialisation                          —algorithme IV.8—
    sinon si φ=MP alors
      Unification des étiquettes                —algorithme IV.9—
    sinon
      Inondation                                —algorithme IV.10—
```

À présent, détaillons chacune des trois étapes, où nous précisons la répartition des actions utilisées.

Initialisation. La phase d'initialisation (algorithme IV.8) consiste à construire trois drapeaux de quatre bits où chaque bit correspond aux voisins nord, est, sud et ouest : $\mathcal{N}^{<}(p), \mathcal{N}^{=}(p)$ et $\mathcal{N}^{I}(p)$. Ils permettent de décrire la topologie locale du pixel p.

Algorithme IV.8 Initialisation.

⋆ p : pixel associé au processeur
⋆ $h'(p) = h(p)$ —*h' correspond à l'altitude du pixel lors de mises à jour successives*—
⋆ $\varphi =$ MP
⋆ $\mathcal{N}^I(p) = \emptyset$
pour tout $q \in \mathcal{N}(p)$ **faire**
  si $h(q) = h(p)$ **alors**
    $\mathcal{N}^=(p) \leftarrow \mathcal{N}^=(p) \cup \{q\}$ —*Construction de $\mathcal{N}^=(p)$*—
  si $h(q) < h(p)$ **alors**
    $\mathcal{N}^<(p) \leftarrow \mathcal{N}^<(p) \cup \{q\}$ —*Construction de $\mathcal{N}^<(p)$*—
  si $h(q) < h'(p)$ **alors**
    $h'(p) \leftarrow h(q)$ —*Construction de $\mathcal{N}^I(p)$*—
    $l(p) \leftarrow l(q)$
    $\mathcal{N}^I(p) \leftarrow \{q\}$
    $\varphi \leftarrow$ NM

La table IV.9 présente l'ensemble des indicateurs utilisés pour l'estimation de complexité. La fonction $|.|$ retourne le cardinal d'un ensemble.

	Initialisation	
1 itération $\Rightarrow$	4 lectures	
(Drapeau $\mathcal{N}^=(p)$)	4 tests (=)	$\triangleright$ok, 1 affectation 1 comb.
(Drapeau $\mathcal{N}^<(p)$)	4 tests (<)	$\triangleright$ok, 1 affectation 1 comb.
(Drapeau $\mathcal{N}^I(p)$)	4 tests (<) $\left\lceil \lvert\mathcal{N}^{\geqslant}(p)\rvert \right\rceil$ écritures	$\triangleright$ok, 3 affectation

Tab. IV.9 – Initialisation : répartition des actions effectuées.

La phase d'initialisation terminée, la machine à état effectue une transition vers l'état MP ou NMen fonction de la topologie locale de p. L'algorithme utilisé est soit l'algorithme IV.9 si aucun voisin d'inondation n'est trouvé, soit l'algorithme IV.10 dans le cas contraire.

Minimum ou Plateau. L'automate est dans l'état MP : le pixel p n'a pu déterminer un voisin d'inondation ($\mathcal{N}^I(p)$ est vide). Grâce à l'algorithme IV.9, le pixel p unifie l'étiquette du plateau sur lequel il se trouve.

Algorithme IV.9 Unification des étiquettes.

```
⋆ φ = MP
  pour tout q ∈ N⁼(p) faire
    si m(q) < m(p) alors
      m(p) ← m(q)
      l(p) ← l(q)
      N^I(p) ← {q}
      φ ← NM
    sinon si l(q) < l(p) alors
      l(p) ← l(q)
```

La table IV.10 décrit l'instrumentation de l'algorithme IV.9.

	Minimum ou Plateau	
1 itération ⇒	$\left\lfloor \lvert\mathcal{N}^{=}(p)\rvert \right\rfloor$ lectures	
	$\left\lfloor \lvert\mathcal{N}^{=}(p)\rvert \right\rfloor$ tests (<)	▷ok, 3 affectation (signal PE)
		▶ko, 1 test (<) (Recherche étiquette minimale)
		▷ok, 1 affectation (Unification)
	$\left\lfloor \lvert\mathcal{N}^{\geqslant}(p)\rvert \right\rfloor$ écritures	

Tab. IV.10 – Minimum ou Plateau : répartition des actions effectuées.

Remarque: Pour simplifier, la table IV.10 présente des lectures sur tous les voisins de même altitude alors que les opérations ne sont effectuées que sur les ports actifs.

Non minimum. Enfin, l'algorithme IV.10 présente le comportement des pixels inondés : l'automate est dans l'état NM. Une simple copie des données d'entrée, suivant $\mathcal{N}^I(p)$, suffit car l'étiquette du minimum d'attraction converge et la réduction à un du demi-degré intérieur conformément au relief topographique autorise uniquement la propagation des données suivant la ligne de plus grande pente. Une large proportion de processeurs se trouve dans cet état pour une image naturelle.

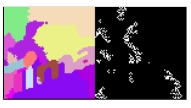
Algorithme IV.10 Inondation.

```
⋆ φ = NM
⋆ |N^I(p)| = 1
  l(p) ← l(N^I(p))
  m(p) ← m(N^I(p))
```

De même, la table IV.11 présente l'instrumentation de l'algorithme IV.10.

Mesures de complexité. Afin de présenter les mesures de complexité sous forme de tableau (tableaux IV.12 page 122 et IV.13 page 124), les symboles suivants sont définis :

– $\sum$: la somme des itérations effectuées par tous les processeurs ;



	Non-Minimum
1 itération ⇒	1 lecture
	3 affectations
	$\left\lceil \lvert\mathcal{N}^{\geqslant}(p)\rvert \right\rceil$ écritures

Tab. IV.11 – Non Minimum : répartition des actions effectuées.

- $\overline{\mu p}$: la moyenne du nombre d'itérations effectuées par processeur ;
- $\wedge_{(i,j)}$: le nombre d'itérations ou actions minimum (réalisé par le processeur de la $i^{ème}$ ligne, $j^{ème}$ colonne. i et j représentent les coordonnées du processeur ayant réalisé ce minimum) ;
- $\vee_{(i,j)}$: le nombre d'itérations ou actions maximum (réalisé par le processeur de la $i^{ème}$ ligne, $j^{ème}$ colonne).

Taille de l'image : 88×72 pixels

Nombre d'itérations :	$\sum$ =	**33 804**	→	$\overline{\mu p}$ =	**5.3**
	$\wedge_{(2,6)}$ =	1		$\vee_{(17,51)}$ =	17
Init :	$\sum$ =	6336	→	$\overline{\mu p}$ =	1.0
mp :	$\sum$ =	7 734	→	$\overline{\mu p}$ =	1.2
	$\wedge_{(1,15)}$ =	1		$\vee_{(17,51)}$ =	16
nm :	$\sum$ =	19 734	→	$\overline{\mu p}$ =	3.1
	$\wedge_{(1,5)}$ =	1		$\vee_{(28,17)}$ =	12
Nombre d'actions :	$\sum$ =	**542 394**	→	$\overline{\mu p}$ =	**85.6**
	$\wedge_{(5,88)}$ =	48		$\vee_{(17,51)}$ =	225
lectures :	$\sum$ =	54 045	→	$\overline{\mu p}$ =	8.5
	$\wedge_{(2,6)}$ =	4		$\vee_{(12,56)}$ =	23
écritures :	$\sum$ =	62 377	→	$\overline{\mu p}$ =	9.8
	$\wedge_{(1,21)}$ =	3		$\vee_{(46,84)}$ =	39
tests :	$\sum$ =	232 825	→	$\overline{\mu p}$ =	36.7
	$\wedge_{(2,6)}$ =	24		$\vee_{(17,51)}$ =	152
combinaisons :	$\sum$ =	112 557	→	$\overline{\mu p}$ =	17.8
	$\wedge_{(1,4)}$ =	13		$\vee_{(29,16)}$ =	27
affectations :	$\sum$ =	80 590	→	$\overline{\mu p}$ =	12.7
	$\wedge_{(2,6)}$ =	1		$\vee_{(29,16)}$ =	36
Chaîne critique (actions cumulées) :				614.6	

Tab. IV.12 – Mesure de complexité pour l'algorithme parallèle (image *Foreman* SQCIF).

Le tableau IV.12 présente les résultats d'une des simulations[1] pour l'image *Foreman* SQCIF. En moyenne, chaque processeur effectue cinq mises à jour (itérations)

[1]Le comportement étant asynchrone, les résultats de simulations successives sont légèrement différents.

de ses variables d'état. La complexité ramenée au niveau pixel est donc faible. Cependant, la variance est forte : une itération pour les minima locaux et les minima d'étiquette minimale, jusqu'à 17 itérations pour le plus chargé. De même, les écarts sur les mesures du nombre d'actions sont élevés, ceci se traduit par un fort déséquilibre des charges.

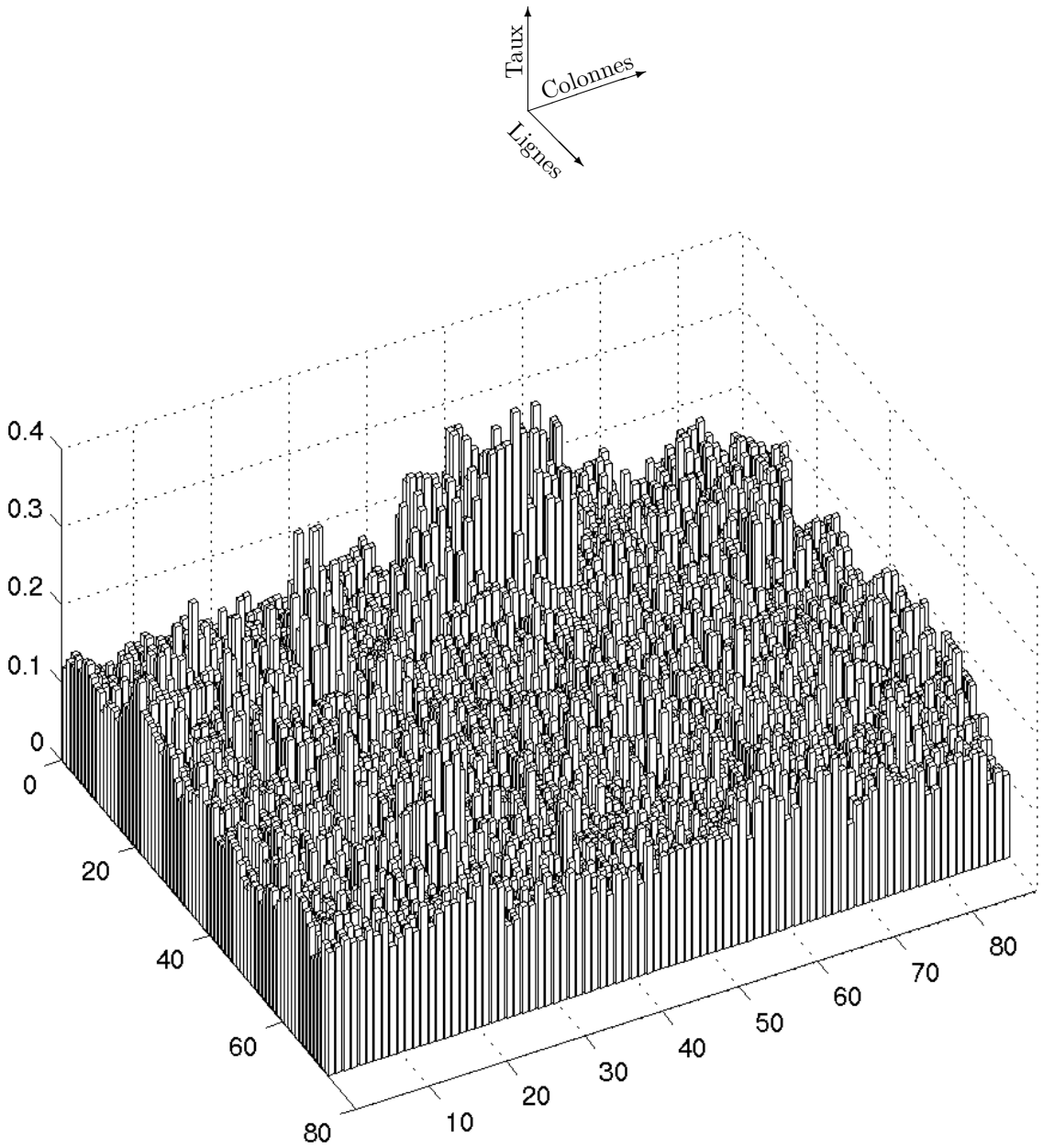


Fig. IV.34 – Taux d'occupation des processeurs pour l'image *Foreman* SQCIF.

La figure IV.34 présente le taux d'occupation (définition IV.6 page 111) des processeurs. Ce graphe illustre la principale caractéristique du réseau à fine granularité : **le déséquilibre des charges**. Le taux d'occupation du processeur (définition IV.6 page 111) le plus actif représente environ 40% de la chaîne critique, et une moyenne de 15% seulement sur l'ensemble des processeurs.

La répartition non homogène des charges se traduit sous forme d'îlots d'activité, particulièrement au niveau du casque du personnage (ligne 15, colonne 45).

Ce faible taux d'occupation s'explique par la localité des traitements et des communications interprocesseurs : la propagation d'un message depuis un processeur (un minimum) vers un processeur éloigné (une feuille de l'arborescence) est longue mais ne demande que peu d'activité pour chacun des processeurs.

La mesure d'activité des canaux de communication est délicate à mesurer. En effet, les communications étant non-bloquantes, il faut détecter le nombre de données non lues avant qu'elles soient écrasées. C'est pourquoi, à ce niveau de cette étude, une mesure précise de la proportion des communications effectives dans le flot de données ne peut-être fournie.

Taille de l'image : 176×144 pixels

Nombre d'itérations :	$\sum$	=	**183 776**	→	$\overline{\mu p}$	=	**7.3**
	$\wedge_{(2,121)}$	=	1		$\vee_{(60,18)}$	=	31
INIT :	$\sum$	=	25 344	→	$\overline{\mu p}$	=	1.0
MP :	$\sum$	=	74 575	→	$\overline{\mu p}$	=	2.9
	$\wedge_{(1,67)}$	=	1		$\vee_{(60,18)}$	=	30
NM :	$\sum$	=	83 857	→	$\overline{\mu p}$	=	3.3
	$\wedge_{(1,75)}$	=	1		$\vee_{(125,22)}$	=	17
Nombre d'actions :	$\sum$	=	**2 692 612**	→	$\overline{\mu p}$	=	**106.2**
	$\wedge_{(144,37)}$	=	48		$\vee_{(60,18)}$	=	365
lectures :	$\sum$	=	272 931	→	$\overline{\mu p}$	=	10.8
	$\wedge_{(2,121)}$	=	4		$\vee_{(3,2)}$	=	40
écritures :	$\sum$	=	309 328	→	$\overline{\mu p}$	=	12.2
	$\wedge_{(144,176)}$	=	3		$\vee_{(124,21)}$	=	53
tests :	$\sum$	=	1 297 391	→	$\overline{\mu p}$	=	51.2
	$\wedge_{(2,121)}$	=	24		$\vee_{(60,18)}$	=	264
combinaisons :	$\sum$	=	464 768	→	$\overline{\mu p}$	=	18.3
	$\wedge_{(1,59)}$	=	13		$\vee_{(125,22)}$	=	33
affectations :	$\sum$	=	348 194	→	$\overline{\mu p}$	=	13.7
	$\wedge_{(10,128)}$	=	1		$\vee_{(126,22)}$	=	45
Chaîne critique (actions cumulées) :					1 928.9		

TAB. IV.13 – Mesure de complexité pour l'algorithme parallèle (image *Susie* QCIF).

La table IV.13 présente la complexité de l'algorithme pour l'image *Susie* QCIF. La chaîne critique de 1 928 actions cumulées pour cette image, par rapport aux 615 actions pour l'image *Foreman* SQCIF, est en partie due à la large région du fond située à gauche (figure IV.23 page 99).

Par construction parallèle, pour des images où la taille des régions est similaire, nous obtiendrions des chaînes critiques similaires et ce, quelque soit la dimension de l'image. Or, la chaîne critique pour l'image *Susie* est bien plus élevée que celle de

l'image *Foreman*, donc ces mesures exhibent une forte dépendance des charges de calcul aux images à segmenter, et en particulier à la présence de grandes régions.

Par ailleurs, cette image est marquée par un large plateau minimum (figure IV.18c page 95c), ce qui se traduit, dans la table IV.13, par un nombre important d'itérations opéré par des pixels à l'état MP. Comme les itérations d'unification sont de complexité bien supérieure à celle d'inondation, l'unification des étiquettes de larges plateaux minima représente une forte proportion de la charge de calcul totale. Cette remarque montre une fois de plus l'intérêt d'effectuer un étiquetage initial aléatoire : le "glissement" d'étiquettes ordonnées sur un plateau est plus coûteux que la diffusion concentrique des étiquettes aléatoires (tout pixel à l'intérieur d'un îlot est inactif), car il implique un nombre d'unifications transitoires supérieur.

Activité du réseau. La figure IV.35a détaille l'évolution du nombre de processeurs actifs présentée par la figure IV.24 (Susie QCIF). Ces mesures, et celles effectuées sur les images *Foreman* SQCIF (figure IV.35b) et QCIF (figure IV.35c), montrent la décroissance exponentielle (pente constante de la courbe dans un repère semi-log) du nombre de processeurs actifs au cours du processus de segmentation (en nombre d'actions cumulées).

Le faible taux d'activité du réseau (définition IV.7) montre que les charges sont fortement déséquilibrées.

Conclusion. L'analyse de l'activité du réseau pour une fine granularité montre une grande variance de la charge des processeurs. La charge des calculs étant fortement déséquilibrée, un faible taux d'occupation des processeurs est observée. Cette observation est confirmée par la chute exponentielle du nombre de processeurs actifs au cours de la segmentation.

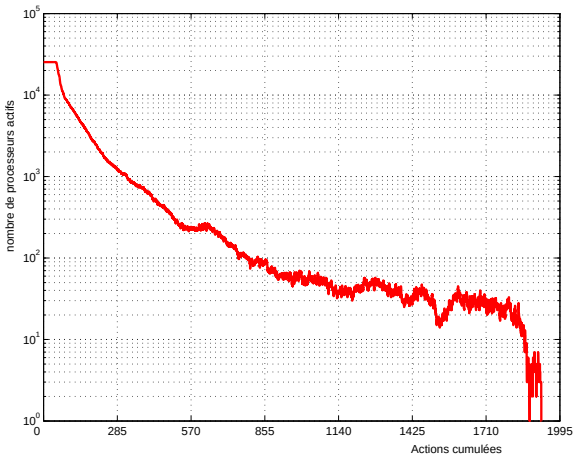
Dans des perspectives de consommation, ces mesures montrent l'intérêt d'utiliser des processeurs capables de se mettre en veille automatiquement, tel que le propose la conception de circuits asynchrones. En effet, si un processeur inactif consomme très peu d'énergie, alors le profil de la consommation sera semblable à celui de l'activité globale du réseau.

Par contre, la sous utilisation des ressources nous pousse à étudier l'influence de la granularité sur le temps de convergence de l'algorithme, la complexité, et le flux des données dans les canaux de communication.

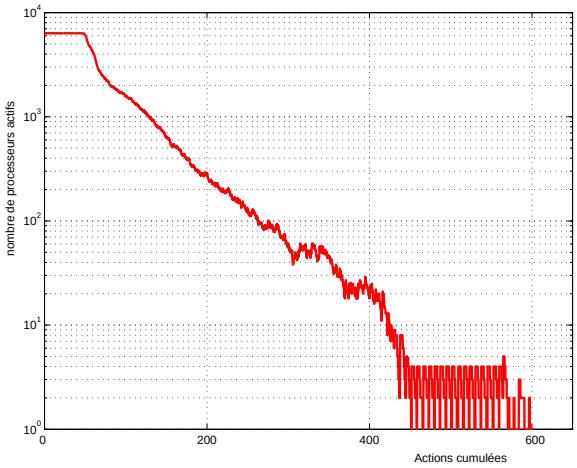
IV.5.1.4 Partitionnement à granularité intermédiaire

Cette partie étudie l'influence de la granularité et la taille des files d'attente sur le nombre d'itérations, d'actions, d'échecs d'écritures et enfin, des communications inutiles. Nous rappelons que pour cette architecture, des communications non-bloquantes où les anciennes données sont prioritaires, sont utilisées pour éviter la perte des données.

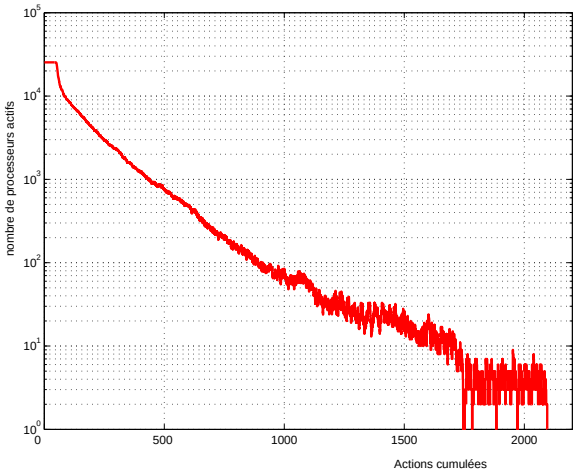
Pour les images de test *Foreman* et *Susie* QCIF, les granularités testées vont de quatre pixels par processeur (réseau de 88×72 processeurs) jusqu'à un bloc SQCIF de pixels par processeur (réseau de 2×2 processeurs).



(a) *Susie* QCIF (taux d'activité du réseau : 5.6%)



(b) *Foreman* SQCIF (taux d'activité du réseau : 14.5%)



(c) *Foreman* QCIF (taux d'activité du réseau : 6.1%)

FIG. IV.35 – Profil du nombre de processeurs actifs en fonction du temps (granularité la plus fine).

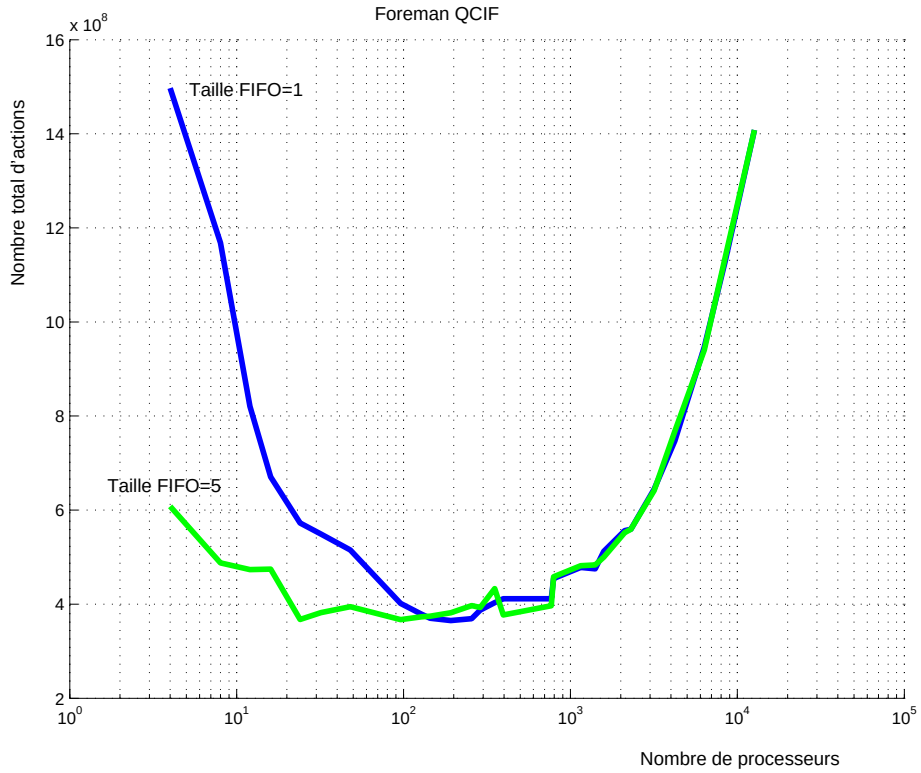


FIG. IV.36 – Evolution du nombre d'actions en fonction de la granularité (*Foreman* QCIF).

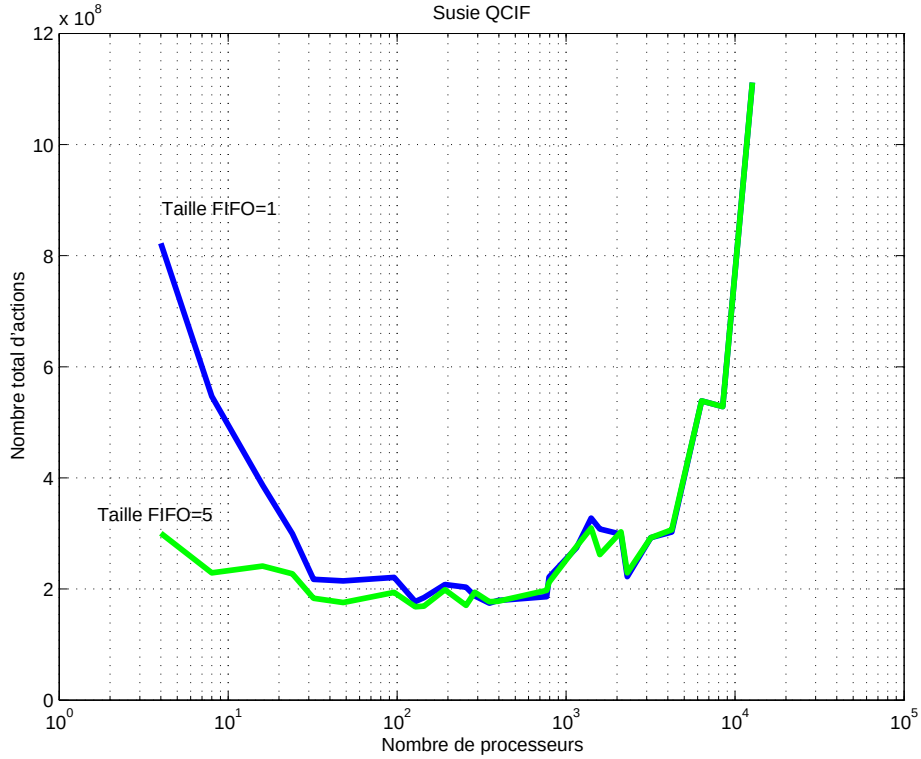
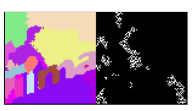


FIG. IV.37 – Evolution du nombre d'actions en fonction de la granularité (*Susie* QCIF).



Influence de la granularité sur le nombre total d'actions. Les figures IV.36 et IV.37 présentent l'évolution du nombre total d'actions, *i.e.* la complexité algorithmique cumulée de l'ensemble des processeurs du réseau, en fonction de la granularité. Le minimum de cette courbe correspond à une granularité de 100 pixels par processeur environ, soit un réseau de 16×12 ou 11×12 processeurs. Elle représente un compromis entre les nombreuses inondations multiples pour la fine granularité (de nombreuses données transitoires sont inutilement propagées et traitées par les processeurs), et les nombreux calculs des coordonnées des pixels et engorgement des canaux de communication pour les plus grosses granularités.

Par ailleurs, une faible taille des FIFO est suffisante. Concevoir une architecture d'une centaine de processeurs environ interconnectés par des canaux ayant une mémoire plus importante serait inutile.

Cette analyse peut-être vue comme une conséquence de la faible proportion de pixels qui contribuent à la convergence l'algorithme (§IV.5.1.3). Les besoins en terme de flux des données chutent exponentiellement (le profil de l'activité des processeurs pour une fine granularité, figure IV.35c page 126, illustre ce propos).

Cette analyse peut-être généralisée aux algorithmes fortement localisés : un partitionnement LPGS peut-être implanté sur un réseau dont les canaux ont une faible capacité de mémorisation (les algorithmes asynchrones de reconstruction morphologique [Rob97] font parties de cette classe).

Inversement, afin de limiter les surcoûts de communication, plus un algorithme effectue des traitements globaux impliquant la majorité des pixels, plus la taille des FIFO doit être élevée (sous l'hypothèse que la stratégie des communications non-bloquantes conserve les données anciennes).

Influence de la granularité sur l'engorgement des FIFO. Afin de compléter l'analyse précédente, nous évaluons l'évolution du surcoût induit par la saturation des files d'attentes.

Le profil des graphes (figures IV.38 et IV.39) montre que naturellement, plus la taille des FIFO est grande, moins il y a d'échecs pour écrire une donnée : plus les mémoires sont grandes, plus les canaux "supportent" les variations du flux des données.

Par ailleurs, nous observons que plus la granularité est grosse, plus les canaux de communication sont engorgés : la figure IV.40 présente trois coupes à granularité constante. Le partitionnement LPGS est donc adapté à une implantation parallèle à granularité intermédiaire d'algorithmes fortement localisés.

Remarque: L'utilisation du partitionnement LSGP (2D-rectilinéaire) présente généralement une caractéristique antagoniste : plus la granularité est grosse, moins les canaux sont engorgés, puisque les traitements sont localisé au sein même de chaque processeur.

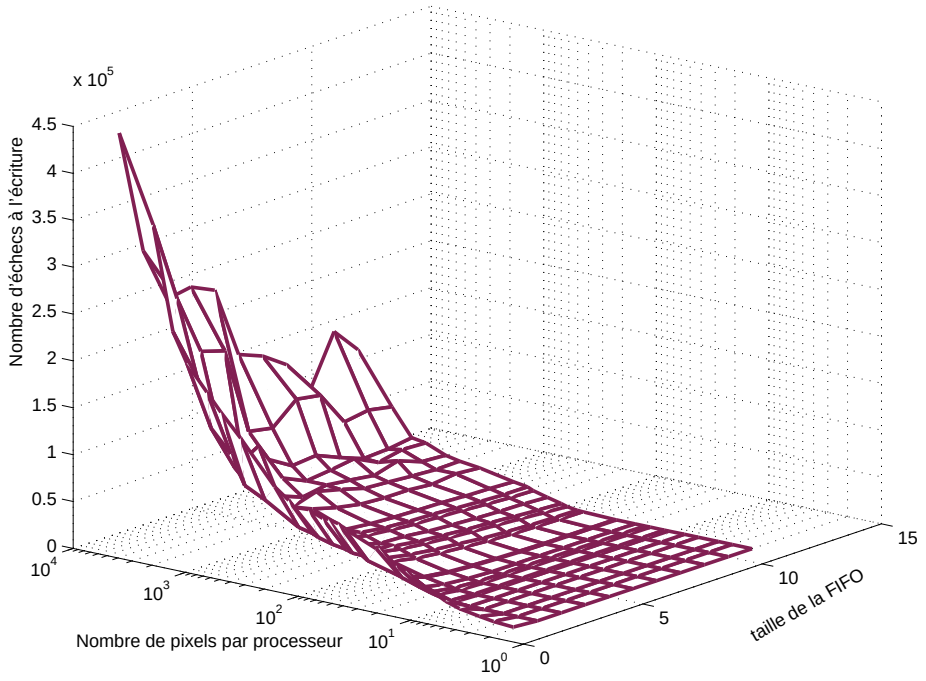


FIG. IV.38 – Influence de la granularité sur l'engorgement des FIFO (*Foreman* QCIF).

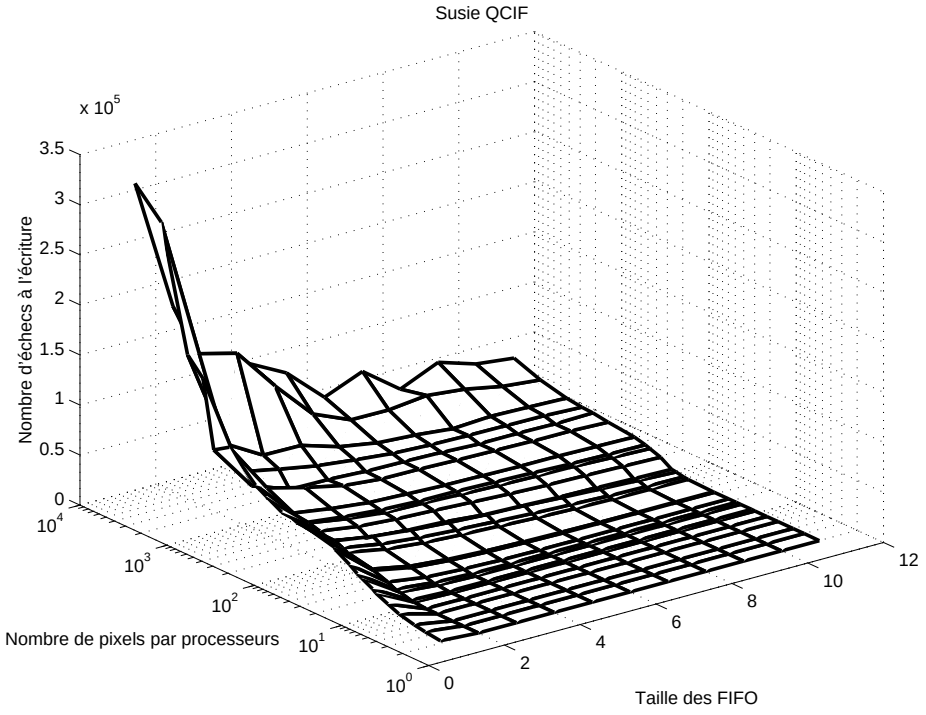
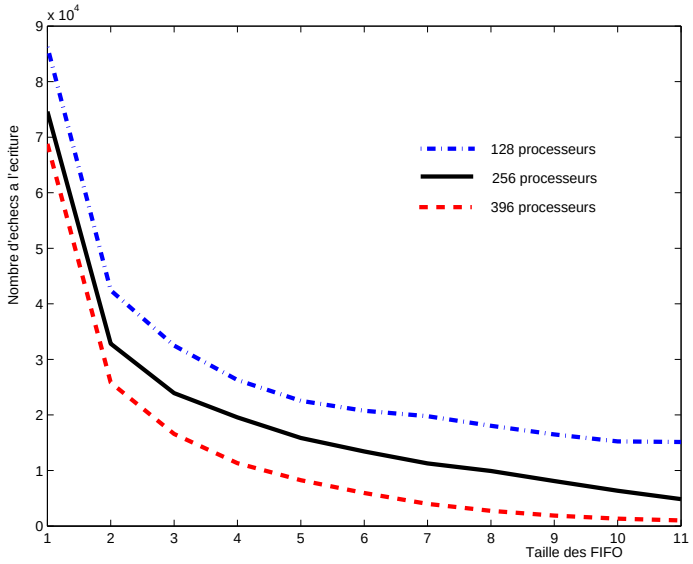
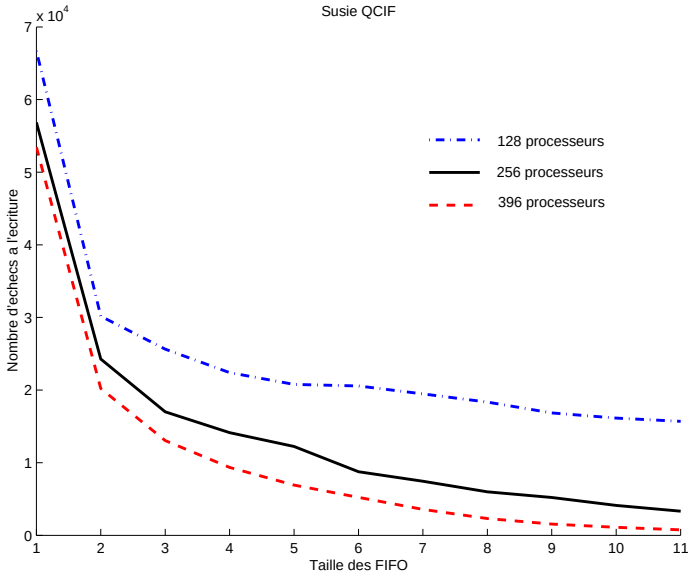


FIG. IV.39 – Influence de la granularité sur l'engorgement des FIFO (*Susie* QCIF).



(a) *Foreman* QCIF



(b) *Susie* QCIF

FIG. IV.40 – Influence de la granularité sur l'engorgement des FIFO : une faible taille des mémoires est suffisante, et plus la granularité est grosse, plus les bus de communication sont souvent engorgés.

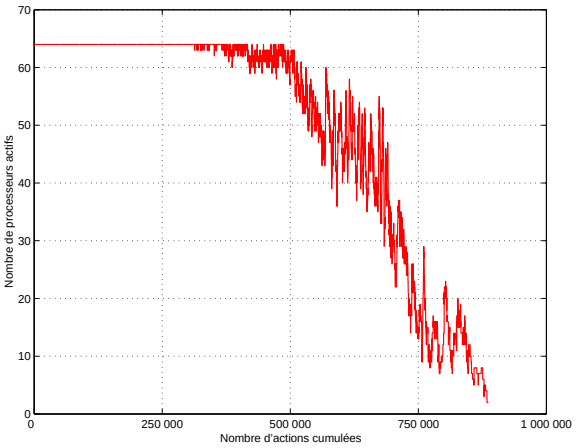


FIG. IV.41 – Activité du réseau 8×8 au cours du temps pour l'image *Foreman* QCIF : les charges sont mieux équilibrées (taux d'activité du réseau : 79%).

Activité du réseau. On observe une répartition des charges mieux équilibrée (figure IV.41) car tous les processeurs sont actifs pratiquement jusqu'au point de convergence : 79% des ressources (activité du réseau, définition IV.7) sont exploitées, comparativement au taux de 6.7% pour la fine granularité, figure IV.35.

Conclusion. Cette étude de granularité procure quelques éléments de réponse sur les spécificités de l'architecture à privilégier : un réseau de quelques centaines de processeurs élémentaires, communicant via des files d'attente de faible taille.

Malgré le surcoût des communications interprocesseurs dû au partitionnement LPGS (détermination des coordonnées du pixel cible, chaque propagation d'une donnée entre deux pixels voisins de l'image nécessite une communication), de meilleures performances sont obtenues grâce à une charge mieux équilibrée. Ce type de partitionnement est d'autant mieux adapté que l'algorithme est caractérisé par des traitements irréguliers et rapidement localisés (voir §IV.4.2).

Un partitionnement 2D-rectilinéaire (LSGP) de l'image impliquerait un fort déséquilibre des charges pour un réseau de plus de 100 processeurs : rapidement, seuls quelques processeurs calculeraient. Cependant, pour un réseau de faible dimension (uniquement quelques processeurs), un partitionnement LSGP est mieux approprié dû à l'engorgement important des files d'attente pour le partitionnement LPGS (figure IV.38). Malheureusement, une quantification des spécificités d'un partitionnement LSGP pour l'algorithme de *Hill-Climbing* réordonnancé n'est pas étudié, et fait partie des perspectives. Cet algorithme de segmentation présentant des caractéristiques similaires à celles des opérateurs morphologiques asynchrones, le lecteur peut trouver dans [Rob97] une étude comparative des deux types de partitionnement.

IV.5.2 Analyse de la vitesse

Dans cette partie, nous évaluons les temps de segmentation des différents algorithmes présentés ainsi que la répartition de l'activité au cours du temps.

Afin d'estimer les performances d'une implantation parallèle, nous utilisons les indicateurs classiques suivants :

Définition IV.8 (Le temps d'exécution). *Noté $T(P)$, il mesure le temps de convergence de l'algorithme sur une architecture de P processeurs.*

Ce temps comprend les temps de calculs internes de chaque processeur et les temps de communication.

Définition IV.9 (L'accélération). *Elle représente le rapport entre le temps de calcul pour un système monoprocesseur, et le temps de calcul pour un système multiprocesseurs.*

$$SP(P) = \frac{T(1)}{T(P)} \tag{IV.2}$$

IV.5.2.1 Architecture séquentielle

Afin d'estimer les temps de calcul pour un système embarqué séquentiel, nous avons effectué les approximations suivantes :
- Un processeur de type embarqué (ARM 9) a une capacité de calcul d'environ 200Mips (Million d'instructions par seconde).
- Chaque action nécessite environ trois[1] instructions machine.

La capacité de calcul estimée de l'ARM 9 est donc d'environ 67.10^6 actions par seconde, soit une latence de $15\,ns$ par action.

À partir des estimations sur les chaînes critiques présentées par les tables IV.5 et IV.7 page 116, les temps de segmentation estimés sont :
- **10 500** μs pour l'image *Foreman* SQCIF (95 Hz).
- **126 500** μs pour l'image *Foreman* QCIF (8 Hz).
- **63 000** μs pour l'image *Susie* QCIF (16 Hz).

Conclusion. À partir des estimations de complexité de l'algorithme séquentiel, un processeur embarqué de faible complexité ne peut segmenter une séquence vidéo QCIF à une fréquence de 25 Hz.

IV.5.2.2 Architecture parallèle à granularité la plus fine

Afin de simuler les temps calcul, chaque action effectuée par un processeur arrête le simulateur durant un délai tiré aléatoirement entre 6 et 8 ns (latence de traitement, §IV.3.4.2). Lorsque le processus de segmentation est terminé, la chaîne critique (sous forme de nombre d'actions cumulées) nous fournit le temps de segmentation.

Temps de segmentation. Grâce à l'environnement de simulation, nous estimons les temps de segmentation à :
- **5** μs pour l'image *Foreman* SQCIF (200 kHz) : $SP(88 * 72) =$ 2100.
- **15** μs pour l'image *Foreman* QCIF (66 kHz) : $SP(176 * 144) =$ 8400.
- **14** μs pour l'image *Susie* QCIF (71 kHz) : $SP(176 * 144) =$ 4500.

Les accélérations obtenues sont remarquables. Toutefois, nous retiendrons plus particulièrement l'ordre de grandeur 1 000 de ces accélérations plutôt que les valeurs

[1] Les lectures et écritures nécessitent 2 cycles, les affectations et combinaisons logiques 1 cycle, les tests 4 cycles en considérant les pénalités, et quelques cycles pour l'organisation interne du programme (boucles, instructions de saut, calculs d'adresses...)

exactes, car les capacités de l'architecture monoprocesseur sont probablement sous estimées : les phases de recherche des minima et d'étiquetage des plateaux (§IV.5.1.2 page 118) de l'algorithme séquentiel sont inoptimisées. En effet, l'accélération pour les images QCIF est plus forte que celle de l'image SQCIF, car les images QCIF étant plus simplifiées, celles-ci présentent des plateaux de grandes surfaces : le coût de recherche de minima et étiquetage est fort pour les plateaux minima, et également pour le coût de propagation des pixels extérieurs sur les plateaux non-minima.

Une accélération d'un facteur 1 000 est attendue pour un réseau de processeurs asynchrones à fine granularité.

Evolution de l'activité. À présent, observons l'impact du flux des données sur les instants d'activité des processeurs.

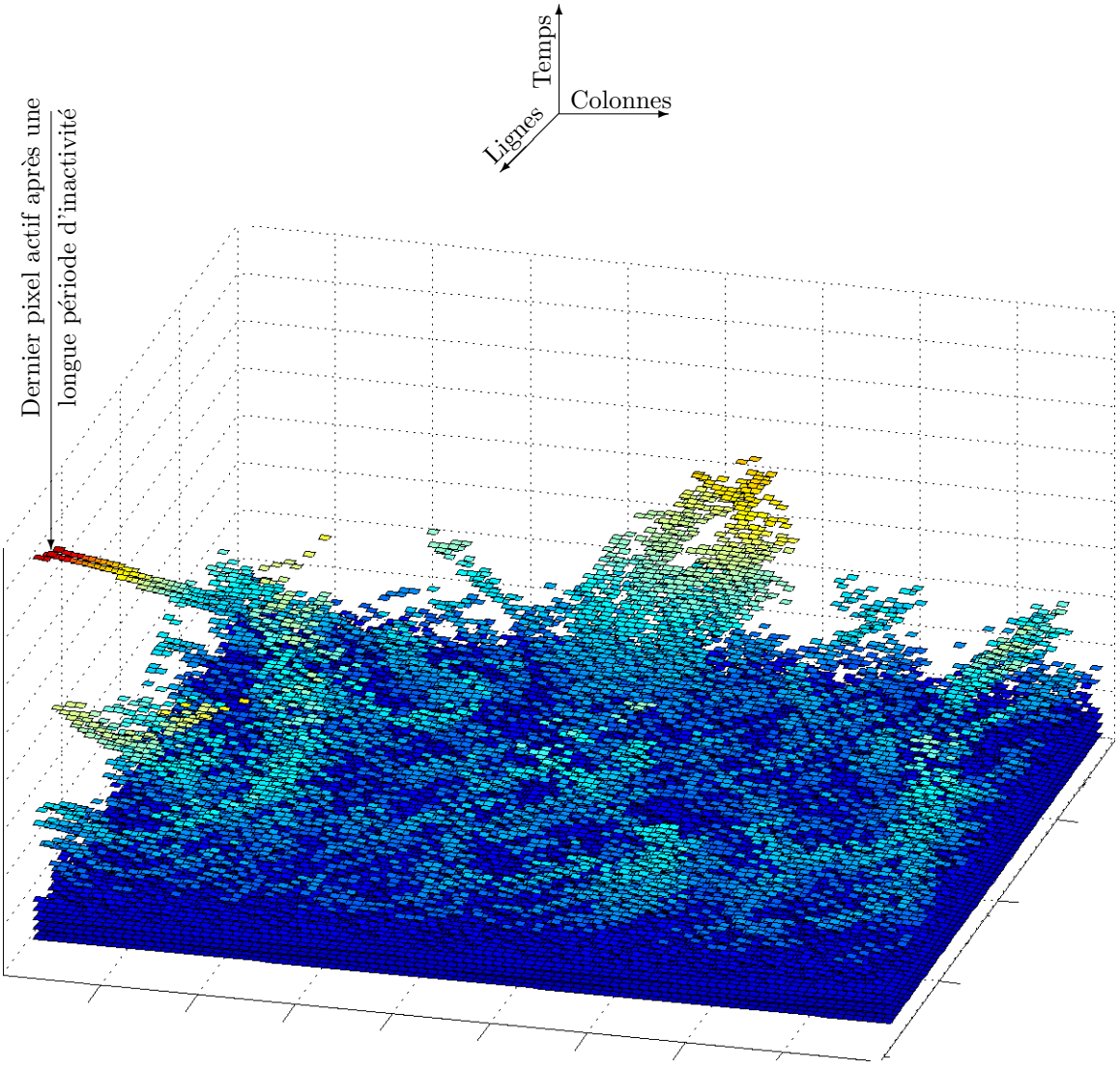


Fig. IV.42 – Instants d'activité des processeurs pour l'image *Foreman* SQCIF.

La figure IV.42 présente les instants où chaque processeur est actif. Ce graphe

représente les images de l'activité du réseau (similaires à la figure[1] IV.24 page 100) au cours du temps t (l'axe vertical).

Sur cette figure, à un instant donné, un processeur **actif** correspond à un pavé plein de base carrée et d'épaisseur la résolution temporelle, et à un **emplacement vide sinon**. La partie basse de ce graphe (pour de faibles valeurs de t) est pleine, car pratiquement tous les processeurs calculent initialement. Par contre, il est intéressant de remarquer que l'enveloppe du graphe est creuse : un faible nombre de processeurs calculent. La figure IV.43 illustre ces propos à l'aide d'une coupe suivant la quinzième ligne du graphe précédent.

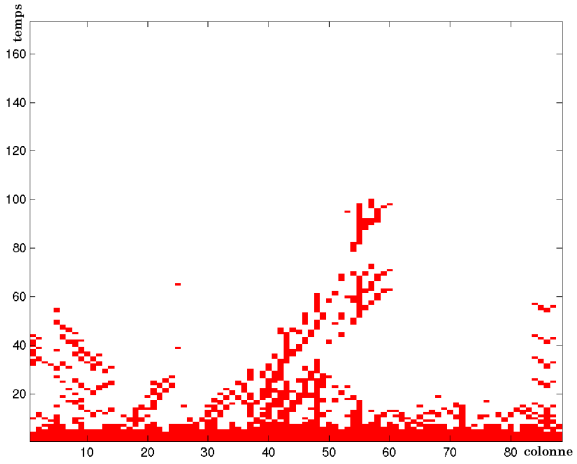


FIG. IV.43 – Coupe du graphe des instants d'activité suivant la quinzième ligne (coupe horizontale passant par le milieu du casque). Le volume situé en dessous de l'enveloppe du graphe est creux.

Par ailleurs, un processeur longtemps inactif peut se faire réveiller par une inondation venant d'un pixel lointain. Par exemple, le processeur de la dernière ligne deuxième colonne (repéré par une flèche dans la figure IV.42) est inactif durant un long moment (la colonne est vide), puis se fait réveiller et atteint le point de convergence. En comparant les figures IV.42 et IV.34, nous vérifions aisément que le processeur qui a la charge de calcul la plus importante (processeur $(17, 51)$), n'est pas le dernier processeur à converger, c'est-à-dire celui qui fixe la fin de la segmentation (processeur $(2, 72)$).

IV.5.2.3 Architecture parallèle à granularité intermédiaire

Dans cette partie, nous quantifions l'influence de la granularité et de la taille des FIFO, sur les temps de segmentation. Des éléments d'analyses quant aux profils des courbes obtenues (temps de traitement et accélération) sont présentés.

À des fins de comparaison entre les temps de calculs obtenus par les architectures précédemment présentées et celles à granularité intermédiaire, nous présentons une estimation obtenue pour une architecture adaptées à l'algorithme de *Hill-Climbing*.

[1]Pour des raisons de visibilités, un tel graphe pour l'image *Susie* QCIF ne peut pas être présenté.

Pour une architecture composée de 256 processeurs connectés par des FIFO de taille 3, nous estimons un temps de segmentation égal à :

- **1 415** μs pour l'image *Foreman* QCIF (700 Hz) : $SP(16 * 16) = 89.$
- **708** μs pour l'image *Susie* QCIF (1.4 kHz) : $SP(16 * 16) = 89.$

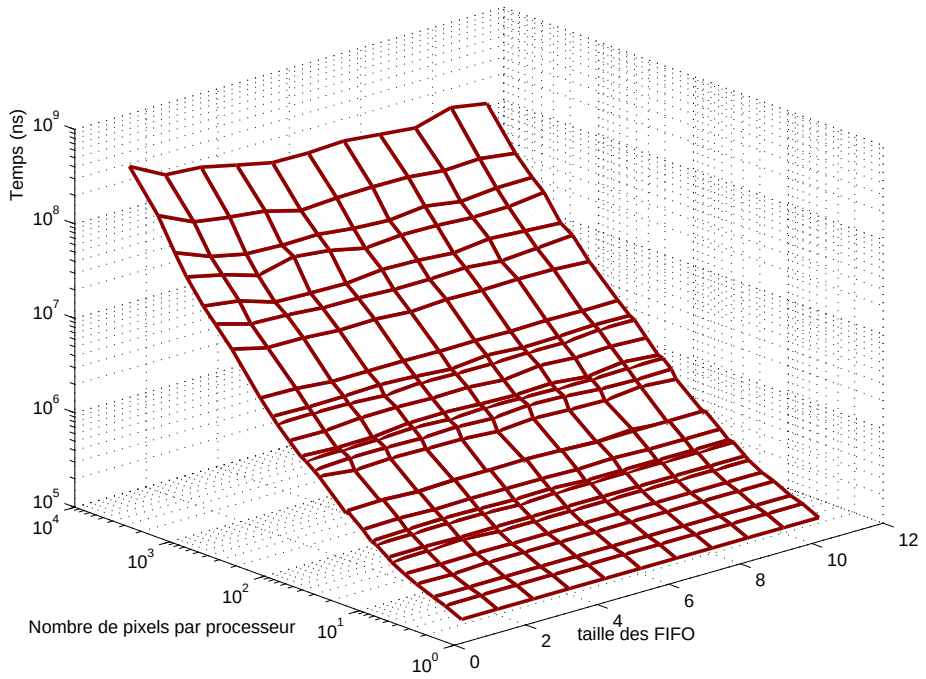


FIG. IV.44 – Influence de la granularité et de la taille des FIFO sur le temps de convergence (*Foreman* QCIF, échelle de temps logarithmique).

Analyse des temps de traitement. Les figures IV.44 à IV.47 présentent l'influence de la granularité et de la taille des FIFO sur les temps de convergence. Pour chaque image *Foreman* ou *Susie*, l'axe temporel logarithmique permet de visualiser plus facilement les temps de convergence pour des grains de faible taille, et l'axe temporel linéaire permet d'apprécier l'évolution exponentielle des temps de traitement en fonction de la grosseur des grains.

Deux principales caractéristiques s'en dégagent :

1. **L'évolution du temps de convergence :** Le temps de convergence croît de façon exponentielle en fonction du nombre de pixels alloués par processeurs. Cette caractéristique illustre que le partitionnement LPGS est plutôt adapté aux tailles moyennes de grain voire aux tailles faibles, plutôt que pour les grosses granularité (caractéristique opposée à celle du partitionnement LSGP où généralement, les grosses granularités donnent de meilleurs résultats du fait que les canaux sont moins engorgés).
2. **L'influence de la taille des files d'attente :** Cette analyse est un complément de celle concernant l'influence de la taille des FIFO sur le nombre total d'action (§IV.5.1.4 page 125). En effet, nous observons également pour les moyennes granularités une faible dépendance entre le temps de convergence et la taille des

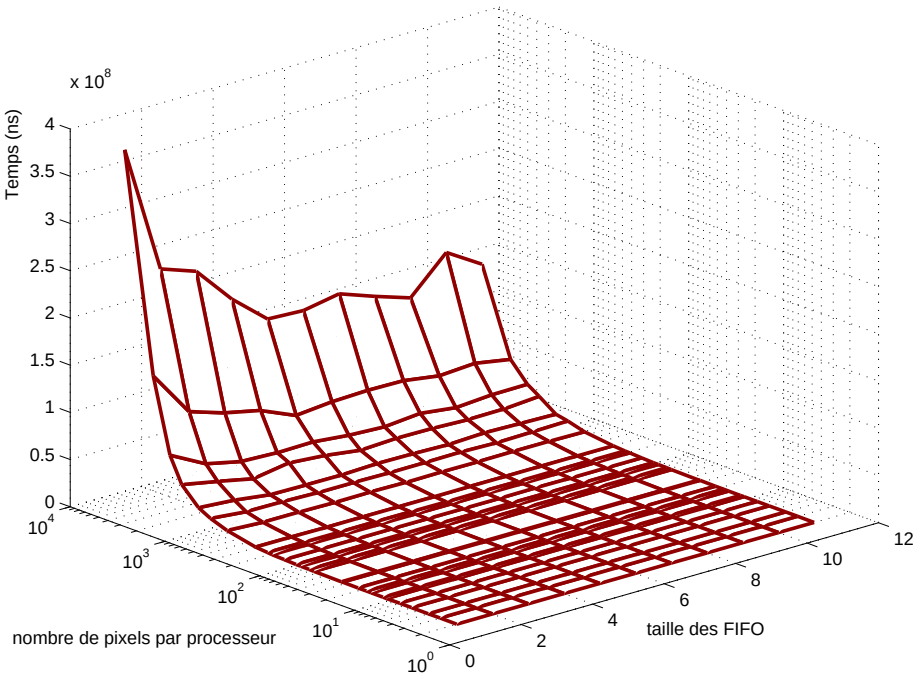


FIG. IV.45 – Influence de la granularité et de la taille des FIFO sur le temps de convergence (*Foreman* QCIF, échelle de temps linéaire).

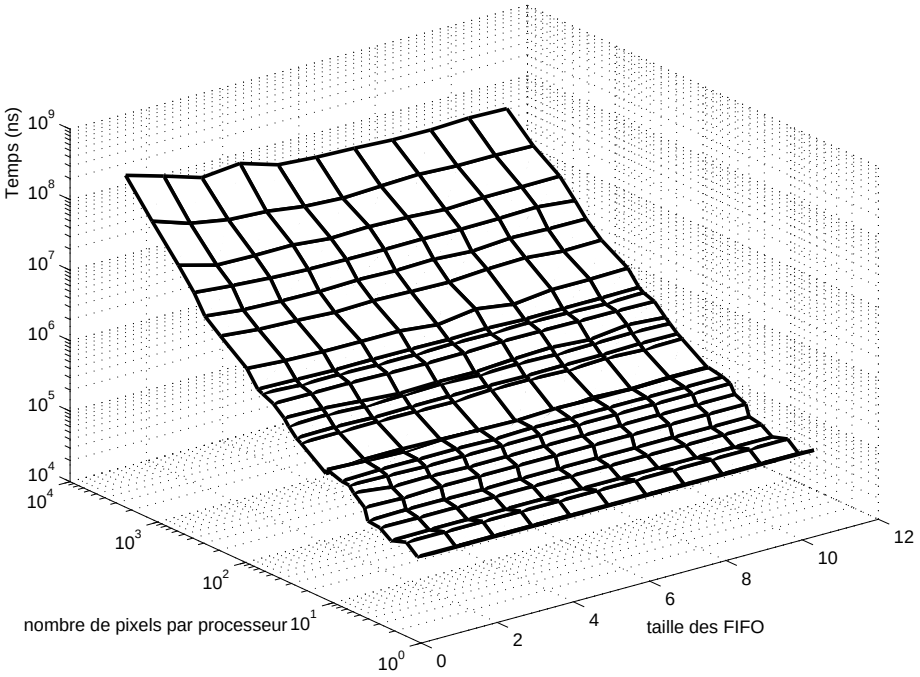


FIG. IV.46 – Influence de la granularité et de la taille des FIFO sur le temps de convergence (*Susie* QCIF, échelle de temps logarithmique).

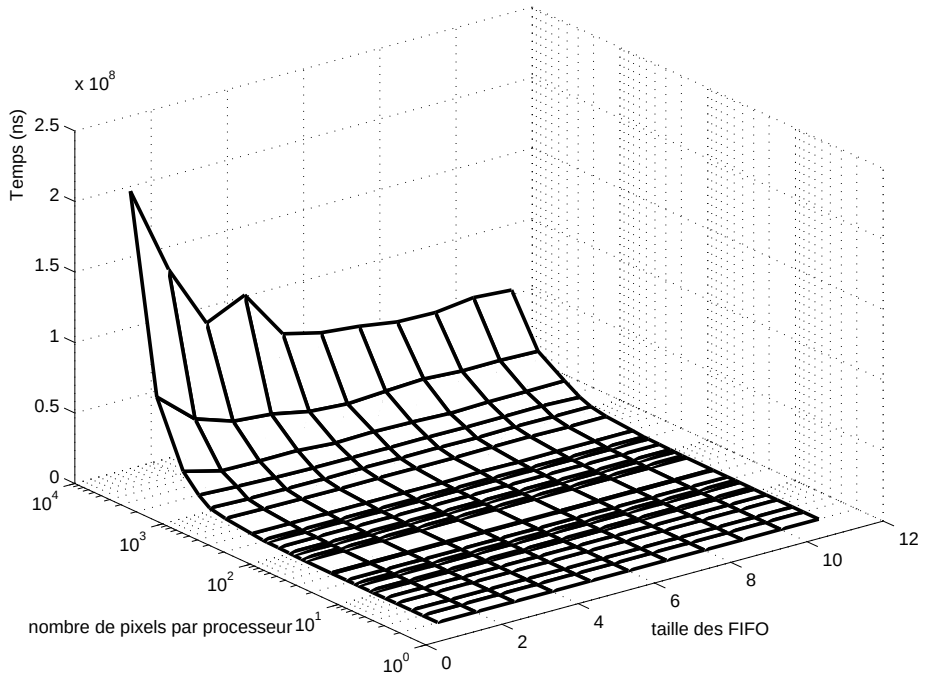


FIG. IV.47 – Influence de la granularité et de la taille des FIFO sur le temps de convergence (*Susie* QCIF, échelle de temps linéaire).

FIFO : **une architecture composée de canaux de mémoires de faible taille est suffisant.**

Cependant, lorsque le nombre de processeurs devient réduit (de l'ordre de 4 ou 8 processeurs), le partitionnement LPGS présente un fort surcoût de communications. Nous observons alors une influence plus marquée de la taille des FIFO.

Le partitionnement LSGP présenterait probablement, pour une taille équivalente de grains, de meilleures performances.

Analyse de l'accélération. Ce paragraphe montre l'évolution de l'accélération en fonction de la granularité (le nombre de processeurs).

Les courbes des figures IV.48 et IV.49 compare l'accélération de notre architecture (en pointillés) pour les images *Foreman* et *Susie*, avec celle d'une architecture parfaite (accélération constante de pente un, trait continu). Cette courbe montre que l'architecture fournissant la meilleure accélération est celle composée de 200 processeurs environ (point le plus proche de l'accélération "parfaite"). L'influence de la taille des FIFO est surtout marquée pour les grosses granularités. Nous observons, aux deux extrémités de cette courbe, les limites du partitionnement LPGS. Pour les granularités les plus grosses, l'accélération est dégradée par le surcoût induit par les communications et l'engorgement des canaux. Pour les granularités les plus fines, la dégradation de l'accélération est due au fort déséquilibre des charges.

Remarque: Pour l'image Susie, l'accélération dépasse l'accélération théorique. Pour les figures IV.48 et IV.49, le temps de référence $T(1)$ pour un système séquentiel est estimé en simulant le même algorithme de granularité intermédiaire sur un processeur

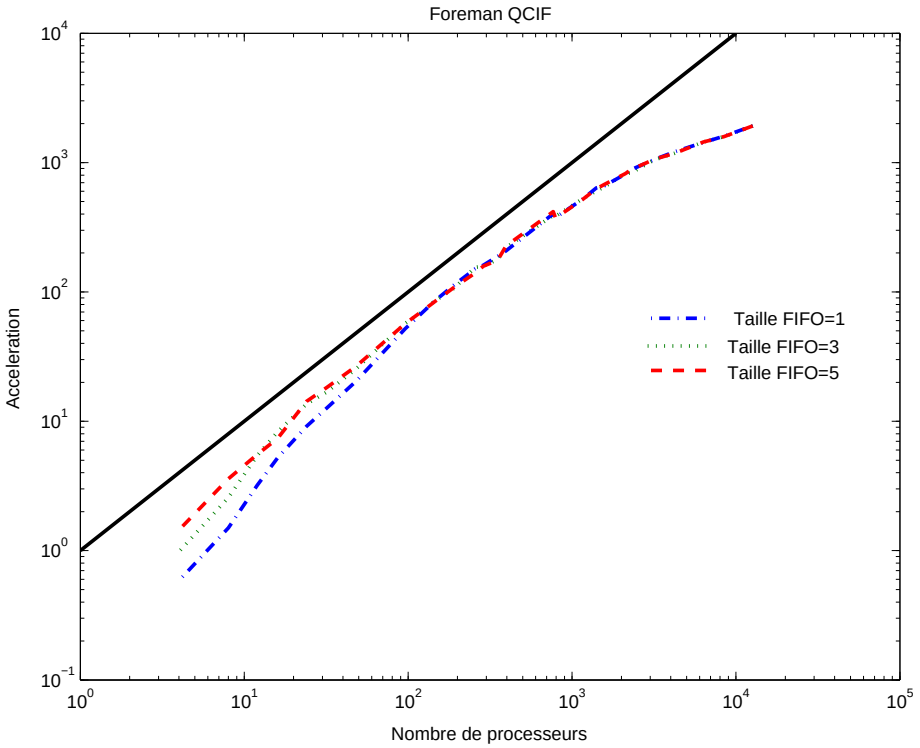


FIG. IV.48 – Influence de la taille des FIFO sur l'accélération (*Foreman* QCIF).

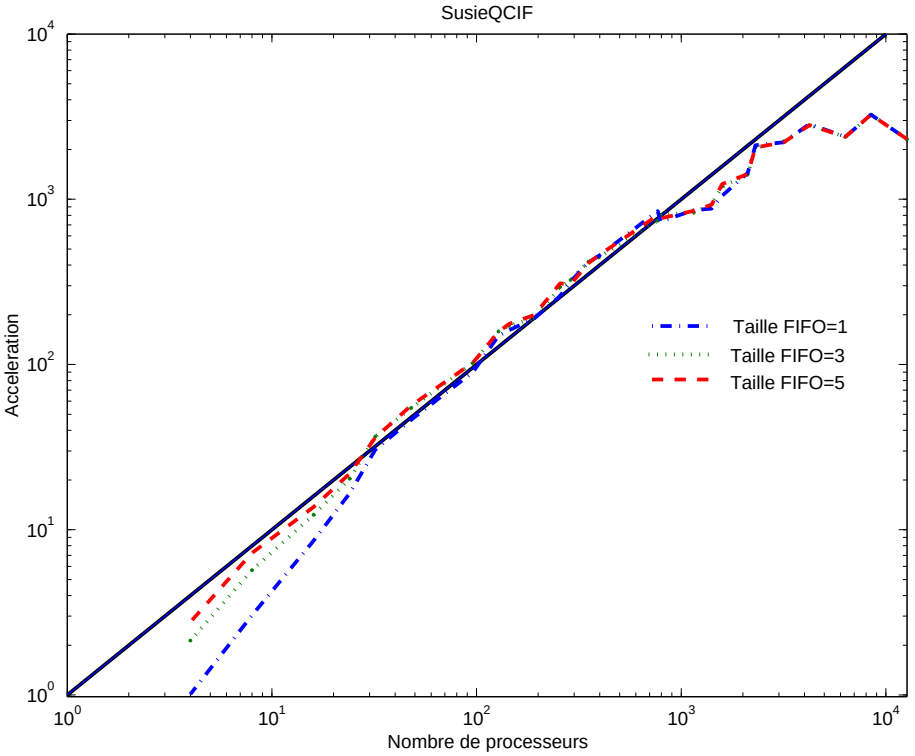


FIG. IV.49 – Influence de la taille des FIFO sur l'accélération (*Susie* QCIF).

rebouclé sur lui-même (un seul processeur pour l'image). Dans ce cas particulier, l'activité mesurée sous-estime le temps de calcul d'un système séquentiel puisque, entre autres, le calcul des adresses des pixels est inutile.

Pour une estimation plus fine de cette accélération, le modèle d'exécution de l'algorithme de *Hill-Climbing* doit être ajusté. Toutefois, l'analyse de cette courbe nous permet de conclure qu'un réseau de quelques centaines de processeurs est la granularité la mieux adaptée au partitionnement LPGS pour l'algorithme de *Hill-CLimbing* réordonnancé (image QCIF).

IV.5.3 Conclusion sur l'analyse des résultats de simulation

Les estimations de complexité algorithmique (exprimée sous formes d'actions et d'itérations) et d'activité débouchent sur une analyse quantitative des modèles simulés. Les architectures étudiées se déclinent en deux catégories : une architecture séquentielle et un large spectre d'architectures allant du grain le plus fin, au grain le plus gros.

La première, **séquentielle**, est une base de travail. Elle nous donne une référence à laquelle on compare les performances de l'algorithme de segmentation réordonnancé implanté sur une architecture parallèle.

L'architecture séquentielle n'est pas en adéquation avec un terminal multimedia portable : même avec l'utilisation de files d'attente (processus d'inondation), la complexité en termes d'actions est élevée (aspect faible consommation électrique *a priori* non respecté), et le temps réel n'est pas respecté (fréquence de segmentation inférieure à 16 images par seconde).

La seconde, **parallèle**, est le cœur de ce chapitre. L'analyse d'activité montre que l'architecture à grain le plus fin permet de réduire de 30% environ le nombre d'actions, mais présente un fort déséquilibre des charges (le taux d'activité est inférieur à 7%).

En perspective d'implantation physique, une granularité augmentée montre une meilleure répartition (utilisation de 79% des ressources) et ce, grâce à un **partitionnement LPGS** : tous les processeurs sont actifs, même pour des calculs fortement localisés. Cependant, les granularités extrêmes sont à éviter : le choix du partitionnement LPGS induit de forts taux de communication dans les configurations à gros grains, et les grains les plus fins engendrent un déséquilibre des charges dû à la localité des traitements de l'algorithme.

En termes de vitesse, l'architecture à granularité la plus fine montre des performances remarquables : plus de 66 000 segmentations d'images QCIF par seconde, soit **un facteur d'accélération de l'ordre de 1 000** par rapport à un système séquentiel. Pour la granularité intermédiaire, nous montrons que le meilleur compromis dimension/coût est un réseau de 256 processeurs connectés en tore-2D par des FIFO de taille 3. Une fréquence de l'ordre de 700 segmentations d'images par seconde est alors obtenue, soit **un facteur d'accélération d'environ 89.**

Nous montrons que les temps de traitement suivent de façon exponentielle la taille des grains : une architecture en adéquation avec un terminal portable doit donc être constitué de **quelques centaines de processeurs simples**. De plus,

pour l'ensemble du spectre des architectures étudiées, une faible taille des FIFO est suffisante : le réseau d'interconnexion est simple.

Algorithme	Séquentiel	À fine granularité	128 processeurs (FIFO=3)	256 processeurs (FIFO=3)
Foreman SQCIF	10 500 μs	5 μs		
Foreman QCIF	126 500 μs	15 μs	2 981 μs	1 415 μs
Susie QCIF	63 000 μs	14 μs	1 313 μs	708 μs

Tab. IV.14 – Étude comparative des temps de segmentation.

La table IV.14 récapitule l'ensemble des temps de segmentation estimés. Nous observons la forte dépendance des temps de calcul aux données, mais comme précisé auparavant, cette dépendance est due principalement à la simplification de l'image qui crée de très grand plateaux minima et non-minima. Par exemple, pour un même format, l'image *Foreman* est deux fois plus longue à segmenter (présence de plateaux non-minima formant une spirale) que l'image *Susie*.

IV.6 Conclusion

Ce chapitre présente une des parties importantes de ces travaux : **la validation de l'algorithme et l'exploration d'architectures parallèles**.

L'étude générale sur les architectures parallèles (§IV.1) nous oriente vers la classe des systèmes SPMD (*Single Program stream Multiple Data stream*) sur une topologie de processeurs faiblement couplés en forme de grille-2D ou de tore-2D.

En perspective de faisabilité d'intégration, différentes tailles de grains sont étudiés. Grâce à la généricité de l'environnement de simulation basé sur la bibliothèque SystemC™, les granularités allant d'un processeur pour l'image jusqu'à un processeur par pixel sont simulés et analysés.

Le raffinement de cette étude (§IV.2) aboutit sur l'utilisation de **communications non-bloquantes**, où les canaux à **mémoire non-nulle** interconnectent des **processeurs asynchrones**. La priorité des lectures sur les écritures permet de réduire d'environ 10% à 20% la chaîne critique des traitements. Suivant la taille du grain employé, différentes stratégies sont adoptées : conservation des données les plus anciennes pour les tailles intermédiaires, ou conservation de la dernière donnée pour la plus fine.

Les performances des architectures à tailles extrêmes de grain sont limitées par le surcoût induit par les communications pour les plus grosses (caractéristique du partitionnement LPGS), et par le déséquilibre des charges pour les plus fines (seulement 5 à 7% des ressources sont utilisées).

Le point de convergence ne pouvant être déterminé, il est détecté par combinaison globale des états d'activité des processeurs. Afin d'accroître la robustesse de la détection, un filtre passe-bas est utilisé pour supprimer les *glitch* de fausse détection.

Dans des perspectives de faisabilité de réalisation et d'évaluation du spectre des architectures parallèles, les granularités allant de la plus fine (un processeur par pixel) à la plus grosse (2 processeurs par image) sont étudiés. Afin d'exploiter au mieux l'asynchronisme algorithmique, le partitionnement LPGS (Localement Parallèle Globalement Séquentiel) est utilisé dans le cadre de granularités intermédiaires.

Nous pensons que de telles architectures sont en adéquation avec l'algorithme de *Hill-Climbing* réordonnancé.

Afin de confirmer la véracité de cette affirmation, un environnement de simulation capable de modéliser le comportement fortement désynchronisé du réseau est établi (§IV.3). De par sa généricité, la bibliothèque de prototypage de haut niveau d'abstraction **SystemC™ permet une exploration aisée d'architectures**.

Ce simulateur étant initialement conçu pour modéliser des architectures synchrones, un artifice est utilisé pour modéliser le comportement asynchrone des processeurs et pour réduire le nombre de fils d'exécution de l'ordonnanceur : une machine à état fini cadencée où la période de l'horloge correspond à la résolution temporelle du simulateur.

L'environnement de simulation une fois établi, le fonctionnement correct du couple algorithme-architecture et la justesse de la segmentation sont vérifiés (§IV.4). Une pré-étude montre qu'une répartition aléatoire des étiquettes supprime les directions de propagation privilégiées sur les plateaux, accroît la qualité du résultat de segmentation, et réduit les charges de calculs.

Enfin, en vue d'une évaluation de la faisabilité d'intégration de ces architectures dans un terminal portable, les résultats de simulation sont analysés (§IV.5). Différents modèles d'exécution et de communication sont explorés et aboutissent, suivant la granularité du réseau recherché, à un compromis entre temps de convergence et coût de calculs.

Les résultats de complexité de l'algorithme séquentiel montrent que **l'implantation d'un codeur vidéo orienté objet nécessite la conception de briques algorithmiques implantées par des circuits dédiés**. Un processeur généraliste ne pourrait pas mutuellement respecter les contraintes de temps de traitement et de consommation. Bien qu'il soit facilement reconfigurable, nous n'avons pas retenu cette solution.

L'étude d'un **ASIC capable d'implanter l'algorithme de segmentation** par ascension de colline (*Hill-Climbing*) se justifie par le **respect des contraintes** de consommation et de puissance de calcul. Ces deux contraintes ne pouvant être optimisées séparément, cette analyse se décline en l'évaluation des charges de calcul et des temps de traitement pour l'ensemble du spectre des architectures. En cherchant le meilleur compromis parmi ces différents résultats, il est alors possible de déterminer l'architecture la mieux adaptée à ses contraintes.

Le couple algorithme-architecture à présent validé, nous devons vérifier si un tel circuit peut s'intégrer dans un terminal multimedia portable. C'est pourquoi par une approche de conception microélectronique, le prochain chapitre étudie la faisabilité de l'intégration de l'algorithme proposé, sa consommation électrique, et sa surface.

Chapitre V

Modélisation pour la conception : un aspect implantation

Nous avons validé le couple algorithme-architecture, et analysé l'activité et la vitesse d'un large spectre d'architectures allant de la granularité la plus fine à la plus grosse (chapitre IV). L'asynchronisme fonctionnelle est bien mis au profit de l'asynchronisme algorithmique, l'architecture est en adéquation avec l'algorithme et *vice versa*.

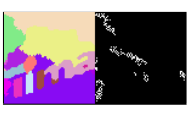
À présent, nous étudions la faisabilité d'intégration de ce couple dans un terminal portable. Nous cherchons à évaluer les caractéristiques de l'architecture en termes d'exactitude des partitions générées, faisabilité de conception, surface, temps de traitement et consommation. Souhaitant exploiter au mieux l'asynchronisme fonctionnelle et les outils de conception existants, des architectures à granularité la plus fine sont conçues avec une technologie synchrone ou asynchrone.

Cette étude clôt l'acte de recherche dédiée à l'étude de faisabilité d'intégration d'une chaîne de codage vidéo dans un terminal multimedia portable. Nous terminerons ce manuscrit par un bilan, résultats et perspectives au chapitre VI.

Les circuits asynchrones sont des circuits électroniques dépourvus de signal d'horloge. Le contrôle global inhérent au circuit synchrone est naturellement distribué sur l'ensemble des modules constituant le circuit asynchrone. Ils sont également désignés sous la terminologie ***data driven*** ou **flot de donnée** car seules les données ordonnancent les différentes phases de calcul. Cependant la maturité des outils de synthèse étant insuffisante pour un circuit d'une telle ampleur, nous avons également conçu un réseau synchrone afin d'évaluer sa surface, ses temps de traitement et sa consommation.

Après avoir décrit les caractéristiques de l'architecture indépendamment du style de conception (§V.1), nous présentons la modélisation bas niveau (VHDL-RTL[1]) d'un réseau de **processeurs synchrones** (§V.2). En vue d'une implantation faiblement consommante, nous envisageons la conception d'un réseau de processeurs asynchrones. Une fois l'environnement de conception introduit (§V.3), nous présentons une première approche de la modélisation CHP (*Communicating Hardware Processes*) d'un pixel asynchrone (§V.4).

[1] *Register Transfert Level*.



V.1 Architecture du réseau

Lors de cette étude, nous avons tout d'abord souhaité valider l'aspect implantation des traitements désynchronisés. Une architecture au niveau de granularité le plus fin est alors étudiée. Les processeurs situés au bord du réseau ne sont pas 4-connexes, nous montrons comment nous rétablissons la régularité du réseau. Comme la fin ne peut-être prévue, elle est détectée par une combinaison globale de l'état d'activité des processeurs.

Connectivité des processeurs. Le réseau, une grille 4-connexe, est composée de processeurs n'ayant pas tous le même degré de connectivité. Ceci est préjudiciable lors de la phase de synthèse car un tel réseau est généré par l'aboutement de processeurs identiques sous forme d'une matrice. Afin de rétablir la régularité de l'architecture, une alternative consiste à instancier un cadre de modules supplémentaires ayant un comportement neutre pour l'algorithme. Ces modules sont d'une complexité bien plus faible que celle des processeurs élémentaires (uniquement réduits à des fils liant des ports, ou connectés à un état logique constant).

Détection de fin. La détection de fin est réalisée par une combinaison logique de l'ensemble des états d'activité des processeurs. Pour des raisons de routage et de consommation, cette combinaison globale est effectuée par **un réseau distribué de portes** OU à deux entrées (figure V.1).

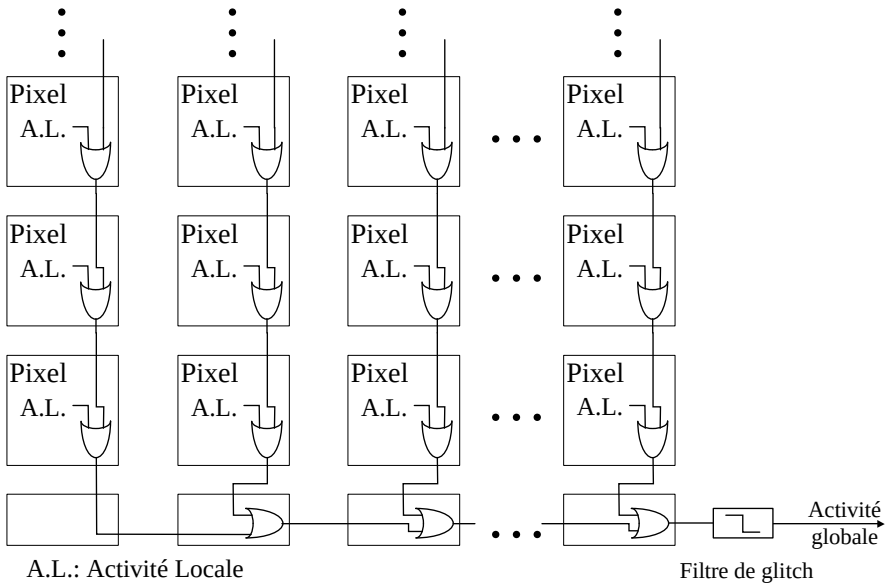


FIG. V.1 – Détection de fin : combinaison logique des états d'activité par un réseau distribué de portes OU.

Dans une première approche et pour des raisons de facilité de conception, nous avons instancié autant de porte OU qu'il y a de processeurs. Bien sûr, une structure en "log" permettrait de réduire le coût d'implantation. Cependant sous cette forme le réseau est irrégulier, l'instanciation des processeurs et des portes OU sont dissociés, donc légèrement plus complexe à synthétiser.

L'état d'activité global du réseau est obtenu par un calcul purement combinatoire (inexistence de signal d'horloge).

V.2 Étude du pixel synchrone

Le but de cette étude est de valider la faisabilité d'implantation microélectronique d'un réseau de processeurs dédié à l'algorithme de *Hill-Climbing*. La seconde idée consiste à effectuer une première estimation des caractéristiques d'un réseau de pixels synchrones : surface, temps de traitement et consommation.

V.2.1 Description générale

Le pixel synchrone est décrit par une machine à état fini de façon similaire à celle présentée par la figure IV.14 page 91. Toutes les variables internes de la machine sont mémorisées par un unique registre cadencé. De cette façon, aucune *Latch* au comportement difficile à simuler est générée lors de la synthèse du pixel.

Chaque pixel est composé d'un module combinatoire comprenant les signaux d'entrée, de sortie et de contrôle, et d'un registre cadencé par l'horloge et initialisé par un signal de *reset*.

Ici, les processeurs ne sont pas instanciés avec l'altitude de leur pixel associé, l'image est donc chargée dans le réseau. Dans un premier temps, nous souhaitons que quatre pixels par cycle d'horloge soient chargés. Le réseau est partitionné en quatre régions suivant un partitionnement régulier 1D-rectilinéaire (figure II.4a page 11) où pour chaque région, les données se propagent suivant un chemin hamiltonien en forme de S (figure A.2 page 180).

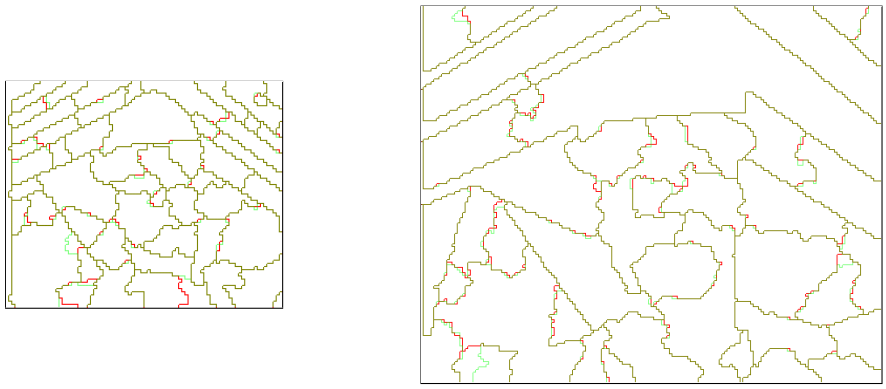
Durant le processus de segmentation, tous les pixels à l'état INIT ou MP consultent simultanément leurs ports suivant le sens horaire, et tous ceux à l'état NM consultent uniquement leur voisin d'inondation. Ce choix est motivé par la limitation de la complexité du processeur élémentaire. L'utilisation de quatre unités arithmétiques, principalement des opérateurs de comparaison 13 ou 15 bits, permettrait de réduire le temps de convergence et de consulter en parallèle tous les processeurs voisins. Cependant, la surface d'un processeur serait alors plus élevée et les ressources ne seraient que partiellement exploitées car seulement environ 25%[1] des processeurs sont à l'état MP et exploiteraient ces unités arithmétiques (tout pixel à l'état NM n'effectue que des copies des données issues de **l'unique** voisin d'inondation).

V.2.2 Validation par simulation

L'environnement de simulation est issu des outils industriels de conception de circuits synchrones. De façon similaire à la section IV.3.3 page 89, le réseau instancié par le simulateur est une matrice de modules interconnectés par des fils (*signals*) suivant un voisinage 4-connexe.

[1] Estimation effectuée sur les trois images test (figures IV.16 à IV.18 page 94).

La conception d'un pixel synchrone montre qu'un circuit microélectronique peut correctement implanter le modèle d'exécution établit lors de l'étude présentée à la section IV.2 page 80. Afin de limiter la complexité du circuit, la priorité des lectures sur les écritures n'est pas implantée, c'est-à-dire que chaque lecture d'une donnée est suivie par une écriture du résultat intermédiaire vers les processeurs voisins.



a) *Foreman* SQCIF a) *Foreman* QCIF

FIG. V.2 – Comparaison des segments obtenus avec un modèle VHDL synchrone (en rouge) et le modèle asynchrone (en vert).

La figure V.2 compare les segments obtenus avec le modèle bas niveau synchrone (programmation en VHDL) et ceux obtenus avec le modèle de haut niveau d'abstraction asynchrone (modélisation SystemC^{TM}). Faute de temps, un réseau de processeurs asynchrones n'a pas pu être instancié et simulé. C'est pourquoi nous reprenons ici les résultats obtenus lors de la simulation SystemC^{TM} (figure IV.29 page 104).

Ces simulations montrent que l'architecture microélectronique segmente correctement les images de test. Les variations sont de l'ordre du pixel, sauf bien sûr dans le cas où les points de bifurcation attribuent une étiquette différente à l'ensemble de la boutonnière. Toutefois, ce comportement ne met pas en défaut le couple algorithme-architecture.

	Chargement de l'image (nombre de cycles)	Segmentation (nombre de cycles)	Total
Foreman SQCIF	1 584	274	1 858
Foreman QCIF	6 336	407	6 743

TAB. V.1 – Temps de calcul exprimé en nombre de cycles d'horloge.

La table V.1 présente les résultats de simulation obtenus pour l'image *Foreman* aux formats SQCIF et QCIF. La propriété remarquable de ce circuit est la faible longueur de sa chaîne critique : **quelques centaines de cycles suffisent** pour segmenter l'image. Nous verrons lors de la synthèse d'un processeur (§V.2.3) que le temps minimal de traversée du processeur est de l'ordre de 30 ns(période de l'horloge), soit une fréquence de segmentation de 121,6 kHz et 81,9 kHz pour respectivement des images SQCIF et QCIF.

Comparativement à un algorithme de segmentation classique, nous rappelons que les 274 et 407 cycles représentent l'ensemble de la chaîne de segmentation : détection

des minima, étiquetage et détermination des lignes de plus grande pente.

Un cycle d'horloge correspond à une itération (définition IV.5 page 110) car une lecture, calculs et écriture sont effectués durant cet intervalle de temps. Il est délicat de comparer les estimations de la table V.1 avec celle de la table IV.12 page 122 car cette dernière exprime la chaîne critique en nombre d'actions. De plus, une itération représente un nombre d'actions fortement variable suivant l'état du pixel (MP ou NM) et le relief topographique.

Comme seulement quatre pixels sont chargés simultanément, le temps de chargement de l'image est égal à $(C * L)/4$ cycles d'horloge. Plus de 80% du temps et de l'énergie sont consacrés au chargement de l'image. Cette proportion pénalise l'efficacité de cette architecture, d'autres alternatives sont donc à envisager.

Par exemple, l'entrée des données peut s'effectuer au niveau de chaque colonne (88 cycles pour le chargement est alors suffisant pour une image SQCIF), ou au sein même de chaque pixel. Cette dernière alternative rejoint le principe **des rétines artificielles** : chaque pixel est doté de son propre système d'acquisition [BZD93] [BN97] [Sic99].

L'architecture étant fonctionnellement validée, les sections suivantes présentent une étude de plus bas niveau : la complexité microélectronique et la consommation.

V.2.3 Synthèse et complexité microélectronique

À partir de la synthèse d'un pixel et d'un réseau de faibles dimensions, cette section présente les estimations des caractéristiques technologiques du réseau (surface, nombre de portes et temps de cycle). Ces estimations sont basées sur les spécificités de la technologie HCMOS-8, soit 0,18μm de longueur de canal pour un transistor.

Pour chaque pixel, les rapports de synthèse montrent une complexité de l'ordre de 419 cellules, dont 84 bascules présentant les points mémoire de chaque pixel, soit environ 900 équivalent porte[1].

Sans aucune approche de conception hiérarchique, le réseau de taille maximale synthétisable par notre station de travail[2] est de dimensions 16×16. L'ensemble est composé de 360,8 kportes pour un fréquence maximale de l'horloge de 34 MHz (environ 30 ns par cycle).

Curieusement, nous observons un accroissement de 56% de la complexité entre la synthèse d'un pixel unique (900 portes) et d'un réseau de pixels (16*16*1409 portes). Toutefois, la synthèse d'un réseau de taille QCIF ou SQCIF doit être réalisée par aboutement de pixels : un processeur est optimisé et instancié sous forme d'une matrice (conception hiérarchique).

Par extrapolation, nous estimons qu'un réseau de dimensions 88×72 est composé d'environ 5 702 kportes. Chaque pixel occuperait environ 17 300 μm^2 (surface obtenue

[1] Rapport de surfaces entre le pixel et une porte NAND composée de quatre transistors.
[2] Sun Ultra-Sparc biprocesseurs, 2 Go de mémoire.

à partir de la synthèse d'un réseau 16 × 16), soit un circuit d'un centimètre carré environ pour un réseau SQCIF.

Par ailleurs pour une image SQCIF 4-connexe, la connectique de chaque processeur élémentaire est composée de 110 fils :
- 3 de contrôle : *clock*, *reset*, chargement-relaxation ;
- 5*8 + 5*13 de données : altitudes codées sur huit bits et les étiquettes sur 13 bits[1], et ce pour quatre entrées et une sortie ;
- 2 d'activité : un bit issu du voisin situé au nord et un bit de sortie vers son voisin du sud.

Cet inventaire montre que plus l'image est grande et la connexité est élevée, plus la surface de chaque pixel est importante car les étiquettes sont codées sur $n_{bit} = \lceil \log_2(C * L) \rceil$ et chaque degré de connexité supplémentaire nécessite $8 + n_{bit}$ fils.

L'estimation de surface précédente montre que pour une 4-connexité, le circuit en vue d'une implantation dans un terminal portable est trop coûteuse. Donc implanter un réseau 6 ou 8 connexe présenterait un mauvais rapport qualité de segmentation/coût d'implantation.

V.2.4 Estimation de consommation

Dans cette partie, nous présentons les estimations de l'énergie consommée par un seul pixel obtenues grâce à l'outil *Analog Artist* de Cadence®. Le voisinage de ce pixel est simulé par un vecteur de test qui stimule ses ports d'entrée. Les mesures effectuées sont exprimées sous forme d'énergie consommée (exprimée en Joule) par cycle d'horloge. L'expression suivante est utilisée :

$$\tilde{E} = V_{alim} * \int_{t_0}^{t_0+\Delta t} |i(t)| dt \tag{V.1}$$

avec $V_{alim} = 1.8\,\mathrm{V}$.

Pour une fréquence d'horloge fixée, l'évaluation de la puissance consommée est déterminée par :

$$\tilde{P} = \frac{\tilde{E}}{\Delta t} \tag{V.2}$$

De même, nous avons utilisé pour ces mesures la technologie HCMOS-8.

V.2.4.1 Vecteurs de test

Afin de tester le comportement électrique du pixel, nous stimulons ses ports d'entrée par un vecteur de test. Dans cette étude, nous en avons utilisé deux où le pixel est soit un minimum local, soit non-minimum sur un relief pentu.

Le chronogramme de base (figure V.3) est composé des cinq phases suivantes :

1. `RESET` : cette phase correspond à l'intervalle de temps où le signal *reset* est à 1 (actif à l'état haut) ;

[1] $13 = \lceil \log_2(88 * 72) \rceil$.

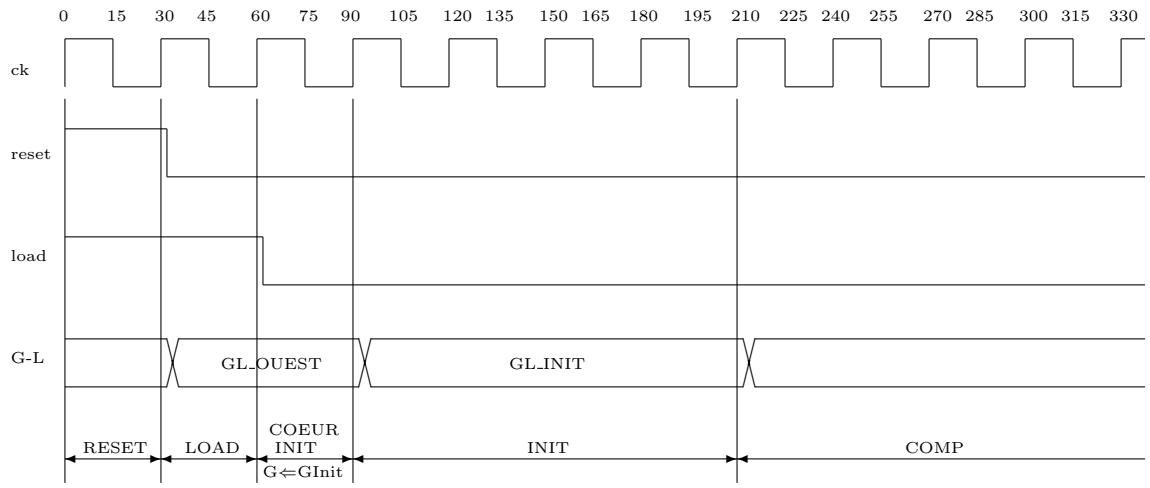


FIG. V.3 – Chronogramme de chaque vecteur de test.

2. `LOAD` : lorsque ce signal de contrôle est à l'état haut, le pixel propage les données d'entrée (chargement de l'image). Cette phase ne dure qu'un cycle d'horloge car seul un pixel est testé ;
3. `COEUR INIT` : durant cette phase (un cycle d'horloge) le cœur du pixel transite du mode de chargement de l'image vers le mode de segmentation. De plus, il mémorise l'étiquette et altitude lui correspondant ;
4. `INIT` : cette phase dure exactement quatre cycles d'horloge. Le pixel détermine sa topographie en comparant son altitude avec celles de ses quatre voisins ;
5. `COMP` : cette dernière phase correspond à la relaxation du réseau où les étiquettes et altitudes se propagent dans le relief.

Les rapports de synthèse indiquent une période minimale d'horloge de 30 ns. Les vecteurs utilisés sont alors fixés en accord avec cette caractéristique. La phase de transition de toute nouvelle donnée s'effectue 200 ps après un front d'horloge montant.

Pixel	(étiquette, altitude)
Courant	`(1AAA,0A)`
Nord	`(10F0,17)`
Est	`(007F,21)`
Sud	`(0001,0F)`
Ouest	`(1FFFF,AA)`

(a) Pixel minimum local.

Pixel	(étiquette, altitude)
Courant	`(0AAA,0A)`
Nord	`(10F0,17)`
Est	`(007F,21)`
Sud	`(0001,07)`
Ouest 1	`(1FFF,02)`
Ouest 2	`(0FFF,02)`

(b) Pixel sur un relief pentu.

FIG. V.4 – Vecteurs de test utilisés (valeurs en hexadécimal).

La figure V.4 présente les *stimuli* pour deux pixels de topographie différente. Les étiquettes et altitudes sont affichées en base hexadécimale. La donnée correspondant à "Ouest 2" dans la figure V.4b simule un étiquetage multiple du pixel.

V.2.4.2 Résultats des mesures

L'estimation de consommation est effectuée en cumulant l'énergie consommée (équation V.1) pour chaque phase. Il est important de noter que les mesures effectuées représentent **l'énergie minimale** consommée par un pixel car l'énergie dissipée par l'arbre d'horloge n'est pas comptabilisé ici. L'ajout de *buffers* d'entrée au modèle permettrait d'estimer plus finement la consommation.

Phase	Minimum Local (pJ/cycle)	Relief Pentu (pJ/cycle)
`RESET`	2.98	2.98
`LOAD`	3.52	4.01
`COEUR INIT`	5.88	6.06
`INIT`	5.42	6.63
`COMP 1`		16.26
`COMP 2`		5.28
Moyenne	4.87	6.79

Tab. V.2 – Energies mesurées correspondant à chaque phase du vecteur de test.

La table V.2 présente l'énergie consommée en picoJoule par cycle pour chaque phase du vecteur de test. La tension d'alimentation étant de 1.8 V, nous mesurons environ 6.8 pJ par cycle.

Durant la phase `INIT`, le pixel non-minimum fixe à deux reprises son voisin d'inondation (d'abords le voisin Sud, puis le voisin Ouest), d'où un accroissement de la consommation (20% environ) par rapport au pixel minimum.

Il est intéressant de remarquer la forte variation de consommation entre les deux cycles de relaxation (`COMP 1` et `COMP 2`). Le premier implique de nombreuses transitions (écriture des nouvelles données) alors que le second correspond à un état stable (écriture de l'étiquette où seul un bit a changé). Cette mesure montre que **tout pixel ayant convergé consomme** environ 5 pJ par cycle.

Afin d'estimer l'énergie consommée par un réseau de dimensions QCIF, supposons que chaque pixel consomme en moyenne 7 pJ par cycle soit 177 nJ par cycle pour l'image. Chaque traitement d'une image requiert un chargement et une relaxation car le rapatriement des résultats de segmentation s'effectue simultanément avec le chargement de l'image suivante. Sachant que le chargement d'une image dure 6 336 cycles et la segmentation environ 300 cycles, chaque traitement d'une image consomme 1.2 mJ, soit 30 mW pour une cadence de 25 images par seconde.

Si le réseau est une rétine, alors seulement 53 μJ est consommé par la segmentation d'une image, soit 1.3 mW pour une cadence de 25 images par seconde.

Dans des perspectives d'estimation plus fine, le signal d'horloge ainsi que les interconnexions doivent être comptabilisés. En effet, il est reconnu que la distribution de l'horloge vers toutes les bascules d'un circuit est fortement consommant. Par ailleurs, les pixels consomment à chaque cycle d'horloge même s'ils ont déjà convergés (comme les minima locaux par exemple).

Comme nous n'avons pas voulu utiliser un mécanisme coûteux et difficile de mise en veille de l'horloge au niveau des pixels inactifs (*gated clock*), ces considérations nous ont poussé à mettre l'accent sur la version asynchrone du pixel (§V.4).

V.2.5 Conclusion

L'étude de faisabilité d'un réseau de pixels autonomes est réalisée grâce à la synthèse d'un pixel synchrone. La simulation d'un réseau QCIF montre l'adéquation d'une telle architecture avec l'algorithme de *Hill-Climbing* réordonnancé, et montre **une fréquence de segmentation de l'ordre de 80 kHz** (hors chargement des images). Cependant, la surface d'un tel réseau (de l'ordre du centimètre carré pour un réseau SQCIF) reste trop élevée pour la technologie actuelle à des fins d'intégration dans un terminal portable. L'architecture consomme quelques dizaines de milliWatts pour une fréquence de segmentation de 25 images par seconde (chargement de l'image compris).

Tout pixel inactif consomme environ 5 pJ par cycle et ce, sans comptabiliser l'énergie consommée par l'horloge. Or nous avons vu à la section IV.5.1.3 (figure IV.35 page 126) que seulement 5% des ressources environ sont utilisées, donc par soucis d'optimisation de la consommation d'énergie, le style de conception asynchrone permettrait alors de réduire de façon drastique cette consommation car dans ce cas, seuls les processeurs actifs consomment.

V.3 Environnement de conception pour circuits asynchrones

Les fortes contraintes de consommation (§I.1 page 1) et le faible taux d'occupation des processeurs (définition IV.6 page 111) justifient la conception d'un pixel asynchrone. En effet, un processeur asynchrone inactif ne consomme quasiment pas d'énergie car aucune transition électrique n'est effectuée.

Dans cette section, nous introduisons les outils utilisés pour modéliser le pixel asynchrone. La lecture des différents programmes et algorithmes présentés dans la section V.4 et annexe C en sera alors plus aisée.

Tous les outils standards de conception de circuits microélectronique nécessitent la présence d'une horloge car ils sont basés sur un modèle synchrone, *i.e.* une succession de modules combinatoires intercalés par des registres cadencés. Cependant, le modèle de conception des circuits asynchrones est radicalement différent [ND97] car les modules combinatoires sont contrôlés par les données elles-mêmes. Des outils de conception dédiés à ce modèle sont donc nécessaires [Ren00].

Dans un premier temps (§V.3.1), nous présentons l'outil de conception développé par le laboratoire TIMA[1]. La seconde partie (§V.3.2) présente quelques éléments de la syntaxe du langage CHP (*Communicating Hardware Processes*) qui nous permettra de modéliser le pixel asynchrone (étude présentée dans la section V.4).

[1] Techniques de l'Informatique et de la Microélectronique pour l'Architecture d'ordinateurs.

V.3.1 Outil de conception

L'outil TAST (TIMA *Asynchronous digital circuit Synthesis Tools*) est un environnement de conception dédié aux circuits asynchrones. Il est constitué d'un compilateur/synthétiseur capable de générer soit un modèle fonctionnel en VHDL, soit une description synthétisée en portes logiques à partir d'une description de haut niveau du circuit (figure V.5).

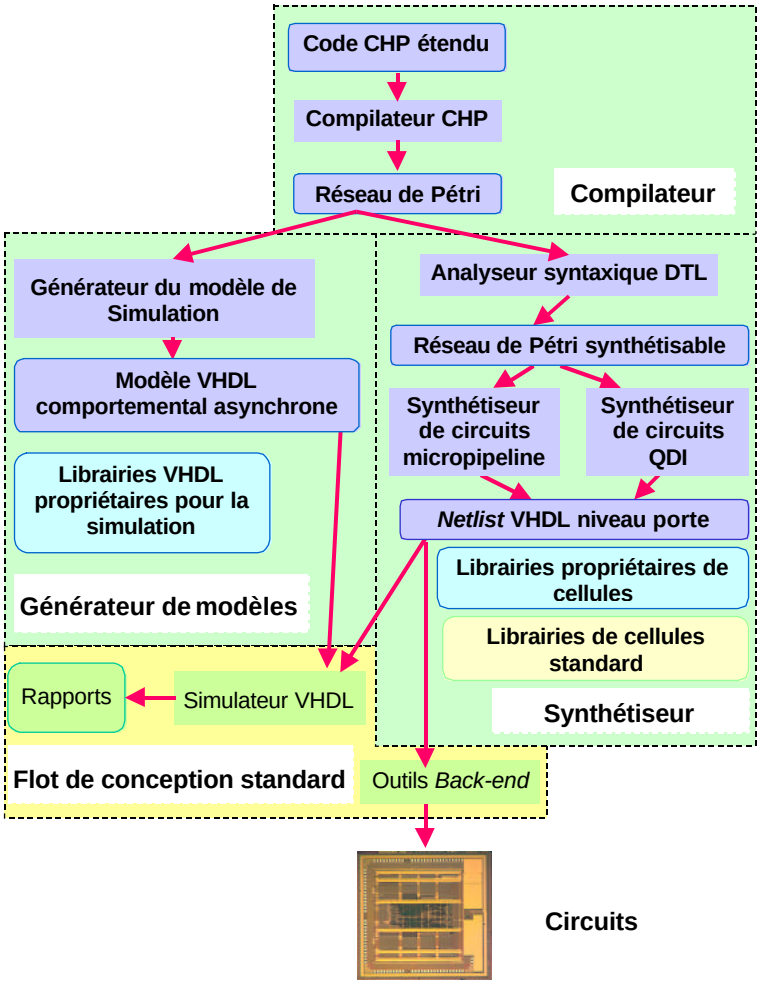


FIG. V.5 – Flot de conception de l'outil TAST (extrait et traduit de [DDRR+02]).

Cette description est effectuée à partir du langage CHP [Mar91], un dérivé du langage CSP (*Communicating Sequential Processes*) [Hoa78] [Hoa85] développé dans le cadre de l'étude de systèmes concurrents communicants. Le CHP est une version étendue du langage CSP incluant des opérateurs matériels spécifiques[1]. Pour des raisons d'expressivité et d'efficacité, ce langage est adapté aux besoins de l'outil TAST ; nous l'appelons **CHP étendu** dans la suite.

Le compilateur TAST génère un format intermédiaire sous forme de réseau de Pétri où différentes cibles peuvent être visées : soit un modèle de simulation fonction-

[1] `http://www-classes.usc.edu/engr/ee-s/577bb/syllabus.html`

nel (générateur de modèle, figure V.5), soit une description synthétisée de circuits *micropipeline* ou QDI (*Quasi Delay Insensitive*) [Viv01] [Hau95] [ND97] (synthétiseur, figure V.5).
La première cible, un modèle VHDL (*Very high scale integration Hardware Description Language*) [ABOR98] comportemental, est la forme utilisée pour valider la modélisation d'un pixel asynchrone (§V.4). Les deux autres cibles sous une forme synthétisée [Rez02] du réseau de Pétri généré sont décrites en détails dans [Ren00]. Afin que cette synthèse soit possible, le modèle CHP doit répondre au formalisme DTL (*Data Transfert Level*) [DDR+01]. Ce niveau de description est l'équivalent du RTL (*Register Transfert Level*) pour la synthèse de circuits synchrones.

V.3.2 Le langage CHP étendu

En perspective des programmes présentés dans la section suivante et annexe C, quelques éléments de syntaxe du langage CHP sont introduits. Une description détaillée peut-être consultée dans [Viv01], [DDRR+02] ou [Sir02].

Un programme est constitué d'instructions. Un groupe d'instructions est délimité par les symboles `[ ... ]`. La répétition d'un groupe d'instructions est symbolisée par `*[ ... ]`.

V.3.2.1 Opérateurs de composition

Les opérateurs de composition déterminent le mode d'exécution de plusieurs instructions. Soient `I1` et `I2` deux instructions :

- `[ I1 ;I2 ]` décrit une exécution séquentielle des instructions (`I2` ne s'exécute qu'une fois `I1` terminé) ;
- `[ I1,I2 ]` décrit une exécution parallèle des instructions (`I1` et `I2` peuvent potentiellement s'exécuter en parallèle).

V.3.2.2 Opérateurs de communication

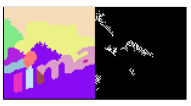
Ces opérateurs décrivent les actions effectuées sur les canaux de communication. Soit `C` un canal, `x` et `y` deux variables :

- `[ ... C!x ... ]` signifie que la variable `x` est écrite (*send*) sur le canal `C` ;
- `[ ... C?y ... ]` signifie que la donnée lue (*receive*) dans le canal `C` est mémorisée dans `y` ;
- `[ #C=>... ]` teste l'activité (*probe*) du canal `C`, il est utilisé par les opérateurs de contrôle (§V.3.2.3). Cet opérateur ne s'applique que sur un port utilisant un protocole de communication passif [DDRR01].

V.3.2.3 Opérateurs de contrôle

La modélisation du contrôle est symbolisée par "`=>`". Si `G` est une fonction booléenne et `I` un groupe d'instructions, **la commande gardée** `G=>I` signifie que la commande `I` est exécutée si et seulement si la garde `G` est vraie.

À partir de cette syntaxe, la table V.3 présente l'ensemble des structures de contrôle disponibles.



Sélection déterministe : Dès qu'une garde est vraie, le groupe d'instructions associé `I` est exécuté. Une seule garde peut-être vérifiée (exclusion mutuelle des gardes).	`[ G1=>I1`
	`@G2=>I2`
	`@ ...`
	`]`

Sélection indéterministe : Attend qu'au moins une garde soit vraie. Les gardes peuvent être non-stables ou non-exclusives, un arbitrage au niveau matériel est alors nécessaire afin de rendre le choix déterministe.	`[ G1=>I1`
	`@@G2=>I2`
	`@@ ...`
	`]`

Boucle déterministe : Tant qu'une garde et une seule est vraie, le bloc d'instructions associé est exécuté. Dès qu'aucune garde n'est vérifiée, la suite du programme est exécutée.	`*[ G1=>I1`
	`@G2=>I2`
	`@ ...`
	`]`

Boucle indéterministe : Tant qu'au moins une garde est vraie, l'arbitre détermine le bloc d'instructions à exécuter. Dès qu'aucune garde n'est vérifiée, la suite du programme est exécutée.	`*[ G1=>I1`
	`@@G2=>I2`
	`@@ ...`
	`]`

Tab. V.3 – Structures de contrôle.

V.4 Étude du pixel asynchrone

Cette partie présente plus en détails le modèle d'un pixel asynchrone sous forme de programmes en langage CHP étendu. Pour des raisons de présentation, seul le processus principal de chaque programme est décrit. L'annexe C page 199 complète la description du pixel.

V.4.1 Description générale

L'architecture du processeur asynchrone correspond à la granularité du réseau la plus fine : chaque pixel est associé à un processeur élémentaire asynchrone calculant de manière localement synchronisée. Entre pixels, nous utilisons des communications bloquantes en lecture (tant qu'au moins un pixel voisin n'envoie pas de données au pixel courant, ce dernier reste bloqué) et non-bloquantes en écriture car la donnée la plus récente est la donnée la plus pertinente (stratégie de conservation de la donnée la plus récente).

Chaque processeur (figure V.6, une représentation similaire mettant plus en avant l'aspect fonctionnel est présentée à la figure IV.15 page 92) est composé de trois catégories de modules :

1. **Les masques** (§V.4.2)**.** Ils permettent de modifier dynamiquement le degré de connectivité du processeur et de mémoriser la dernière donnée reçue. Nous dirons qu'un masque est ouvert s'il autorise le passage des données vers le cœur,

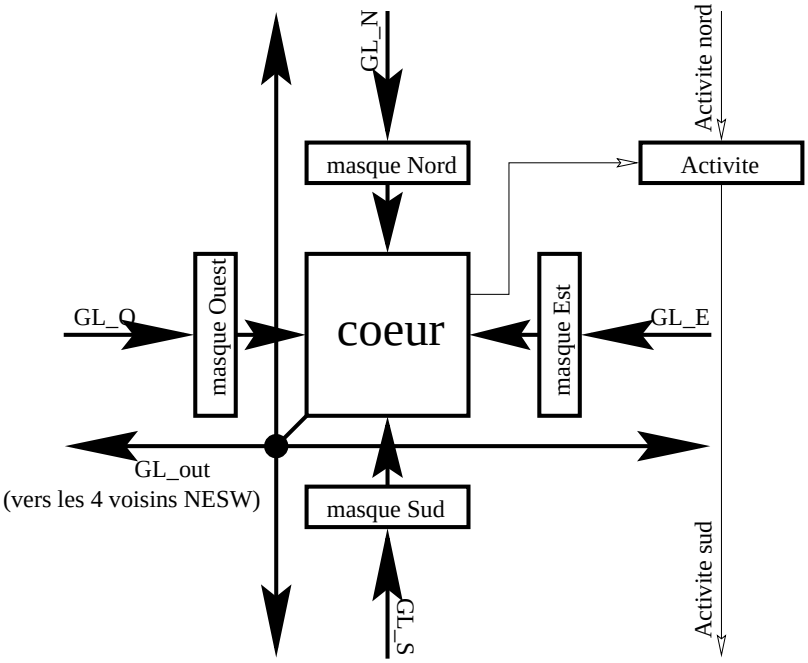


FIG. V.6 – Synoptique d'un pixel asynchrone.

et un masque est fermé s'il refuse tout transfert ;

2. **Le cœur** (§V.4.3)**.** C'est le module le plus complexe car il implante la machine à trois états (§III.2.2 page 36). Son canal de sortie forme une fourche et propage les données (étiquette et luminance) vers ses quatre voisins ;
3. **Le module de combinaison des activités** (§V.4.4)**.** Ce module contribue à la détection de fin. Il effectue un OU logique entre l'activité du processeur courant et l'activité issue du voisin situé au nord (figure V.1 page 144).

Notations. Lors de la description des différents modules, nous utiliserons les conventions suivantes :

- les canaux de communications sont représentés par une lettre majuscule. Les lettres `E` et `S` indiquent respectivement un canal d'entrée et un canal de sortie ;
- pour les boucles de mémorisation, les lettres `C` (*Current*) et `N` (*Next*) indiquent respectivement la valeur courante d'un canal ou d'une variable et l'état suivant ;
- les variables sont notées en minuscules (`x`, `a`, `b`...) ;
- les variables internes de la machine à état commencent par une majuscule. (`Etat`, `InitOk`...).

Afin d'éviter toute confusion entre les "variables internes" propres au processeur et les "variables" propres au langage CHP (par exemple la variable `x` lors d'une communication `E?x`), toute "variable interne" commence par une majuscule et toute "variable" est en minuscules.

Par soucis de description synthétisable, nous avons extrait les variables internes du processeur de l'ensemble des processus afin que les programmes se rapprochent d'une description au niveau DTL (§C.1 page 199) [Viv01].

V.4.2 Le masque

Le rôle du masque est de transmettre la dernière donnée reçue vers le cœur uniquement si ce dernier le souhaite. Les masques implantent des communications non-bloquantes entre deux pixels avec conservation de la dernière donnée (§IV.1.4 page 78) et constituent une mémoire de taille un pour chaque canal.

Le masque implante une communication non-bloquante avec le cœur du pixel : les données sont transmises au plus tôt si ce dernier en a demandé une. Nous verrons (§V.4.3) qu'à partir de communications non-bloquantes, le comportement du cœur lors de son initialisation est bloquant en lecture avec les quatre voisins.

Nous avons vu (§III.3 page 39) que seule la dernière donnée est importante, afin que l'étiquette du minimum d'attraction se propage. La perte de données intermédiaires est sans importance, voire même contribue à une diminution significative du temps de convergence puisque le nombre d'écritures et de lectures est réduit. C'est pourquoi, nous avons modélisé une interface (masque) qui puisse mémoriser la dernière donnée transmise tout en fournissant une autonomie au processus du cœur voisin et du cœur courant.

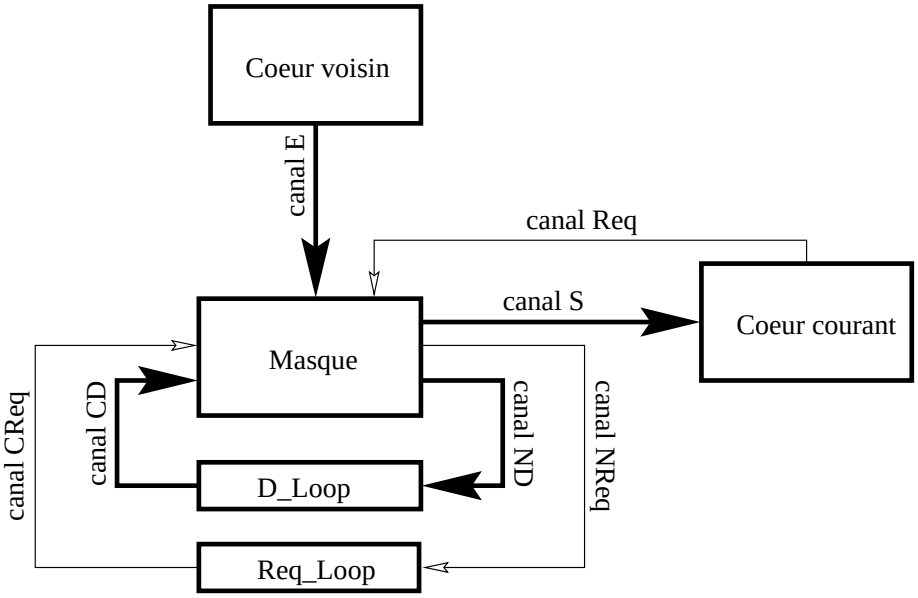


FIG. V.7 – Synoptique du masque.

La figure V.7 présente le schéma du masque (interface d'entrée). Les flèches épaisses correspondent à un bus de plusieurs bits alors qu'une flèche fine correspond à un seul voire zéro bits [DDR+01]. Le pixel voisin envoie une donnée au masque qui acquitte aussitôt cette nouvelle donnée (communication non-bloquante, algorithme V.1) : le pixel voisin continue aussitôt ses tâches annexes.

Le cœur courant demande une donnée au masque par l'intermédiaire d'un signal d'accès (canal `Req`). Cette demande est mémorisée par le masque afin que le cœur puisse continuer des tâches annexes même si des données sont indisponibles (par exemple le traitement d'une donnée disponible au niveau d'un autre voisin).

Remarque: Il est important de ne pas confondre la "requête du cœur" `Req` qui

correspond à la demande d'une donnée au masque (signal indépendant de l'implantation) et le "signal de requête" couramment employé pour implanter des protocoles de communication "données groupées" (signal dépendant de l'implantation) [RVR97].

L'ensemble des figures présentées dans cette partie V.4 ne font pas figurer la voie de retour (signal d'acquittement) nécessaire à l'implantation de communications bloquantes entre processus d'un module. Donc toute mention explicite d'une requête fait référence à une requête au niveau système, et non pas au niveau implantation matérielle d'une communication bloquante.

Algorithme V.1 Ecriture non-bloquante où la dernière donnée est prioritaire, et transmission au plus tôt des données.

```
 1: *[ #E =>
 2:    [ #CD =>
 3:      [ CD? , E?x ;                    —ancienne donnée non consommée—
 4:        ND!x ]                         —ancienne donnée écrasée—
 5:      @not(#CD) =>
 6:      [ E?x ; ND!x ]                   —ancienne donnée déjà consommée—
 7:    ]
 8:    @@#Req => [ Req? , NReq! ]         —demande d'accès ?—
 9:    @@#CReq and #CD =>                 —donnée et demande d'accès disponibles—
10:    [ CD?x ; [ S!x , CReq? ]
11:    ]
12:  ]
```

Cet extrait de programme (algorithme V.1) est sensible à l'arrivée d'une donnée issue du cœur voisin sur le port d'entrée (ligne 1). Le fait de tester la présence ou non d'une donnée dans le canal `CD` (ligne 2) permet de placer la dernière donnée reçue dans la boucle (processus `D_Loop`) et de supprimer la donnée précédente si le cœur ne l'a pas consommée entre-temps.

La ligne 8 joue le rôle de registre. Elle mémorise la demande d'accès du cœur. Cette écriture suppose qu'il ne peut y avoir deux demandes d'accès successives sur le canal `Req` sans qu'il y ait envoie de données sur le canal de sortie `S` (hypothèse respectée par le modèle d'exécution du cœur).

Ensuite, les lignes 9 et 10 transmettent au cœur la dernière donnée disponible dans le canal CD. Les donnés contenues dans les canaux "Courant" sont consommées, laissant la place disponible pour des données suivantes éventuelles.

Remarques:

- Cet algorithme implante des communications non-bloquantes entre le masque et le cœur où la stratégie est la conservation de la donnée la plus récente. La modélisation CHP d'une communication non-bloquante conservant la donnée la plus ancienne est présentée dans [Rob97].

- Il est important de noter que le masque n'effectue pas une communication bloquante durant la phase d'initialisation. Si deux processeurs voisins déterminent leur topologie locale (phase d'initialisation) à des vitesses fortement différentes, une distorsion du relief est possible (figure IV.9 page 83). Si nous souhaitons une initialisation correcte du pixel, chaque masque `x` doit contrôler le flux des

données, donc doit mémoriser la variable `PixInitOkx`, décrite §V.4.3.1.

Les gardes utilisées sont indéterministes (`@@`) car elles peuvent être vérifiées simultanément. Lors de la synthèse de ce module, des arbitres [RQFR01] sont implantés afin de rendre le choix déterministe.

Le canal `Req` permettant au cœur de demander une donnée au masque peut-être supprimé grâce au choix judicieux des ports de communication lors de leur déclaration. Un canal de communication est composé d'un port de sortie, d'un fil et d'un port d'entrée. Deux modules communiquent lorsque l'un d'entre eux initie une communication. Afin d'écarter tout conflit, un canal est toujours doté d'un port actif pouvant initier une communication et d'un port passif "attendant" une demande de communication.

L'outil TAST permet de déclarer des ports d'entrée-sortie passifs ou actifs, un port d'entrée passif (resp. actif) est connecté à un port de sortie actif (resp.passif). **Le test d'activité d'un canal (*probe*) ne s'effectue alors que sur un port de communication déclaré passif.**

Lors d'une première alternative, l'origine du canal `S` est un port actif et la destination un port passif. Le masque ne peut donc pas interroger le cœur sur sa volonté de recevoir une donnée. C'est pourquoi le canal `Req` entre le cœur du pixel et le masque (figure V.7) permet d'informer le masque des *desiderata* du cœur.

En perspective de l'optimisation des masques de communication, il est judicieux d'utiliser au niveau du masque, des ports passifs pour les deux canaux `E` et `S`, ainsi le canal `Req` est naturellement implanté par le protocole de communication (signaux de bas niveau). Le programme CHP correspondant est alors le suivant :

Algorithme V.2 Masque (mémoire tampon de taille un).

```
 1: *[ #E =>
 2:    [ #CD =>
 3:      [ CD? , E?x ;                    —ancienne donnée non consommée—
 4:        ND!x ]                         —ancienne donnée écrasée—
 5:      @not(#CD) =>
 6:      [ E?x ; ND!x ]                   —ancienne donnée déjà consommée—
 7:    ]
 8:    @@#S and #CD =>                    —donnée disponible et écriture possible—
 9:    [ CD?x ; S!x ]
10:  ]
```

De même, des gardes indéterministes sont utilisée car elles ne sont pas mutuellement exclusive. Si une écriture est possible alors que toutes les données sont consommées (cas `#S and not(#CD)`), le canal de sortie `S` est laissé vide. Cet état est exploité lors de la gestion de l'activité du pixel (§V.4.4 page 162 et §C.3.3 page 205)

V.4.3 Le cœur

Cette partie du pixel constitue la partie fonctionnelle du système, elle implante la machine à trois états de l'algorithme de *Hill-Climbing* réordonnancé (figure III.2 page 37).

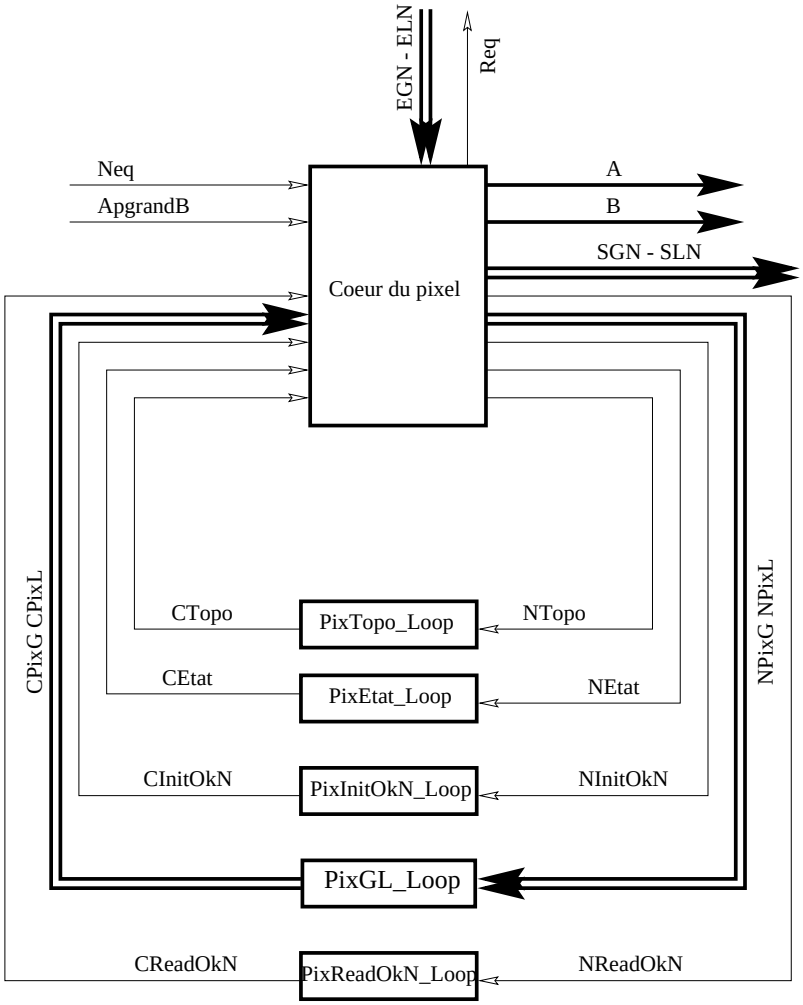


FIG. V.8 – Synoptique du cœur du pixel.

La figure V.8 présente quelques processus du cœur du pixel. Pour des raisons de lisibilité, seuls les bus et processus associés à la direction nord sont représentés. Une description détaillée de ce module est présentée dans l'annexe C.3 page 201.

Les sections suivantes présentent les variables internes du pixel (§V.4.3.1) figurant sur la figure V.8, décrivent globalement le fonctionnement du cœur (§V.4.3.2), introduisent un opérateur arithmétique adapté à nos besoins (§V.4.3.3) et montrent comment sont combinées les activités (§V.4.4).

V.4.3.1 Les variables

Dans la figure V.8, l'ensemble des boucles correspondent à l'extraction des variables internes du cœur du pixel. À chaque variable interne `X` du processeur correspond un processus `pixX_Loop`, un canal véhiculant la valeur courante `CX` et la prochaine `NX`. Les principales variables internes du cœur sont :

– `G` et `L` : respectivement de 8 et 13 bits, elles contiennent l'altitude et étiquette du pixel au cours du processus de segmentation. Au point de convergence, `L` contient l'étiquette de son minimum d'attraction ;

- `Topo` est un drapeau d'un bit indiquant si la machine est à l'état MP ou NM ;
- `Etat` est un drapeau d'un bit indiquant si le pixel est en cours d'initialisation ou de relaxation ;
- `PixInitOkx` : cette notation correspond à quatre drapeaux d'un bit où `x` indique une direction (N,E,S ou O). Ils permettent de bloquer la machine à l'état INIT tant que toutes les données ne sont pas lues (assure le voisinage complet $\mathcal{N}(v)$ dans les équations III.5 à III.7 page 36) ;
- `ReadOkx` regroupe également quatre drapeaux d'un bit. Chacune de ces variables implante le $i^{\text{ème}}$ bit de la variable parallèle $net_G[v]$ (§III.3.2 page 43) où v représente le pixel courant.

V.4.3.2 Description du cœur

La structure du programme est présentée par l'algorithme V.3.

Algorithme V.3 Structure du programme inclu dans le cœur du pixel.

⋆ Envoie d'une demande de données aux masques N,E,S et O.
⋆ Envoie vers les quatre voisins le niveau de gris et étiquette initiaux du pixel
—"Calcule" initialise ou inonde le pixel suivant l'état (`Etat`) de la machine—
1: Calcule(Nord) *—voir §C.3.1 page 202—*
2: Calcule(Est)
3: Calcule(Sud)
4: Calcule(Ouest)
5: Transition de l'état d'initialisation à l'état d'inondation *—voir §C.3.2 page 204—*
6: Gestion de l'activité du pixel *—voir §C.3.2 page 204—*

L'accès à chaque ligne de l'algorithme V.3 est aléatoire, c'est-à-dire que l'ordonnancement des lignes est inconnu. Cet algorithme n'est pas parallèle puisqu'il est impossible d'exécuter deux lignes simultanément.

La procédure "Calcule" est détaillée dans l'annexe C.3.1 page 202. Si plusieurs données sont disponibles dans différents masques, il est préférable d'implanter une exécution aléatoire des lignes 1 à 4. Ainsi aucune direction de propagation n'est privilégiée.

Lorsque tous les drapeaux `PixInitOkx` sont à 1, la ligne 5 effectue la transition de l'état d'initialisation à l'état d'inondation. Le programme CHP est présenté §C.3.2 page 204.

Enfin, nous devons détecter la convergence du réseau, c'est pourquoi la ligne 6 (§C.3.3 page 205) détermine l'état d'activité du cœur, et transmet ce dernier vers le processus "Activité" que nous présentons au paragraphe V.4.4.

V.4.3.3 Opérateur de comparaison

Les niveaux de gris et étiquettes sont soumis à des opérateurs de comparaisons (algorithmes IV.8 et IV.9 page 121), donc chaque cœur intégre un opérateur arithmétique. Nous rappelons qu'il faut au minimum 13 bits (8 192 valeurs) pour attribuer une étiquette unique aux 6 336 pixels d'une image SQCIF.

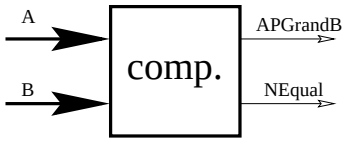


FIG. V.9 – Synoptique du comparateur. A et B sont les opérandes d'entrée sur 13 bits, les sorties (1 bit) sont à 1 si $A \geqslant B$ et si $A \neq B$.

Afin de réduire la surface du circuit, nous décrivons un comparateur 13 bits (soustracteur agrémenté d'une logique de comparaison en sortie, figure V.9) dérivé d'un additionneur séquentiel. Lors de la comparaison des niveaux de gris (8 bits), nous fixons à 0 les 5 bits de poids fort.

Les canaux `A`, `B` correspondent aux deux valeurs à comparer, `Neq` et `ApgrandB` sont à 1 si respectivement `A`$\neq$`B` et si `A`$\geqslant$`B`.

D'autres alternatives telles que celles présentées dans [FSR02] peuvent également être utilisées.

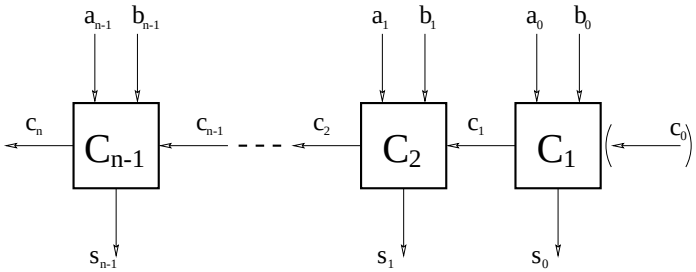


FIG. V.10 – Soustracteur séquentiel.

À partir de l'étude de l'unité arithmétique du processeur ASPRO (*ASynchronous PROcessor*)[RVR99] [Viv01], nous avons décrit un soustracteur séquentiel 13 bits (figure V.10).

Contrairement au calcul de soustraction à l'aide d'opérateurs d'addition, il n'est pas nécessaire d'effectuer un complément à deux de l'opérande `B`. Chaque cellule du comparateur, construite de façon similaire à l'additionneur un bit [Viv01] usuellement appelé *full-adder* (algorithme V.4), effectue une soustraction un bit avec une retenue entrante et génère en sortie un bit pour le résultat de la soustraction et un bit pour la retenue sortante.

Algorithme V.4 GP : Generate/Propagate du *full-adder*.

```
*[ A?a , B?b;
   [  a='0' and b='0' => GP!"0"
     @a='1' and b='1' => GP!"1"
     @a=not(b)        => GP!"2"
   ]
 ]
```

Étant adapté à nos besoins, le comparateur est de moindre complexité par rapport aux unités arithmétiques classiques. Chaque cellule est composée des deux processus suivants (algorithmes V.5 et V.6) :

Algorithme V.5 GP : Generate/Propagate du comparateur.

```
*[ A?a , B?b;
   [  a='0' and b='1' => GP!"0"
     @a='1' and b='0' => GP!"1"
     @a=b             => GP!"2"
   ]
 ]
```

Algorithme V.6 SC : Sub/Carry.

```
*[ GP?g ;
   [  g=0 => Cout!'0' , [ Cin?c ; S!c ]
     @g=1 => Cout!'1' , [ Cin?c ; S!c ]
     @g=2 => Cin?c ; Cout!c , S!not(c)
   ]
 ]
```

L'inversion bit à bit de l'opérande `B` usuellement effectuée auparavant lorsque des cellules *full-subber* sont utilisées, est intégrée dans les gardes du processus GP (algorithme V.5) ; elle est alors transparente.

Lors d'une soustraction à partir d'additionneurs élémentaires, le complément à deux de `B` est réalisé en fixant la première retenue c_0 à 1 (figure V.10). De même afin de simplifier au maximum l'architecture, la première cellule du comparateur est particularisée afin de prendre en compte, à l'intérieur même de ses calculs, cette retenue (voir annexe C.3.4 page 205).

Le dernier bit de retenue c_n est le bit de signe de la soustraction, nous l'avons appelé `ApgrandB`. Il est à 1 si `A`$\geqslant$`B` et à 0 sinon (`A`$<$`B`).

Il ne reste plus qu'à tester l'égalité (`A`=`B`) afin de compléter le calcul de comparaison. Ceci est réalisé par un test de nullité du résultat implanté par un OU de tous les bits du résultat de la soustraction. Le résultat est alors dirigé vers le canal `Neq` : à 0 si `A`=`B`, à 1 sinon.

V.4.4 L'activité

Ce processus participe à la détection de fin des calculs. L'état d'activité du pixel dépend de la présence ou non de données au niveau des masques d'entrée. Si de nouvelles données sont transmises au cœur, celui-ci se réveille et les consomme. Le pixel s'endort dès qu'il n'y a plus de donnée disponible.

La figure V.11 présente ce module où les deux bits d'activité (le pixel courant `PixActif` et le résultat de la combinaison de tous les processeurs situés au nord `ColActif`) sont mémorisés et traités séparément. Une description détaillée est présentée dans l'annexe C.4 page 206.

Dès que le cœur ou le pixel nord voisin modifie le signal `PixActif` ou `ColActif`, le calcul local d'un OU des deux signaux est effectué. Le résultat `Activite` est alors aussitôt propagé vers le pixel sud.

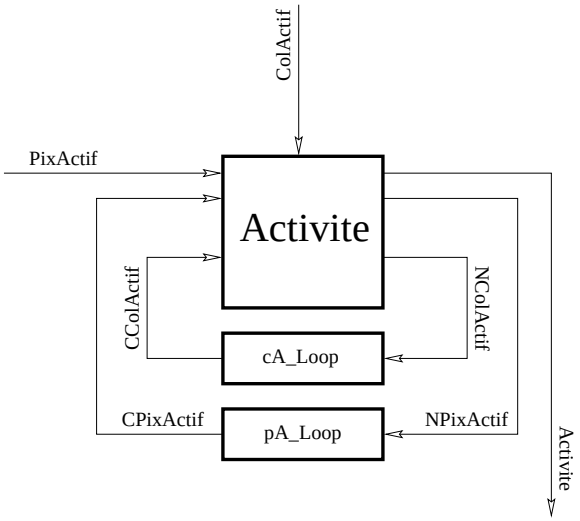


FIG. V.11 – Combinaison de l'activité.

V.4.5 Validation par simulation

Grâce au générateur de modèles (outil TAST, figure V.5), nous avons vérifié le bon comportement du pixel asynchrone. Les différents vecteurs de test simulent l'ensemble des configurations topographiques possibles :

- minimum local : phase d'initialisation seule, les quatre masques refusent toute donnée ;
- plateau minimum : vérification de l'unification de l'étiquette sur le plateau ;
- plateau non-minimum : transition de la phase MP à la phase NM ;
- non-minimum : détermination d'un voisin d'inondation lors de la phase d'initialisation, seul un masque est ouvert.

L'ensemble des états logique du processeur sont visualisés grâce à un simulateur VHDL classique.

Faute de temps, l'utilisation du synthétiseur n'a pu aboutir car certaines étapes restent à ce jour encore manuelles, comme la synthèse d'arbitres par exemple (implantation des gardes indéterministes `@@`). Nous n'avons donc pas pu analyser le comportement temporel et électrique d'un pixel asynchrone à partir d'une description VHDL au niveau porte. Cette étude fait partie des perspectives de cette thèse.

V.5 Conclusion

Afin de valider l'architecture parallèle implantant l'algorithme de *Hill-Climbing*, nous effectuons la conception matérielle d'un processeur élémentaire. Dans un premier temps, la granularité la plus fine (un pixel par processeur) est étudiée. Nous envisageons à plus long terme la synthèse d'un processeur à plus grosse granularité.

Le but ici est de démontrer la faisabilité de conception d'un réseau asynchrone, de quantifier la complexité microélectronique (surface), d'évaluer la vitesse de traitement du couple algorithme-architecture (chaîne critique), et de quantifier la consommation.

Afin d'exploiter au mieux les capacités des différents outils de conception disponibles, nous avons conçu un pixel synchrone et un pixel asynchrone.

V.5.1 Pixel synchrone

La technologie synchrone bénéficie d'outils très complets qui nous ont permis de valider le couple algorithme-architecture, et de quantifier la surface, la chaîne critique et la consommation.

La comparaison des segments obtenus par la simulation du modèle de bas niveau VHDL avec celui de haut niveau SystemC^{TM} asynchrone est sans appel : **l'architecture microélectronique segmente correctement les images.**

L'instanciation du réseau peut s'effectuer par aboutement des pixels car la topologie du réseau est une grille. Ainsi, la conception de cette architecture est réduite à l'optimisation d'un pixel et à sa duplication sous forme de lignes et de colonnes.

La surface et la consommation du réseau complet sont alors estimées en multipliant par le nombre total de pixels, les résultats obtenus pour un pixel.

Nous mesurons un temps de traversée de 30 ns environ (période minimale de l'horloge), soit **une capacité de calculs d'environ 17 000 segmentations d'images par seconde !** Les temps de traitement de la brique de segmentation sont minimes, les contraintes de temps induites par une chaîne de codage vidéo sont respectées.

Un réseau SQCIF consomme environ 53 μJ par segmentation d'image (chargement de l'image non compris). Il est important de noter que cette estimation ne prend pas en compte la distribution de l'horloge dans le réseau qui est généralement fortement consommante. Si une fréquence de segmentation de l'ordre de 25 Hz est suffisante, le réseau de 6 336 processeurs consommerait environ 1,3 mW (arbre d'horloge et chargement non compris).

Chaque pixel occupe environ 17 300 μm^2, soit un circuit d'environ 1 cm^2 pour un réseau SQCIF (88×72). Si le format d'image QCIF est souhaité pour le terminal multimedia, cette étude montre que la conception d'un circuit d'environ 4 cm^2 (technologie HCMOS-8) ne vérifie pas les contraintes d'intégration (*packaging*) et de coût de fabrication (le prix de revient pour la fabrication d'un tel circuit est bien trop importante). Une alternative utilisant un processeur à plus grosse granularité est alors à envisager : la réduction du nombre d'opérateurs de comparaison et l'optimisation de la surface dédiée à la mémorisation des variables nous permettraient de réduire la surface du circuit.

V.5.2 Pixel asynchrone

Grâce à l'outil TAST, nous avons vérifié le bon comportement d'un pixel asynchrone pour l'ensemble des configurations topographiques possibles. Seule une description VHDL comportementale a pu être générée.

Toutefois, les résultats obtenus lors de l'étude du pixel synchrone nous permettent d'effectuer une spéculation sur la surface, la consommation, et la vitesse du circuit.

De par l'implantation de chemins de données double rails [Ren00] et l'électronique de contrôle légèrement plus complexe, la surface ramenée au niveau pixel risque d'être plus élevée, au maximum d'un facteur deux. Seule la synthèse d'un pixel nous permettrait de la quantifier plus finement. Nous laissons cet aspect de l'étude à des travaux ultérieurs.

Quant à la vitesse d'exécution, les circuits asynchrones fournissent un résultat au plus tôt, les temps de traitement son dépendants des données. Nous estimons que la chaîne est au pire aussi longue que celle d'un pixel synchrone.

Lors de l'étude du pixel synchrone, chaque cycle d'horloge correspond à la latence d'une itération (définition IV.5 page 110). Ceci montre donc que les analyses de vitesse (§IV.5.2 page 131) effectuée pour une architecture asynchrone sous-estiment les capacités de notre architecture, car on estime que la chaîne critique moyenne du pixel asynchrone est équivalente à celle du pixel synchrone.

La réduction de la consommation est le point le plus prometteur. Tout processeur inactif consomme quasiment rien : sa logique est bloquée. Or nous avons vu lors de l'analyse de l'activité du réseau (figure IV.35b page 126) que seulement 5% des ressources environ sont utilisées, soit **une consommation divisée par 20.**

Si on suppose que la consommation d'un pixel asynchrone est de l'ordre de celle mesurée pour le pixel synchrone, seulement 2,6 μJ sont consommés par segmentation d'images SQCIF, et **seulement 65 μW** pour une fréquence de 25 segmentations par seconde. Seule une étude plus complète permettrait de confirmer ces estimations.

Chapitre VI

Conclusion et perspectives

Etude de faisabilité d'une chaîne de codage vidéo orienté-objet pour terminaux multimedia portable : le point de départ de nos travaux. À présent, nous établissons un bilan des orientations prises au cours de cette thèse, des difficultés rencontrées, des solutions proposées, et des expérimentations menées. Nous précisons également dans quelles mesures nous avons contribué à cette étude de faisabilité d'intégration d'une chaîne de codage vidéo orienté objet dans un terminal multimedia portable. Enfin, nous présenterons quelques perspectives de poursuite de ces travaux.

VI.1 Bilan

Dans cette étude, nous souhaitons apporter des éléments de réponses sur les possibilités d'intégration d'un codeur vidéo orienté objet dans un terminal multimedia portable. L'étude bibliographique sur ce type de codeur montre la forte complexité de la norme MPEG-4 et les nombreuses possibilités qu'elle inclut. Nous avons restreint le domaine à l'objet, élément sémantique d'une image, et plus précisément sur **l'extraction de ces objets** à partir d'une image brute, *i.e.* composée de simples pixels en couleurs ou luminance. À ce niveau des études, l'objectif est d'intégrer et de pouvoir utiliser le *Core Profile* de la norme MPEG-4, c'est-à-dire la représentation d'un flux vidéo sous-forme de plans superposés ayant une forme quelconque, dans un terminal portable.

L'opérateur de segmentation constitue un élément de base essentiel pour une chaîne totale d'extraction, de codage et de composition des objets. Nous nous sommes donc focalisés sur les méthodes de segmentation, et plus particulièrement sur celles présentant une faible complexité algorithmique (utilisation d'opérateurs simples) afin qu'elles puissent s'intégrer dans un terminal embarqué, et ne nécessitant aucune information *a priori*. Cependant, allier simplicité de l'algorithme et extraction des objets sans aucune information *a priori* est utopique. C'est pourquoi nous nous sommes orientés vers la segmentation non-sémantique de l'image qui est souvent utilisée comme pré-traitement d'une chaîne de segmentation sémantique. Comme à notre connaissance aucune proposition d'implantation rapide et à très basse consommation de cet opérateur n'existe, nous proposons une alternative originale alliant algorithme désynchronisé et architecture parallèle asynchrone.

Notre choix s'est porté sur **l'algorithme de *Hill-Climbing*** car ses traitements sont locaux, et nous montrons qu'il est possible de le réordonnancer afin que son implantation sur une architecture parallèle soit plus efficace. **L'algorithme développé est asynchrone** dans le sens où les traitements des pixels sont désynchronisés, et seul un point de synchronisation global (au niveau image) subsiste : le point de convergence. À l'aide d'**une modélisation du flux des données dans un graphe**, nous démontrons que sous cette forme, l'algorithme de *Hill-Climbing* réordonnancé est **un algorithme de segmentation qui converge et qui génère des partitions exactes.** Bien que le chemin de convergence soit inconnu, la position des lignes de partage des eaux (lignes séparant deux régions distinctes de l'image) est juste. Seule leur position exacte sur les plateaux non-minima ou suite au comportement aléatoire des pixels pivots est indéterminée. Toutefois, l'ensemble possible des partitions générées fait partie du paradigme de la segmentation.

Pour une segmentation pertinente de l'image, cet algorithme doit être précédé par le calcul de l'image gradient. Par exemple, des opérateurs morphologiques simples tels qu'érosions/dilations peuvent être utilisés. Nous ne nous sommes pas concentrés sur ce point car des opérateurs asynchrones de ce type existent [Rob97]. Ceux-ci peuvent tout à fait s'associer à la version réordonnancée de l'algorithme de *Hill-Climbing* car aucun point de synchronisation global supplémentaire n'est introduit. Par exemple, certaines zones rapides de l'image peuvent tout à fait être en cours de segmentation alors que d'autres plus lentes sont encore à l'étape de calcul du gradient.

L'intérêt des algorithmes asynchrones est alors triple : une composition aisée, une implantation parallèle efficace, et une mise en œuvre faiblement consommante grâce aux circuits asynchrones.

Afin de valider l'algorithme de *Hill-Climbing* réordonnancé pour une architecture dédiée, nous simulons cette implantation grâce à la bibliothèque de prototypage SystemC™. Nous déterminons que l'architecture la mieux adaptée à notre algorithme est un réseau de processeurs asynchrones faiblement couplés suivant un modèle de parallélisme SPMD. Afin d'exploiter au mieux la localité des traitements, nous utilisons un partitionnement du type «localement parallèle globalement séquentiel» (LPGS). L'asynchronisme matériel est motivé par une consommation d'énergie conditionnelle : seuls les processeurs qui calculent consomment. Une première modélisation «flot de données» (*data driven*) aboutit sur un point bloquant : les temps de simulation et les besoins en ressources machine avec la bibliothèque SystemC™ sont fortement pénalisants (plus de 2 Go de mémoire nécessaire).

Nous proposons donc une alternative utilisant l'horloge de SystemC™ où le modèle de chaque processeur est fonctionnellement asynchrone. L'horloge du simulateur correspond uniquement à une base temporelle du *scheduler*, le *quantum* de temps (une nanoseconde). Les calculs asynchrones sont alors simulés par une attente du processeur durant un nombre de cycles tiré aléatoirement entre 6 et 8. Les temps de simulation et les besoins en mémoires deviennent alors compatibles avec l'exploration d'architectures.

La modélisation à un haut niveau d'abstraction nous permet d'explorer rapidement un large spectre d'architectures parallèles allant de la granularité la plus fine (un processeur par pixel) jusqu'à la plus grosse (un processeur pour l'image). SystemC™

nous permet de paramétrer l'architecture : dimensions du réseau, taille mémoire des canaux, redirection des fichiers de visualisation...

À partir des résultats de simulation, l'analyse de l'activité du réseau et des temps de traitement sont effectués. Par comparaison avec un système séquentiel adapté aux terminaux embarqués, nous estimons que **l'architecture à granularité la plus fine serait environ 1 000 fois plus rapide.** Par ailleurs, cette simulation établit que cette solution à fine granularité n'utilise en moyenne chaque ressource (processeur) que pendant 15 % du temps de segmentation (taux d'occupation d'un processeur, définition IV.6 page 111).

Les simulations pour une granularité intermédiaire montrent que la taille de grain offrant le meilleurs compromis entre utilisation des ressources et temps de traitement se situe aux alentours de 100 pixels par processeur, soit 256 processeurs pour une image QCIF. Par ailleurs, une faible taille de l'ordre de quelques éléments pour les FIFO est suffisante. **Cette architecture à granularité intermédiaire permettrait alors de réduire par 89 les temps de segmentation.**

Afin de valider la faisabilité d'implantation d'un tel réseau, nous synthétisons un réseau de processeurs. Une description asynchrone en CHP est effectuée pour prouver qu'un pixel asynchrone est fonctionnellement réalisable. Une étude complémentaire est alors nécessaire afin que l'outil TAST puisse générer une liste synthétisée de portes. En l'état actuel des études, seule une modélisation fonctionnelle est réalisée, le comportement d'un pixel pour l'ensemble des topographies locales possibles est vérifié et validé.

Pour obtenir plus d'informations sur la consommation et la surface d'un tel réseau, nous nous sommes donc penchés sur la modélisation VHDL d'un réseau de pixels synchrones, et la synthèse avec des outils industriels de conception de circuits synchrones. Un réseau de processeurs à granularité la plus fine pour une dimension d'image de 88×72 processeurs pourrait segmenter **17 000 images SQCIF par seconde !** Une telle performance montre la pertinence de son implantation dans une chaîne de *tracking* où l'opérateur de segmentation doit représenter une charge de calculs marginale. Malheureusement, cette solution nécessiterait $1\,\mathrm{cm}^2$ de silicium. Elle ne respecterait pas les contraintes de coût et d'intégration liées aux terminaux embarqués.

Si la puissance de calcul va au-delà des besoins, nous montrons qu'une fréquence de segmentation de 25 images par seconde nécessiterait un besoin en énergie de quelques milliwatts seulement. Toutefois, tous les processeurs consomment et la distribution de l'horloge sur l'ensemble du réseau n'a pas été comptabilisée dans cette estimation. Or nous savons que ceci représente un coût non négligeable dans les architectures synchrones. De plus, l'analyse d'activité du réseau montre qu'environ seulement 5% des processeurs en moyenne sont actifs durant le processus de segmentation (taux d'activité du réseau, définition IV.7 page 111).

Ainsi, l'utilisation d'une technologie asynchrone permettrait de consommer uniquement lorsque cela est nécessaire, et réduirait de façon drastique la consommation. Même si un processeur élémentaire asynchrone consomme environ le double de nos estimations, **un réseau de processeurs asynchrones ne consommerait que quelques centaines de microwatts.**

VI.2 Perspectives

Seule la surface du circuit pour la technologie actuelle empêche la concrétisation du projet. Une granularité intermédiaire permettrait de respecter les contraintes de surface et de coût (le silicium est cher) : une RAM adaptée réduirait la surface des points mémoire implantés ici sous forme de registres, et le nombre de comparateurs arithmétiques serait aussi réduit. C'est pourquoi, nous pensons que la synthèse d'un processeur asynchrone pour une granularité intermédiaire permettrait de lever ce verrou.

Les programmes CHP fournis en annexe sont un point de départ pertinent pour cette étude. La synthèse d'un tel processeur permettrait alors de déterminer plus finement la consommation d'un réseau de processeurs asynchrones à granularité intermédiaire.

Cette thèse développe principalement l'aspect algorithmique. Nous ne nous sommes pas concentrés sur l'aspect exploitation des données et implantation concrète du système. C'est pourquoi le problème du chargement de l'image dans le réseau reste un problème ouvert. Une proposition consiste à synthétiser une rétine (insertion du système d'acquisition des images au sein même du pixel), cependant il faudra alors se concentrer sur la manière dont les régions obtenues seront exploitées.

Un second aspect est la sursegmentation de l'image : si aucun prétraitement n'est réalisé, et si le gradient brut est utilisé pour segmenter l'image, le résultat est généralement sursegmenté. Dans ce sens, nous avons envisagé une piste et développé, lors de l'encadrement d'un stage ingénieur [Pel02], une version asynchrone de l'algorithme de fusion de régions suivant un critère de volume [Vac95].

(a) Image originale à segmenter (*Claire* SQCIF).

(b) Partition obtenue à partir de l'image gradient **sans préfiltrage**.

(c) Partition obtenue après segmentation-fusion **asynchrone**.

FIG. VI.1 – Algorithme de segmentation-fusion asynchrone (extrait de [Pel02]).

La figure VI.1 présente les premiers résulats très prometteurs où les traitements sont localisés au niveau pixel. L'algorithme réalise **simultanément** l'algorithme de *Hill-Climbing* réordonnancé tel que présenté dans ce manuscrit, le calcul du volume de chaque région et la fusion des régions (toute région absorbe sa voisine si cette dernière ne vérifie pas un critère de volume).

La sursegmentation d'image peut également être évitée (figure VI.2) si l'algorithme de *Hill-Climbing* est itéré plusieurs fois pour une même image (le point de convergence atteint, tout pixel conserve le niveau de gris de son minimum d'attraction, seuls les connexions du réseau (net_G) et l'état de la machine sont réinitialisés).

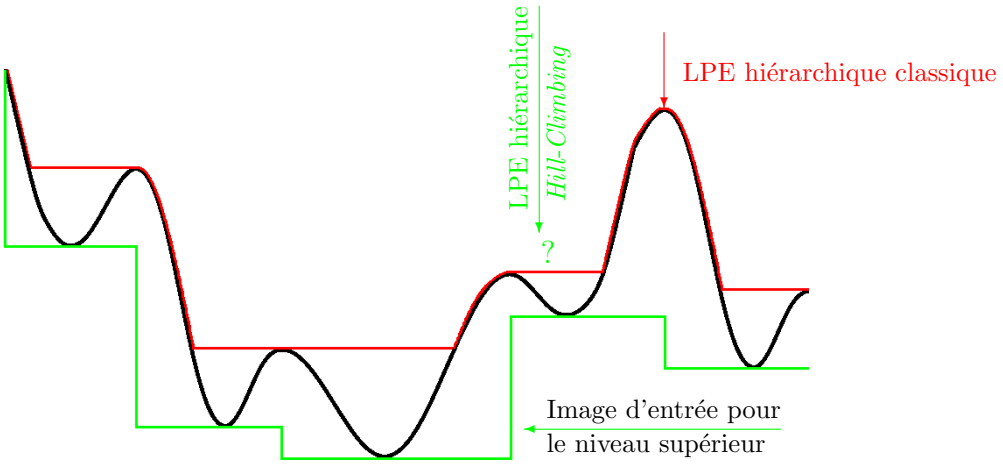


FIG. VI.2 – Illustration d'une hiérarchie de segmentation construites par l'algorithme de *Hill-Climbing* réordonnancé (en vert).

L'image d'entrée à un niveau de hiérarchie supérieur est composée de plateaux minima (les minima les plus pertinents) et de plateaux non-minima construits à partir des minima moins pertinents. Ainsi, cet emploi de l'algorithme de *Hill-Climbing* réordonnancé intègre la simplification d'image sans qu'aucune étape de simplification ne soit réalisée auparavant. C'est un corollaire de l'algorithme de LPE hiérarchique proposé dans [Beu94].

Annexes

Avant-propos

Cette partie du manuscrit regroupe trois annexes dédiées à :

A. la théorie des graphes ;
B. la morphologie mathématique ;
C. la programmation CHP du pixel.

La première regroupe les terminologies empruntées à la théorie des graphes. Elle peut servir de dictionnaire que l'on pourra consulter au cours de la lecture du chapitre III.

La seconde est dédiée à la morphologie mathématique. La présentation des différents opérateurs et filtres utiles à la simplification d'images, des notions de topographie, et des définitions formelles de segmentation d'images précise certaines notions évoquées dans ce mémoire.

Enfin, la troisième est plutôt destinée aux concepteurs de circuits asynchrones. Les programmes commentés développés en langage CHP *Communicating Hardware Processes* décrivant le comportement du pixel sont présentés.

Annexe A

Les graphes

Afin que l'ensemble des éléments issus de la théorie des graphes soient aisément consultables au cours de la lecture de ce manuscrit, nous avons regroupé l'ensemble des définitions utilisées dans cette annexe. Nous l'avons constituée à partir de notes de cours mises à disposition par P. Lopez [Lop00] et quelques pages internet[1,2].

Cette annexe présente quelques terminologies et concepts de base (§A.1), les éléments connexes d'un graphe (§A.2), puis quelques graphes particuliers (§A.3) utilisés par l'algorithme de segmentation. Enfin, la représentation des images par les graphes (§A.4) clôt cette annexe. La liste des définitions est présentée à la page xix.

A.1 Définitions de base

Définition A.1 (Graphe). *Un graphe $G = (V, A)$ est constitué d'un ensemble de nœuds (ou sommet) V et de paires de nœuds $A \subseteq V \times V$.*

Définition A.2 (Arc). *Le couple $a = (u, v)$ de A est appelé un arc. Il crée un lien du nœud u vers le nœud v.*

L'origine de a est le nœud u, et son extrémité est v.

Définition A.3 (Graphe non-orienté, arête). *Un graphe est non-orienté si l'orientation des liens entre deux nœuds quelconques u et v ne joue aucun rôle. Dans ce cas, les liens sont appelés **arête** (edge) $e = \{u, v\}$.*

L'algorithme de segmentation proposé n'utilisant que des graphes orientés, la suite de cette annexe ne présente que les éléments relatifs à cette catégorie de graphes.

Définition A.4 (Arc opposé). *Pour tout arc a de A, un arc **opposé** $\bar{a}$ est un arc dont l'origine est l'extrémité de a, et dont l'extrémité est l'origine de a.*

Définition A.5 (Graphe symétrique). *Un **graphe symétrique** est donné par un ensemble V de sommets et un ensemble A d'arcs tel que, pour tout a de A, il existe un arc opposé $\bar{a}$.*

[1] `http://www.jura.ch/lcp/cours/dm/graphes/lexique.html`
[2] `http://www.math.fau.edu/locke/graphthe.htm`

Définition A.6 (Arc incident, Voisin). *Soit $a = (v, u)$ un arc, a est dit **incident** à ses sommets : v est le **voisin** de u.*

Définition A.7 (Voisinage). *Pour tout nœud u d'un graphe $G = (V, A)$, son voisinage (Neighborhood) $\mathcal{N}(u)$ est défini par :*

$$\mathcal{N}(u) = \{v \in V \mid (v, u) \in A\} \tag{A.1}$$

On note $\overline{\mathcal{N}}(v) = \mathcal{N}(v) \cup \{v\}$ le voisinage étendu.
Remarque: Dans le cadre d'un graphe valué (définition A.29), cette définition du voisinage d'un nœud implique la convention suivante : un nœud v est voisin de u si ce dernier peut **lire** les données de v.

Définition A.8 (Fils, père). *v est le fils de u, et u est le père de v si $a = (u, v)$ est un élément de A.*

Un synonyme de fils est **descendant** ou **successeur**. Un synonyme de père est **ascendant** ou **prédécesseur**.

Définition A.9 (Ancêtre). *Nous appelons **ancêtres** de v l'ensemble de tous les nœuds p tel qu'il existe un chemin[1] joignant p à v.*

Définition A.10 (Multigraphe). *Un **multigraphe** $G = (V, A)$ est un graphe orienté pour lequel il peut exister plusieurs arêtes entre deux sommets.*

Les multigraphes ne sont pas utilisés car ici, les graphes sont exclusivement appliqués à la représentation d'images par un réseau où chaque nœud est relié à son voisin par un unique arc.

Définition A.11 (Sous-graphe). *Un graphe $G^* = (V^*, A^*)$ est un **sous graphe** de $G = (V, A)$ ssi $V^* \subseteq V$, $A^* \subseteq A$ et l'ensemble des arcs A^* sont incidentes uniquement aux sommets de V^*.*

Définition A.12 (Graphe partiel). *Un graphe $G' = (V', A')$ est dit **partiel** de $G = (V, A)$ s'il est engendré par un sous ensemble $A' \subseteq A$ et par tous les sommets de G ($V' = V$).*

Définition A.13 (Union, intersection). *Soit $P = (X, E)$ et $Q = (Y, F)$ deux graphes, on notera :*
- *$P \cup Q = (X \cup Y, E \cup F)$ l'**union** de deux graphes,*
- *$P \cap Q = (X \cap Y, E \cap F)$ l'**intersection** de deux graphes.*

Définition A.14 (Demi-degré extérieur). *Pour tout nœud v d'un graphe G, l'ensemble des arcs sortants w^+ de v est défini par :*

$$\forall v \in V, w^+(v) = \{u \in V \mid (v, u) \in A\} \tag{A.2}$$

*Le **demi-degré extérieur** de v dans G est défini par : $d_G^+(v) = |w^+(v)|$.*

[1] Définition A.16 page 179.

Définition A.15 (Demi-degré intérieur). *Pour tout nœud v d'un graphe G, l'ensemble des arcs entrants w^- de v est défini par :*

$$\forall v \in V, w^-(v) = \{u \in V \mid (u, v) \in A\} \tag{A.3}$$

*Le **demi-degré intérieur** de v est défini par : $d_G^-(v) = |w^-(v)|$.*

Dans ce manuscrit, nous appelons **degré de connexité d'un nœud** ou **degré**, le demi-degré intérieur.

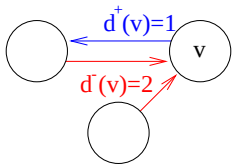


FIG. A.1 – Le degré d'un nœud.

A.2 Connexité dans les graphes

La connexité fait principalement appel à la notion de chemins dans un graphe.

Définition A.16 (Chemin). *Un chemin π de cardinal n entre deux pixels u et v sur un graphe orienté $G = (V, A)$ est un n-tuple de nœuds $(u_1, u_2, \ldots, u_n)$ tel que $u_1 = u$ et $u_n = v$, et $\forall i \in [1, n-1], (u_i, u_{i+1}) \in A$. L'origine et l'extrémité de π sont respectivement u et v.*

S'il existe un chemin joignant le sommet u au sommet v, alors u est joignable par v, noté $u \rightsquigarrow v$.

Proposition A.1 (Unicité des chemins). *Tout chemin de cardinal n joignant deux nœuds quelconques est défini de façon unique par une liste ordonnée de n nœuds dans G, où G n'est pas un multigraphe.*

Cette proposition découle de l'unicité de chacun des arcs $(u_i, u_{i+1}$ dans un graphe non multi-graphe.

Définition A.17 (Chemin simple). *Un chemin est **simple** si tous les nœuds sont distincts.*

Définition A.18 (Chemin hamiltonien). *Un chemin hamiltonien est chemin qui passe une et une seule fois par tous les sommets d'un graphe.*

Définition A.19 (Cycle). *Un chemin $\pi = (p_0, p_1, \ldots, p_l)$ forme un **cycle** si $l > 0$ et si $p_0 = p_l$.*

Définition A.20 (Boucle). *Une **boucle** est un arc qui connecte un sommet avec lui-même.*

Définition A.21 (Diamètre). *Le **diamètre** d'un graphe est la longueur du plus court chemin entre les sommets les plus distants.*

Définition A.22 (Point d'articulation). *Un point d'articulation est un point qui sépare le graphe en parties non connexes si on l'enlève.*

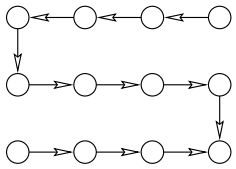


FIG. A.2 – Exemple de chemin hamiltonien en forme de S.

A.3 Graphes particuliers

Définition A.23 (Graphe simple). *Un graphe $G = (V, E)$ est **simple** :*
- *s'il n'est pas un multigraphe,*
- *s'il n'existe pas de boucle.*

Définition A.24 (Graphe acyclique). *Un graphe sans cycle (définition A.19) est appelé **acyclique**.*

Définition A.25 (Graphe planaire). *Un graphe planaire peut être représenté sur un plan (ou une sphère) sans intersection entre ses arcs.*

Définition A.26 (Graphe fortement connexe). *Un graphe orienté est dit **fortement**[1] **connexe** si pour toute paire ordonnée de sommets distinct (u, v), il existe un chemin de u vers v ($u \rightsquigarrow v$) et de v vers u ($v \rightsquigarrow u$).*

Définition A.27 (Arborescence). *Une **arborescence** est un graphe orienté où chaque sommet possède un seul ascendant sauf un qui n'en a pas : la **racine**.*

Pour tout sommet x de l'arborescence, il existe un chemin unique de la racine vers x.

Définition A.28 (Forêt d'arborescences). *Une forêt d'arborescences est une union de sous-arborescences*[2].

Soit $\mathbb{S}$ un domaine de définition quelconque.

Définition A.29 (Graphe valué). *Un graphe $G = (V, A, h)$ est dit **valué** si à chaque sommet V est associé une fonction $h : V \to \mathbb{S}$.*

Dans ce mémoire, cette représentation est souvent utilisée pour représenter la luminance de chaque sommet en prenant pour domaine $\mathbb{S}$, les valeurs entières allant de 0 à 255.

Définition A.30 (Graphe pondéré). *Un graphe $G = (V, A, \omega)$ est dit **pondéré** si à chaque arc de A est associé une masse ou **coût de connexion** $\omega : A \to \mathbb{S}$.*

Cette représentation nous permet d'insérer aisément le coût de passage d'un message entre deux nœuds voisins, comme par exemple les latences de transmissions ou les distances topographiques (définition B.33 page 193).

Définition A.31 (Réseau). *Un **réseau** $G = (V, A, h)$ est un graphe orienté valué.*

[1] L'adjectif **fort** indique que cette définition s'applique uniquement aux graphes orientés.
[2] Sous-arborescence : Sous-graphe constituant une arborescence.

Définition A.32 (Arborescence de recouvrement). *Une arborescence recouvrante $G_\curlywedge = (V_\curlywedge, A_\curlywedge)$ d'un graphe $G = (V, A)$ est un graphe partiel de G : $V_\curlywedge = V$ et $A_\curlywedge \subseteq A$.*

Définition A.33 (Arborescence de recouvrement minimum). *Une **arborescence de recouvrement minimum** $G_\curlywedge = (V_\curlywedge, A_\curlywedge, \omega)$ d'un graphe pondéré $G = (V, A, \omega)$ est une arborescence de recouvrement dont le coût de connexion de tous nœuds voisins est minimal :*

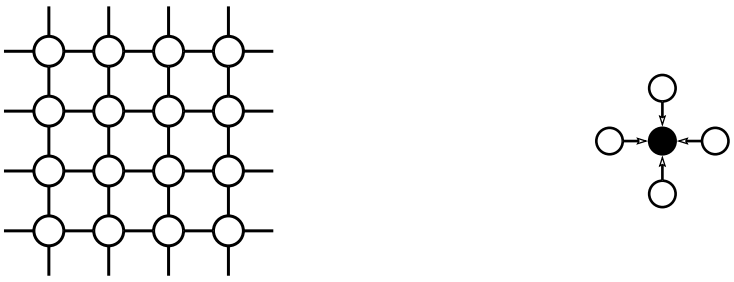
$$\forall a = (u, v) \in A_\curlywedge, \omega(a) = \min_{w \in V \mid (u,w) \in A} \omega(u, w) \tag{A.4}$$

A.4 Graphes appliqués au traitement d'image

Dans l'ensemble des définitions suivantes, nous interprétons le graphe valué $G = (V, A, h)$ comme un relief topographique où pour tout nœud u de V, $h(u)$ représente son altitude.

Définition A.34 (Grille). *Une grille est un graphe planaire simple où le degré de connexité[1] de chaque nœud est constant. Une grille est générée par sa maille : motif minimal de nœuds.*

Par abus de notation, on notera d_G^- le degré de connexité du graphe.



(a) Grille 4-connexe (b) Maille associée

FIG. A.3 – Exemple de grille et sa maille associée.

Définition A.35 (Image digitale). *Une **image digitale** est une grille valuée $G = (V, A, h)$ où les sommets sont appelés pixels (picture element).*

Pour l'ensemble de ce manuscrit, nous supposons que le degré de connexité d'une image digitale est toujours supérieur ou égal à quatre.

Définition A.36 (Chemin descendant, ascendant). *Un **chemin descendant** (resp. ascendant) est un chemin le long duquel les altitudes vont toujours en ordre décroissant (resp. croissant).*

*Un chemin **strictement** ascendant ou descendant est un chemin le long duquel les altitudes vont toujours en ordre **strictement** ascendant ou descendant. On note par $\Pi_f^\downarrow(p)$ (resp. $\Pi_f^\uparrow(p)$) l'ensemble des chemins strictement descendants (resp. strictement ascendant) partant d'un sommet p.*

[1] Définition A.15

Définition A.37 (Pixel minimum). *Un pixel minimum est un pixel à partir duquel il n'existe pas de chemin descendant :* $\Pi_f^{\downarrow}(p) = \emptyset$.

Définition A.38 (Complet inférieurement). *Une image digitale* $G = (V, A, f)$ *est* ***complet inférieurement*** *lorsque tout pixel* p *de* V *non minimum possède au moins un voisin* q *d'altitude strictement inférieure.*

Annexe B

Morphologie Mathématique

Brièvement, cette annexe présente les principaux opérateurs et transformations morphologiques appliqués au domaine de l'image. Elle complète les notions évoquées au cours des chapitres II et III. Ces opérateurs prennent tout leur sens et leur intérêt d'application dans le cadre de cette thèse, de par leur efficacité [ZMM99] [GM00] [Gom01] et leur simplicité d'implantation algorithmique et architecturale [Rob97] [GDM98] [DM01].

En guise d'introduction, nous rappelons l'historique de la morphologie mathématique (§B.1). À partir des notions de bases (§B.2), nous présentons quelques opérateurs élémentaires (§B.3) qui sont à la base d'autres plus complexes, tels que les opérateurs géodésiques (§B.4). Comme le présente l'historique, l'image est interprétée comme un relief topographique, d'où un emprunt des terminologies et l'établissement de mesures se référant à cette représentation (§B.5). Enfin nous présentons quelques définitions formelles des opérateurs de segmentation (§B.6).

La liste des définitions est présentée à la page xix.

B.1 Introduction et historique

La morphologie mathématique est la science de la forme et de la structure, basée sur des concepts de théorie des ensembles, topologiques et géométriques. Née des concepts émis en 1901 par Minkowski, elle a connu son essort à partir des années 1980 grâce aux travaux de G. Matheron, C. Lantuejoul, J. Serra et F. Meyer pour ne citer qu'eux. Elle est appliquée avec succès dans diverses disciplines, telles que minéralogie, diagnostic médical et histologie, compression de séquences vidéo et multimédia, vision par ordinateur, *etc.*

Elle est la branche non-linéaire du traitement d'images fondée sur la théorie des ensembles. Elle repose sur deux opérations élémentaires : le «sup» et le «inf». Elle propose un ensemble de transformations et de filtres variés par comparaison de l'image à analyser avec des éléments structurants de géométrie connue. Une seconde branche de la morphologie mathématique consiste à interpréter l'image luminance comme un relief topographique, donnant lieu, entre autres, à des algorithmes de segmentation robustes.

B.2 Notions de base

Une description détaillée des principaux opérateurs présentés dans cette annexe se trouve dans [Vac95] [Ser82] [Ser88] [Ser00].

Les opérations de bases s'appliquent sur des images binaires ou luminance. Les images binaires font appels à des opérateurs sur des ensembles, où par convention, les éléments de ces ensembles sont les pixels blancs. Les images luminance sont décrites comme des fonctions de $\mathbb{R}^2$ vers $\mathbb{R}$ où, par extensions des opérateurs binaires, les fonctions réalisent les opérations équivalentes.

Définition B.1 (Treillis). *Le treillis en morphologie mathématique est l'équivalent de l'espace vectoriel pour le traitement linéaire. La structure fondamentale est le treillis complet constitué d'un ensemble L muni d'une relation d'ordre $\leqslant$ telle que :*

$$\begin{cases} a \leqslant a \\ a \leqslant b, b \leqslant a \quad \Rightarrow \quad a = b \\ a \leqslant b, b \leqslant c \quad \Rightarrow \quad a \leqslant c \end{cases} \tag{B.1}$$

Pour toute famille d'éléments $\{X_i\} \in L$, il existe, dans L, un supremum et un infimum :

$\wedge\{X_i\}$ ou Inf : plus grand minorant
$\vee\{X_i\}$ ou Sup : plus petit majorant

Les lois fondamentales $\wedge$ et $\vee$ permettent de généraliser les relations d'ordres aux fonctions. Il apparaît alors deux classes de morphologie :

- **La morphologie binaire :** la relation d'ordre est l'inclusion.
- **La morphologie numérique :** soit $\mathcal{F}$ l'ensemble des fonctions de $\mathbb{R}^2$ dans $\mathbb{R}$, la relation d'ordre partielle est :

$$\forall g, h \in \mathcal{F}, g \leqslant h \Leftrightarrow \forall x \in \mathbb{R}^2, g(x) \leqslant h(x)$$

Dans le cas binaire, l'opérateur "sup" correspondra à l'union d'ensembles : $\cup$ alors que dans le cas numérique, il correspondra au sens usuel "sup" ou $\vee$. Par dualité, l'opérateur "inf" correspondra à l'intersection d'ensemble $\cap$ ou à l'opérateur numérique "inf" ou $\wedge$.

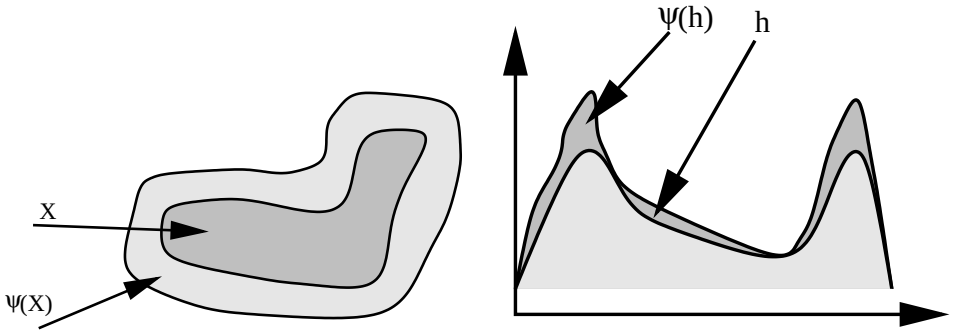
Définition B.2 (Extensivité et Anti-Extensivité). *Une transformation est extensive (resp. anti-extensive) si son résultat est toujours plus grand (resp. plus petit) que l'original.*

$$X \subseteq \psi(X) \qquad \forall h \in \mathcal{F}, \psi(h) \geqslant h \tag{B.2}$$

La figure B.1 illustre une transformation extensive dans le cas binaire et dans le cas des fonctions.

Définition B.3 (Croissance). *Une transformation est croissante si la relation d'ordre entre deux ensembles et leur transformée est conservée.*

$$X \subseteq Y \Rightarrow \psi(X) \subseteq \psi(Y) \qquad \forall g, h \in \mathcal{F}, \psi(g) \geqslant \psi(h) \tag{B.3}$$



a) Cas binaire b) Cas des fonctions

FIG. B.1 – Extensivité d'un opérateur ψ.

Définition B.4 (Dualité). *Deux transformations ψ_1 et ψ_2 sont duales si et seulement si appliquer l'une revient à appliquer l'autre sur le complémentaire de l'ensemble puis à complémenter le résultat final :*

$$\psi_2 = \{\psi_1(X^c)\}^c \tag{B.4}$$

Définition B.5 (Idempotence). *Une transformation est dite idempotente d'ordre n lorsqu'elle est invariante par itération. Si on note $\psi^k(X)$ l'application de ψ itérée k fois sur l'ensemble X, on dira que l'idempotence est atteinte lorsque :*

$$\psi^{k+1}(X) = \psi^k(X) \tag{B.5}$$

Cette définition est équivalente à celle donnée dans le cadre des réseaux associatifs (définition III.13 page 42).

B.3 Transformations morphologiques élémentaires

Le principe de base de la morphologie mathématique est de comparer les objets d'une image X à un objet B de référence, de taille et de forme données. Cet objet de référence B, appelé **élément structurant**, possède un centre (un point particulier). L'ensemble des points constituant B sont alors définis par rapport à ce centre..

Nous notons $\breve{B}$ le transposé de B, c'est à dire le symétrique de B par rapport à son centre (repéré par une croix).

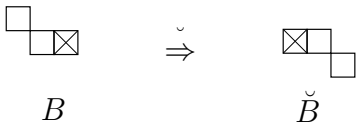


FIG. B.2 – Elément structurant B et son transposé $\breve{B}$.

La boule de rayon fixe est l'élément structurant le plus usité car il présente de bonne propriétés de symétrie ($B = \breve{B}$) et d'homogénéité. Dans certains cas, des

éléments structurants asymétriques sont utilisés afin d'extraire de l'image des caractéristiques géométriques particulières.

Pour tout pixel p d'une image, soit $\mathcal{N}(p)$ son voisinage (définition A.7 page 178).

Définition B.6 (Boule élémentaire). *Le sous-ensemble $B(p)$ de tous les pixels de $\mathcal{N}(p)$ qui sont à la distance*[1] *1 de p est appelé la boule élémentaire de taille 1.*

Dans le cas des images binaires, l'élement structurant est plan. Par contre, dans le cas des images en niveau de gris, nous utilisons des fonctions, donc l'élément structurant peut-être de forme quelconque (plat, conique, ...).

B.3.1 Erosion et dilatation

Ces opérations de base sont à l'origine d'un très grand nombre de transformations plus avancées.

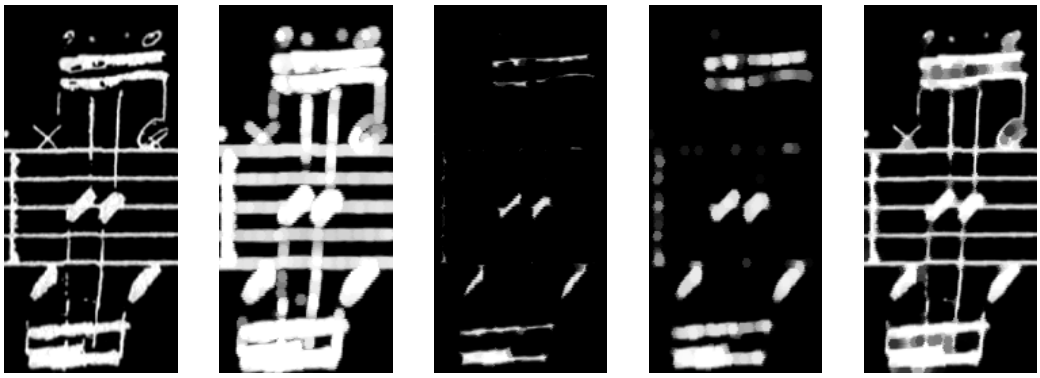
Définition B.7 (Dilatation). *Soit I une image binaire et B un élément structurant, on définit pour tout $x \in I$ B_x, l'élément B translaté en x. La dilatation d'un sous ensemble $X \subset I$ par B (notée $\delta_B(X)$) est l'union des points $x \in I$ tels que B_x intersecte X :*

$$\delta_B(X) = \{x \in I \mid B_x \cap X \neq \emptyset\} \tag{B.6}$$

Dans le cas d'une image numérique (figure B.3b), dilater une image f par B revient à donner à tout pixel x du domaine de l'image, la valeur maximale de l'image f dans la fenêtre d'observation définie par B, lorsque B est translaté en x :

$$\delta_B(h)(x) = \max\{h(y) \mid y \in B_x \cap I\} \tag{B.7}$$

L'opérateur de dilatation est un opérateur **croissant** et **extensif**.



(a) Original (b) Dilatation (c) Erosion (d) Ouverture (e) Fermeture

FIG. B.3 – Transformations de base par un élément structurant héxagonal de taille 3 pixels.

Par dualité, on définit l'érosion $\epsilon_B(X)$ (figure B.3c).

[1] Distance euclidienne, par exemple.

Définition B.8 (Erosion). *Un point x appartient à l'érodé de X, $\epsilon_B(X)$, si et seulement si B translaté en x est entièrement inclus dans X.*

$$\epsilon_B(X) = \{\delta_B(X^c)\}^c \tag{B.8}$$
$$= \{x \in I \mid B_x \subset X\} \tag{B.9}$$

Dans le cas d'une image numérique, l'érosion est définie par :

$$\epsilon_B(f)(x) = \min\{f(y) \mid y \in B_x \cap I\} \tag{B.10}$$

L'opérateur d'érosion est un opérateur **croissant** et **anti-extensif**.

L'application itérative de tels opérateurs permet de définir des opérations plus complexes [Ser00] telles que squelettes [Mey87], amincissement, épaississement, érosion ultime...

B.3.2 Ouverture, fermeture

Définition B.9 (Ouverture). *Une ouverture est une opération croissante, anti-extensive et idempotente.*

L'ouverture morphologique γ_B (figure B.3d) est définie à partir de l'érosion et de la dilatation :

$$\gamma_B = \delta_{\check{B}} \circ \epsilon_B \tag{B.11}$$

Elle supprime les objets de taille inférieure à l'élément structurant et transforme les presqu'îles en îles.

Définition B.10 (Fermeture). *La fermeture morphologique est définie par l'opération duale de l'ouverture.*

Elle consiste à dilater puis à éroder l'image.

$$\varphi_B = \epsilon_{\check{B}} \circ \delta_B \tag{B.12}$$

Cette fois-ci, les composantes sombres de dimension inférieure à l'élément structurant sont supprimées (figure B.3e), la fermeture "bouche" les trous.

B.3.3 Les gradients morphologiques

Construit à partir d'opérateurs de bases (érosion, dilatation, différence), le gradient morphologique fournit une mesure du module du gradient d'images numériques.

Le gradient symétrique et asymétrique. Le gradient morphologique symétrique Δ en un point peut-être décrit comme la plus grande différence entre les valeurs de son voisinage [Beu90, chap.2]. À partir d'opérateurs morphologiques, il se traduit comme la différence entre le dilaté et l'érodé :

$$\Delta(f) = \delta(f) - \epsilon(f) \tag{B.13}$$

Cet opérateur empêche la localisation des objets fins [Mar96, chap.4] et crée des lignes de crêtes "larges", d'autant plus si l'élément structurant est de grande taille.

Les gradients asymétriques supérieur Δ^+ et inférieur Δ^- sont plus précis :

$$\Delta^+(f) = \delta(f) - f \tag{B.14}$$
$$\Delta^-(f) = f - \epsilon(f) \tag{B.15}$$

Notons que, quelquesoit l'image f, le gradient morphologique est toujours positif car la dilatation $\delta(.)$ est extensive, et l'érosion $\epsilon(.)$ est anti-extensive (définition B.2 page 184).

Le gradient régularisé. Le principal inconvénient du gradient est sa grande sensibilité au bruit. Deux images, à première vue similaires, peuvent générer deux images gradient très différentes. Pour cela, différents algorithmes de régularisation, par compositions d'érosions et de dilations de différentes tailles, permettent de filtrer l'image afin d'atténuer le bruit et les méfaits de la digitalisation —une zone de faible pente est digitalisée par plusieurs plateaux, d'où l'apparition de minima dans le gradient numérique au lieu d'un seul dans le cas analogique—. Pour plus d'informations, vous pouvez consulter [Beu90, chap.2]).

Le gradient présenté ci-dessous est une forme régularisée du gradient morphologique.

Le gradient morphologique multiéchelles. Un gradient multiéchelles à base d'opérateurs morphologiques est présenté dans [Wan98]. Soit $f(x, y)$ l'image, et B_i un groupe d'éléments structurants carrés (une 8-connexité est supposée ici).

La taille de B_i est $(2i + 1) * (2i + 1)$ pixels pour i allant de 0 à 3. Le gradient morphologique multiéchelles est défini par :

$$MG(f) = \frac{1}{3}\sum_{i=1}^{3} \epsilon_{B_{i-1}}(\delta_{B_i}(f) - \epsilon_{B_i}(f)) \tag{B.16}$$

B.4 Les transformations géodésiques

Une transformation géodésique considére deux ensembles, le masque géodésique et les marqueurs, et consiste à transformer l'image marqueur contrainte par l'image masque. Seule la plus connue et son dual sont présentés : la **dilatation** et **l'érosion géodésique**.

Définition B.11 (Dilatation géodésique). *La dilatation géodésique de taille n notée $\delta^n(R, X)$ d'un ensemble X inclus dans un masque R est définie par :*

$$\delta^1(R, X) = \delta_B(X) \cap R \tag{B.17}$$
$$\delta^n(R, X) = \underbrace{\delta^1(R, \delta^1(R, \ldots \delta^1(R, X)))}_{n\ fois} \tag{B.18}$$

Dans le cas numérique, elle est définie par :

$$\delta^1(f_R, f) = inf(\delta_B(f), f_R) \tag{B.19}$$
$$\delta^n(f_R, f) = \underbrace{\delta_B(f_R, \delta_B(f_R, \ldots \delta^1(f_R, f)))}_{n\ fois} \tag{B.20}$$

Définition B.12 (Erosion géodésique). *L'érosion géodésique est définie par la transformation duale :*

$$\epsilon^n(f_R, f) = \left(\delta^n(f_R^c, f^c)\right)^c \tag{B.21}$$

B.4.1 Filtres morphologiques connexes

L'étude théorique des filtres morphologique est un vaste domaine qui dépasse le cadre de cette annexe. Nous nous contenterons seulement d'évoquer la famille des opérateurs connexes. Une classification de ces opérateurs est présentée dans [Gom01, chap.3].

Définition B.13 (Filtre morphologique). *Toute transformation ψ croissante et idempotente d'ordre 1 sur un treillis définit un filtre morphologique.*

$$\forall (f,g) \in \mathcal{F} \times \mathcal{F}, \begin{cases} f \leqslant g \Rightarrow \psi(f) \leqslant \psi(g) \\ \psi(f) = \psi \circ \psi(f) \end{cases} \tag{B.22}$$

Les filtres morphologiques connexes usuels sont l'**ouverture** et la **fermeture** (figures B.4a-b) par reconstruction géodésique (§B.4.1), les **filtres aréolaires** [Vin92] (figure B.4c) et les **nivellements morphologiques** [Mey98], [Mey00] (non présentés ici)

(a) Ouverture par reconstruction

(b) Fermeture par reconstruction

(c) Filtre aréolaire ($\lambda = 40$)

FIG. B.4 – Filtres géodésiques.

L'opération de reconstruction consiste à reconstituer, par itération jusqu'à idempotence d'une transformation géodésique, les composantes connexes du masque géodésique intersectant avec l'ensemble marqueur.

Définition B.14 (Ouverture par reconstruction). *L'ouverture de f_R par reconstruction selon f, est le supremum des dilatations géodésiques de f dans f_R, considéré comme fonction de f_R :*

$$\gamma^{rec}(f_R, f) = \vee\{\delta^n(f_R, f), n > 0\} \tag{B.23}$$

Définition B.15 (Fermeture par reconstruction). *La fermeture par reconstruction est définie par la transformation duale :*

$$\varphi^{rec}(f_R, f) = \left(\gamma^{rec}(f_R^c, f^c)\right)^c \tag{B.24}$$

Définition B.16 (Ouverture aréolaire binaire). *Soit $\{X_i\}$ les éléments compacts d'une image binaire et $\lambda \geqslant 0$. L'ouverture aréolaire de paramètre λ de X est définie par :*

$$\gamma^a_\lambda(X) = \bigcup\{X_i \mid i \in I, Aire(X_i) \geqslant \lambda\} \tag{B.25}$$

où $Aire(X_i)$ correspond au nombre de pixels de X_i.

Définition B.17 (Ouverture aréolaire numérique). *Pour une fonction $f : I \to \overline{\mathbb{R}}$, l'ouverture aréolaire $\gamma^a_\lambda(f)$ est définie par :*

$$\big(\gamma^a_\lambda(f)\big)(x) = \vee\{h \leqslant f(x) \mid x \in \gamma^a_\lambda(T_h(f))\} \tag{B.26}$$

avec $T_h(f)$ l'image seuillée (*Threshold*) de f à l'altitude h :

$$T_h = \{p \in I \mid f(p) \leqslant h\} \tag{B.27}$$

Lors de la simplification d'image par ouverture (resp. fermeture) géodésique par reconstruction, il est d'usage d'utiliser comme image marqueur l'érodé (resp. dilaté) de taille n de l'image à filtrer. Si la boule élémentaire est l'élément structurant des opérateurs géodésiques, toute structure dont **une** des dimensions est inférieure à $2n$ est éliminée. Si les structures linéaires sont importantes pour l'interprétation de l'image, il est alors préférable d'utiliser les filtres aréolaires car l'élément structurant "s'adapte" aux formes de l'image [Mar96]. Par exemple dans la figure B.4c, le cercle en haut à droite de l'image est conservé car sa surface est supérieure à celle de l'élément structurant (quarante ici), alors qu'il a été supprimé par la boule de taille trois dans la figure B.4a où l'image marqueur est obtenue par l'érodé de taille trois de l'image à filtrer (figure B.3c).

B.5 Topographie

Dans cette partie, nous rappelons succinctement les travaux de F. Meyer [Mey94], S. Beucher [Beu90] et A. Bieniek [BM98] où l'image luminance est interprétée comme un relief topographique.

Comme plusieurs éléments de topographies s'appuient sur la représentation par graphe d'une image, quelques définitions de l'annexe A sont utilisées.

Soit $G = (V, A, h)$ une image digitale. Soit $\pi = (p_1, p_2, \ldots, p_n)$ un chemin de cardinal n.

B.5.1 Eléments de base

Définition B.18 (Plateau). *Un plateau est un graphe fortement connexe de pixels d'altitude constante.*

Soit $G = (V, A, h)$ un graphe, et $P = (V_P, A_P, h) \subset G$ un plateau d'altitude (constante) h_P.

Définition B.19 (Frontière). *La **frontière** ∂_P d'un plateau $P = (V_P, A_P, h)$ d'altitude h_P est l'ensemble des sommets $u \in V_P$ ayant au moins un voisin $v \in V$ d'altitude $h_P \neq h(v)$.*

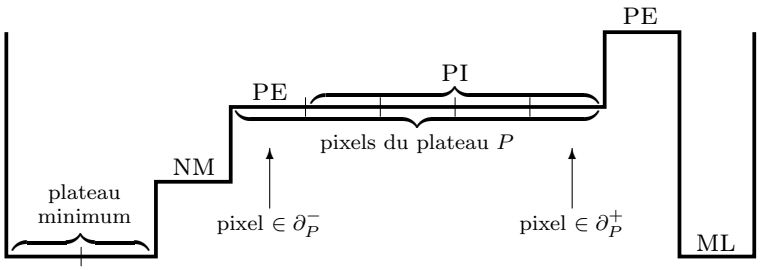


FIG. B.5 – Taxinomie des pixels d'une image.

Définition B.20 (Frontière inférieure, supérieure). *La* ***frontière inférieure*** ∂_P^- *(resp.* ***supérieure*** ∂_P^+*) de* P *est l'ensemble des nœuds* $v \in V_P$ *ayant au moins un voisin* $u \in V$ *dans* G *d'altitude* $h(u) < h_P$ *(resp.* $h(u) > h_P$*).*

Définition B.21 (Pixel extérieur). *Un* ***pixel extérieur*** *est un pixel appartenant à la frontière inférieure (noté* PE*).*

Définition B.22 (Pixel intérieur). *Un* ***pixel intérieur*** *(noté* PI*) est un pixel d'un plateau de surface supérieure ou égale à deux, et dont tous ses voisins sont d'altitude supérieure ou égale :*

$$\forall p_I \in V \text{ est un pixel intérieur } \Leftrightarrow \begin{cases} \forall p \in \mathcal{N}(p_I), h(p) \geqslant h(p_I) \\ \exists p \in \mathcal{N}(p_I) \mid h(p) = h(p_I) \end{cases} \tag{B.28}$$

Définition B.23 (Intérieur d'un plateau). *L'intérieur d'un plateau* P *est le sous-graphe de* P *où les nœuds frontière* $\partial_P = \partial_P^- \cup \partial_P^+$ *sont supprimés :*

$$P_{Int} = (V_{Int}, A_{Int}) \text{ est l'intérieur de } P = (V, A) \Leftrightarrow V_{Int} = V \setminus \partial_P \tag{B.29}$$

et

$$\forall u \text{ et } v \in V_{Int},\ (u, v) \in A_{Int} \Leftrightarrow (u, v) \in A \tag{B.30}$$

Remarque: Le terme **intérieur** est légèrement différent suivant s'il concerne un pixel ou un plateau. Afin de simplifier les expressions, nous avons nommé pixel intérieur (définition B.22), un pixel situé sur l'intérieur d'un plateau (définition B.23) **ou** sur la frontière supérieure (définition B.20).

Définition B.24 (Minimum régional). *Un minimum régional* $\mathcal{M}$ *[Vin93] est une composante connexe de pixels de même valeur* $h_{\mathcal{M}}$ *telle que tous les pixels du voisinage de* $\mathcal{M}$ *aient une valeur strictement supérieure.*

Définition B.25 (Minimum local). *Un minimum local est un plateau minimum constitué d'un seul pixel (noté* ML*).*

B.5.2 Mesures topographiques

Définition B.26 (Longueur d'un chemin). *La longueur du chemin π est défini par $\|\pi\| = \sum_i dist(p_i, p_{i+1})$.*

Remarque: Dans ce manuscrit, seules des images 4-connexe sont utilisées, la distance entre deux pixels voisins est alors toujours égale à un. Par contre, l'utilisation de voisinages de Chamfer [Mey94] impliquerait des distances différentes suivant le voisinage considéré.

Définition B.27 (Distance géodésique). *La distance géodésique $D^g_X : X \times X \to \overline{\mathbb{R}}$, par rapport à un sous-graphe de référence X, s'écrit :*

- *$D^g_X(x, y)$ = Inf. de la longueur des chemins, s'il en existe, allant de x à y en restant inclus dans X,*
- *$D^g_X(x, y) = +\infty$ sinon.*

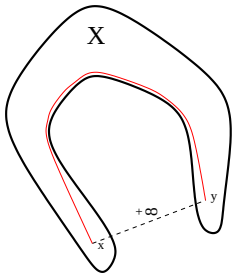


FIG. B.6 – Distance géodésique $D^g_X(x, y)$.

Définition B.28 (Distance géodésique d'un point à une frontière). *La distance géodésique de p, un pixel quelconque d'un plateau P, à ∂_P est définie par :*

$$D^g_P(p, \partial_P) = \min_{\forall p' \in \partial_P} D^g_P(p, p') \tag{B.31}$$

Définition B.29 (Pente). *La pente entre deux pixels voisins p et $q \in \mathcal{N}(p)$ pour $h(q) < h(p)$ est définie par :*

$$pente(p, q) = \frac{h(p) - h(q)}{dist(p, q)} \tag{B.32}$$

La pente n'étant pas définie pour deux pixels voisins d'un plateau, les définitions suivantes de cette section ne sont pas définies sur un plateau.

Définition B.30 (Pente de descente maximale). *La pente de descente maximale (lower slope) de la fonction h au point p est définie par :*

$$LS(p) = \max_{\substack{p' \in \mathcal{N}(p) \\ h(p') < h(p)}} \left(\frac{h(p) - h(p')}{dist(p, p')} \right) \tag{B.33}$$

Définition B.31 (Coût de passage d'un pixel à son voisin). *Le coût des déplacements de la position $h(p)$ à une position voisine $h(p')$ sur un relief topographique est définie par :*

$$\text{cost}(p,p') = \begin{cases} LS(p') * dist(p',p) & si\ h(p') > h(p) \\ LS(p) * dist(p',p) & si\ h(p') < h(p) \\ \frac{LS(p')+LS(p)}{2} * dist(p',p) & si\ h(p') = h(p) \end{cases} \tag{B.34}$$

Définition B.32 (π-distance topographique). *La π-distance topographique d'une fonction h entre deux sommets $p = p_1$ et $q = p_n$ est calculée en sommant l'ensemble des coûts entre deux sommets voisins le long du chemin $\pi = (p_1, p_1, \ldots, p_n)$:*

$$D_h^\pi(p,q) = \sum_{i=1}^{n-1} \text{cost}(p_i, p_{i+1}) \tag{B.35}$$

Définition B.33 (Distance topographique). *La distance topographique entre deux sommets correspond à la π-distance minimum :*

$$D_h(p,q) = \min_{\pi[p \rightsquigarrow q]} D_h^\pi \tag{B.36}$$

Remarques:

- La distance topographique n'est pas réellement une distance telle qu'on la définit généralement car D_h ne vérifie pas la troisième propriété ($D_h(p,q) = 0$ pour tout chemin d'un plateau) :
  1. positivité : $D_h(p,q) \geqslant 0$,
  2. symétrie : $D_h(p,q) = D_h(q,p)$,
  3. séparabilité : $D_h(p,q) = 0 \nLeftrightarrow p = q$, (non vérifiée !)
  4. inégalité triangulaire : $D_h(p,q) \leqslant D_h(p,r) + D_h(r,q)$
- Nous avons $q \in \Gamma(p)$ si et seulement si $\text{cost}(q,p) = h(q) - h(p)$. Dans les autres cas, le coût est plus élévé,
- Soit π un chemin de plus grande pente. Si nous choisisons une chemin γ tel qu'il ne soit pas de plus grande pente, alors $D_h^\gamma > D_h^\pi$.

B.5.3 Topographie construite

Cette section présente les principales composantes d'un relief topographique, obtenues à l'aide de calculs de distances ou à l'aide d'algorithmes itératifs.

Définition B.34 (Zone d'influence). *Soit, dans $\mathbb{R}^2$, $Y = \{Y_i,\ i \in \mathbb{N}\}$ formé de N composantes connexes compactes, toutes incluses dans le compact X [Ser00] (une image binaire par exemple) La **zone d'influence géodésique** de Y_i dans X se définit comme le lieu des points p de X qui sont géodésiquement plus proches de Y_i que de toute autre composante connexe de Y.*

$$zi_X(Y_i) = \{p \in X,\ \forall k \neq i, D_X^g(p,Y_j) < D_X^g(p,Y_k)\} \tag{B.37}$$

où la distance géodésique $D_X^g(p,Y)$ est la plus petite des distances géodésiques du point p à tous ceux de l'ensemble Y.

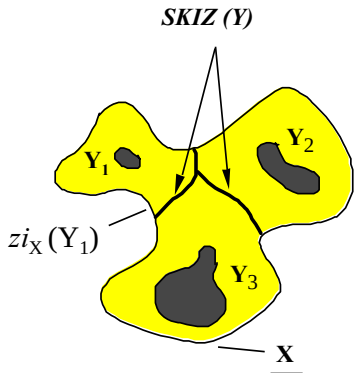


FIG. B.7 – Zone d'influence et son squelette.

La figure B.7 illustre cette zone zi_X. Pour les images multivaluées (une image luminance par exemple) nous employons plutôt le terme "bassin" emprunté à la géomorphologie.

Définition B.35 (Bassin d'attraction). *Le bassin d'attraction est constitué de l'ensemble des points ayant un seul et même attracteur [Nog98, chap.2], [BM98].*

Cette définition B.35 fait intervenir un processus de segmentation par calcul de distances.

Définition B.36 (Bassin versant). *Le bassin versant est constitué de l'ensemble de points ayant une seule et même source (minimum local ou plateau minimum).*

Cette définition B.36 fait intervenir un processus de segmentation par croissance de germes (par inondation).

Définition B.37 (Voisins de plus grande pente). *Pour tout p, les voisins $q \in \mathcal{N}(p)$ vérifiant $pente(p,q) = LS(p)$ sont dits de plus grande pente*[1]*. Ils forment un ensemble de points noté $\Gamma(p)$.*

Pour tout pixel intérieur p, $\Gamma(p)$ est vide. Si $|\Gamma(p)| > 1$, alors p est appelé **pivot** ou **point de bifurcation**. Une image est dite **semi-complète inférieure** si pour tout pixel p non-minimum, $|\Gamma(p)| > 1$.

L'ensemble des sommets q pour lesquels $p \in \Gamma(q)$ est noté $\Gamma^{-1}(p)$.

Définition B.38 (Ligne de plus grande pente). *La ligne de plus grande pente est un chemin orienté $\pi = (p_0 = p, p_2, \ldots, p_l = q)$ tel que deux pixels consécutifs sur le chemin maximisent la pente :*

$$\forall i \in [1, l], p_{i-1} \in \Gamma(p_i) \tag{B.38}$$

Cette définition ne s'applique pas pour des chemins situés sur des plateaux minima, ainsi que sur des images semi-complète inférieures. Si cette propriété n'est pas vérifiée, la ligne de plus grande pente n'existe pas pour un pixel intérieur p_I, car $LS(p_I)$ n'est défini. La notion de voisin de plus grande pente est donc étendue aux pixels de la frontière inférieure [BM98].

Soit ∂_P^- la frontière inférieure d'un plateau non-minimum : $\forall p' \in \partial_P^-, \Gamma(p') \neq \emptyset$. Soit $D_P^g(p,q)$ la distance géodésique d'un point p au point q sur un plateau P.

[1] La pente et $LS(p)$ sont définies par les définitions B.29 et B.30 page 192.

Définition B.39 (Voisin de plus grande pente généralisé). *L'ensemble des voisins de plus grande pente généralisés pour un pixel intérieur p :*

$$\widehat{\Gamma}(p) = \left\{ \bigcup_{\substack{q \in \partial_P^- \\ D_P^g(p,q) = D_P^g(p,\partial_P^-)}} \Gamma(q) \right\} \tag{B.39}$$

Cette définition s'applique également pour tout pixel p tel que $\Gamma(p)$ ne soit pas vide (même sur un "plateau" constitué d'un seul pixel), car p est un pixel frontière inférieure, d'où :

$$\forall p \in P \mid \Gamma(p) \neq \emptyset, \quad D_P^g(p,p) = D_P^g(p,\partial_P^-) = 0 \Rightarrow \widehat{\Gamma}(p) = \Gamma(p)$$

Définition B.40 (Amont). *Un pixel q est localisé en **amont** [Mey94] d'un pixel p, s'il existe un chemin de plus grande pente joignant p à q.*

B.6 Les lignes de partage des eaux

L'algorithme de segmentation par détermination des lignes de partage des eaux (LPE) construit les bassins versants d'une image. Il transforme une image luminance en une image où chaque bassin est étiqueté par une étiquette unique et les LPE par une étiquette spéciale "W" : c'est pourquoi cet algorithme est parfois appelé **transformation *watershed*** [RM01]. Il appartient à la famille des algorithmes de segmentation basé sur les transitions (§II.3.2.3 page 21).

Dans cette partie, nous présentons quelques définitions et algorithmes permettant de construire ces LPE.

B.6.1 LPE par inondation

Pour une image noir et blanc, les bassins d'attraction sont estimés par un processus de récurrence mettant en œuvre les zones d'influences (définition B.34) pour chaque niveau successif [NS96].

Soit $h : I \to \mathbb{N}$ une image luminance de domaine I, avec h_{min} et h_{max} le minimum global et le maximum global de h. On définit une relation de récurrence avec les niveaux de gris j allant de h_{min} à h_{max}, où les bassins versants associés à chaque minimum régional de h sont successivement étendus. Soit X_j l'union de cet ensemble de bassins au niveau j. Soit T_j l'image seuillée de h au niveau j (equation B.27 page 190). Une composante connexe de T_{j+1} peut être soit un nouveau minimum régional, soit l'extension d'un des bassins de X_j. Dans ce dernier cas, on détermine les zones d'influence géodésique (équation B.37) de X_j dans T_{j+1}, impliquant une mise à jour de X_{j+1}. Soit MIN_j l'union de l'ensemble des minima régionaux à l'altitude j. L'algorithme de segmentation par inondation est définie par la relation de récursivité suivante :

$$\begin{cases} X_{h_{min}} & = \{p \in I \mid h(p) = h_{min}\} = T_{h_{min}} \\ X_{j+1} & = \text{MIN}_{j+1} \cup zi_{T_{j+1}}(X_j), \text{ avec } j \in [h_{min}, h_{max}[\end{cases} \tag{B.40}$$

Les lignes de partage des eaux sont décrites par les SKIZ : les squelettes par zone d'influence définis comme la soustraction ensembliste de l'union des zones d'influence à l'image (voir figure B.7 page 194) :

$$Wshed(h) = I \setminus X_{h_{max}} \tag{B.41}$$

L'implantation de cette définition est coûteuse, car elle implique la détermination des zones d'influence pour tous les niveaux. Une autre solution, moins onéreuse en charge de calculs, fait intervenir une file d'attente hiérarchique ou FAH [Mey91]. Les minima régionaux, points sources des bassins versants, sont successivement étendus suivant l'ordre d'apparition des pixels dans la file de priorité a, où a correspond à l'altitude de l'eau au cours du processus d'inondation.

Cependant, la présence de points de bifurcation dans l'image biaisent cet algorithme.

B.6.2 LPE par distance topographique

Cette famille d'algorithmes (par fléchage, par ruissellement [Beu90], par ascension de colline [Mey94]...) détermine les bassins d'attractions d'une image en déterminant les distances entre un point quelconque de l'image et l'ensemble de ses minima.

Soit $(m_i)_{i \in I}$ l'ensemble des minima régionaux de la fonction h.

Définition B.41 (CB par mesure de distance topographique). *Le bassin d'attraction ou CB (Catchment Bassin) noté $CB(m_i)$ d'un minimum régional m_i est localisé par l'ensemble des points $x \in I$ qui sont plus prêts de m_i que de n'importe quel autre minimum régional pour la distance topographique :*

$$CB(m_i) = \{x \in I \mid \forall j \in I, j \neq i : h(m_i) + D_h(x, m_i) < h(m_j) + D_h(x, m_j)\} \tag{B.42}$$

Cette définition inclue le cas où l'altitude des minima est différente[1]. On dit alors que tous les points du bassin d'attraction $CB(m_i)$ sont attirés par le point m_i. Les points de la LPE sont alors ceux n'appartenant à aucun bassin d'attraction, c'est-à-dire les points ayant plus d'un attracteur.

Pour des images dépourvues de plateaux non-minima, la ligne de plus grande pente joignant deux points p et q minimise la distance topographique $D_h(p, q)$ (remarques de la définition B.33 page 193). Donc, définir les bassins d'attraction à partir de cette constation revient à dire que tout pixel appartenant à une ligne de plus grande pente sont attirés par le même minimum : celui situé à l'extrémité initiale de cette ligne.

Si l'image à segmenter comprend des plateaux non-minima, il est nécessaire, soit de transformer préalablement l'image en une image semi-complète inférieure (équation III.27 page 48, algorithme décrit dans [BM98] ou [MR98]) afin de les supprimer, soit d'utiliser la notion de voisin de plus grande pente généralisé (définition B.39 page 195).

Soit h le relief d'une image luminance, et l la fonction associant à chaque pixel une étiquette, un algorithme est un algorithme de segmentation par construction des LPE suivant les lignes de plus grandes pentes, s'il vérifie la définition suivante [MB00] :

[1] L'ajout des termes $h(m_i)$ et $h(m_j)$ permet de comparer des distances à partir d'un même point référence (le niveau de la mer en quelques sortes) et ce, quelque soit le minimum traité.

Définition B.42 (LPE par construction des lignes de plus grande pente). *Soit h et l deux fonctions associant à tout pixel une altitude et une étiquette.*

1. $l(m_i) \neq l(m_j)\ \forall i \neq j$, *avec* $\{m_k\}_{k\in I}$ *l'ensemble des minima des* h
2. *pour tout pixel* p *avec* $\widehat{\Gamma}(p) \neq \emptyset$, $\exists p' \in \widehat{\Gamma}(p)$ *avec* $l(p) = l(p')$

Définition B.43 (CB par détermination des lignes de plus grande pente). *Pour l'algorithme de segmentation décrit ci-dessus, un bassin d'attraction* $CB(m_i)$ *d'un minimum régional* m_i *est l'ensemble des pixels d'étiquette* $l(m_i)$ *:*

$$CB(m_i) = \{p \mid l(p) = l(m_i)\} \tag{B.43}$$

Annexe C

Programmes CHP

Cette annexe regroupe les programmes CHP du pixel asynchrone et complète la section V.4 page 154. Ces programmes ont permis la validation **fonctionelle** du pixel asynchrone et permettrons peut-être d'aider ceux qui souhaitent réaliser un réseau de processeurs asynchrones dédié à la segmentation d'images.

Avertissement : À des fins de synthèse, ces descriptions doivent être réécrites au niveau DTL. La plupart des gardes n'étant pas mutuellement exclusives, de nombreuses gardes indéterministes `@@` sont à introduire. Par ailleurs, certains groupes d'instructions introduisent l'opérateur de séquentialité «;». La réecriture de ces parties sous forme de machines à état fini [Viv01] est alors nécessaire.

Tout d'abord, nous rappelons comment une variable est extraite (§C.1) afin de décrire un programme au niveau DTL. Il sera alors possible de compléter la liste des programmes modélisant les trois modules d'un pixel asynchrone : les masques (§C.2), le cœur (§C.3) et la combinaison de l'activitée (§C.4).

C.1 Extraction des variable

La forme DTL[1] d'un programme CHP[2] nécessite l'extraction des variables utilisées par une boucle. Pour cela, un processus est créé pour chaque variable indépendante.

Puisqu'il est utilisé de nombreuses fois pour décrire le pixel, nous ne détaillons pas tous ces processus dans cette annexe. Cependant, l'algorithme suivant présente sa forme générique, où `x` est la variable à extraire, et `n` la taille de cette variable (en nombre de bit).

```
⋆ process x_Loop
⋆   port( c_Cx : out DI DR[n] ;
⋆         c_Nx : in DI DR[n] )
⋆ variable v_x : DR[n] ;
1: [ c_Cx!X_DEFAUT ;
```

[1] Data Transfer Level

[2] Communicating Hardware Processes

```
 2:  *[ c_Nx?v_x ; c_Cx!v_x
 3:   ]
 4: ]
```

Comme nous avons utilisé la convention de nommer `c_Cx` le canal véhiculant la variable **x** **C**ourante, et `c_Nx` le canal véhiculant la variable **x** **N**ext (suivante), il est possible de déterminer l'ensemble des processus manquants.

C.2 Le masque

Le masque joue le rôle d'interface d'entrée pour le cœur du pixel et de mémoire tampon de taille unitaire. Ce programme n'est pas décrit au niveau DTL, donc il n'est pas synthétisable.

Afin d'éviter toute ambiguïté dans les notations, nous présentons l'interface Est. La lettre `N` indiquant un canal *Next* ne se confond alors plus avec la lettre `N` indiquant la direction Nord.

Un choix judicieux des protocoles de communication (protocole passif pour `c_pEG` et `c_pEL` au lieu d'actif par défaut) permettrait de simplifier cette interface. De plus, ceci permettrait de supprimer les ports `c_pixReqE`, `c_CpixReqE` et `c_NpixReqE`.

```
 ⋆ process MemE
 ⋆   port ( c_rEG : in DI DR[8];            —Port d'entrée, reçoit un gris du voisin Est—
 ⋆          c_rEL : in DI DR[13];
 ⋆          c_pixReqE : in DI SR;                          —Requête du cœur du pixel—
 ⋆          c_pEG : out DI DR[8];                      —Sortie du gris vers le cœur—
 ⋆          c_pEL : out DI DR[13];
 ⋆          c_CEG : in DI DR[8];                        —Extraction des variables—
 ⋆          c_CEL : in DI DR[13];                              —Étiquette courante—
 ⋆          c_CpixReqE : in DI SR;                             —Requête courante—
 ⋆          c_NEG : out DI DR[8];                                  —Gris suivant—
 ⋆          c_NEL : out DI DR[13];
 ⋆          c_NpixReqE : out DI SR
 ⋆        )
 ⋆ variable v_newG : DR[8];
 ⋆ variable v_newL : DR[13];
 1: [ c_CpixReqE? ;  —Bug ? Par defaut, PetriNet met une valeur valide dans CpixReq...or on
    n'en veut pas !—
 2:  *[ #c_rEG and #c_rEL =>
 3:     [ #c_CEG and #c_CEL =>
 4:       [ c_CEG? , c_CEL? ,          —anciennes donnees non consommees par le pixel—
 5:        c_rEG?v_newG , c_rEL?v_newL ;
 6:         c_NEG!v_newG , c_NEL!v_newL                  —anciennes donnees ecrasées—
 7:       ]
 8:       @not(#c_CEG and #c_CEL) =>
 9:       [ c_rEG?v_newG , c_rEL?v_newL;  —l'ancienne valeur a déjà été consommée par le
    pixel—
10:         c_NEG!v_newG , c_NEL!v_newL
11:       ]
12:     ]
13:     @#c_pixReqE =>                                —une nouvelle requette est arrivee ?—
14:     [ c_pixReqE? ,   —libere la requette, le pixel peut ainsi poursuivre la scrutation de ses
    ports—
15:       c_NpixReqE!                                  —memorisation de la requette—
```

```
16:      ]
17:      @#c_CpixReqE and #c_CEG and #c_CEL =>  —le pixel veut-il une donnee et une donnee
     valide est-elle presente ?—
18:      [ c_CEG?v_newG , c_CEL?v_newL ;
19:        c_pEG!v_newG , c_pEL!v_newL , c_CpixReqE?              —relache la requette—
20:     ]
21:    ]
22: ]
```

C.3 Le cœur

Le programme CHP du cœur représente un code de 700 lignes environ, processus d'extraction des variables compris. C'est pourquoi, nous le présentons sous une forme découpée.

```
* process pixCoeur
*   port( c_pNG , c_pEG , c_pSG , c_pWG : in DI DR[8] ;
*         c_pNL , c_pEL , c_pSL , c_pWL : in DI DR[13] ;
*         c_CReadOkN , c_CReadOkE , c_CReadOkS , c_CReadOkW : in DI DR ;
*         c_CInitOkN , c_CInitOkE , c_CInitOkS , c_CInitOkW : in DI DR ;
*         c_CEtat : in DI DR ;                                  —INIT ou COMPUTE—
*         c_CTopo : in DI DR ;                                        —Topologie—
*         c_CpixG : in DI DR[8] ;
*         c_CpixL : in DI DR[13] ;
*         c_ONG , c_OEG , c_OSG , c_OWG : out DI DR[8] ;
*         c_ONL , c_OEL , c_OSL , c_OWL : out DI DR[13] ;
*         c_NReadOkN , c_NReadOkE , c_NReadOkS , c_NReadOkW : out DI DR ;
*         c_NInitOkN , c_NInitOkE , c_NInitOkS , c_NInitOkW : out DI DR ;
*         c_NEtat : out DI DR ;                                 —INIT ou COMPUTE—
*         c_NTopo : out DI DR ;                                       —Topologie—
*         c_NpixG : out DI DR[8] ;
*         c_NpixL : out DI DR[13] ;
*         c_pixReqN, c_pixReqE, c_pixReqS, c_pixReqW : out DI SR ;      —signaux de
  requette—
*         c_pixActiv : out DI DR ;                              —gestion de l'activite—
*         c_A , c_B : out DI DR[13] ;                                 —comparateur—
*         c_APGrandB , c_NEgal : in DI DR                 —resultats du comparateur—
*       )
* variable v_G, v_newG : DR[8] ;
* variable v_G13, v_newG13 : DR[13] ;              —utilise pour la conversion de type—
* variable v_L, v_newL : DR[13] ;
* variable v_L13, v_newL13 : DR[13] ;              —evite d'ecrire 2 fois newL (sinon pas
  synthetisable ?)—
* variable v_ReadOkN,  v_ReadOkE, v_ReadOkS, v_ReadOkW : DR;
* variable v_InitOk : DR;
* variable v_APGrandBG , v_NEgalG : DR;
* variable v_APGrandBL  : DR;
* [ c_pixReqN! , c_pixReqE! , c_pixReqS! , c_pixReqW! ,
*   c_ONG!GINIT , c_OEG!GINIT , c_OSG!GINIT , c_OWG!GINIT , —envoie les donnees d'init
  aux voisins—
*   c_ONL!LINIT , c_OEL!LINIT , c_OSL!LINIT , c_OWL!LINIT ;
  [
   *[
        Calcule(N)                                          —voir détails §C.3.1—
     @ Calcule(E)
```

```
      @ Calcule(S)
      @ Calcule(O)
      @ Transition                                   —voir détails §C.3.2—
      @ Activité                                     —voir détails §C.3.3—
    ]
  ]
```

C.3.1 Consommation d'une donnée

Le programme suivant, appelé "Calcule" au §V.4.3.2 page 160, correspond à la gestion d'une seule direction : le nord.

Remarques:

- `PIX_ACTIVE` est une constante fixée à "1",
- `INIT` et `COMP` sont deux constantes complémentaires d'un bit,
- `MP` et `NM` sont deux constantes complémentaires d'un bit.

```
 1: @#c_pNG and #c_pNL =>                                  —nouvelle donnee reveille le pixel—
 2: [ #c_CReadOkN="1"[2] =>          —Mauvaise initialisation ? Peut-être mettre plutôt
    #c_CReadOkN="1"[2] or #c_CInitOkN="0"[2] sinon les quatre voisins ne sont pas consultés
    lors de la détermination de la topologie locale...—
 3:  [c_CpixG?v_G , c_CpixL?v_L ,
 4:   c_pixActiv!PIX_ACTIVE ;                                —le pixel se reveille—
 5:   [ #c_CEtat = INIT =>
 6:     [ c_CInitOkN?v_InitOk , c_CEtat? , c_NEtat!INIT ;   —eco nrj : evite de lire Etat
    qd inondation—
 7:       [ v_InitOk = "0"[2] =>
 8:         [ c_pNG?v_newG , c_pNL?v_newL ;
 9:           v_newG13 := v_newG , v_G13 := v_G ,                —conversion de type—
10:           v_newL13 := v_newL , v_L13 := v_L ;                  —copie de variables
    (synthetisable ?...)—
11:           c_A!v_newG13 , c_B!v_G13 ,
12:           c_APGrandB?v_APGrandBG , c_NEgal?v_NEgalG ;     —calcule 'v_newG<v_G'—
13:           [ v_APGrandBG = "0"[2] =>     —detection voisin inondation : v_newG<v_G ⇔
    not(v_newG⩾v_G)—
14:             [ c_NpixG!v_newG , c_NpixL!v_newL ,
15:               c_CTopo?, c_NTopo!NM ,            —consomme et change de topologie—
16:               c_CReadOkN? , c_CReadOkE? ,
17:               c_CReadOkS? , c_CReadOkW? ,          —interdit toute lecture ailleurs—
18:               c_NReadOkN!'1' , c_NReadOkE!'0' ,
19:               c_NReadOkS!'0' , c_NReadOkW!'0' ,     —interdit toute lecture ailleurs—
20:               c_NInitOkN!'1'
21:             ]
22:            @v_NEgalG="0"[2] =>                                  —v_newG=v_G—
23:             [ #c_CTopo = MP =>
24:               [ c_A!v_newL13 , c_B!v_L13 , —v_newL13 evite de consommer v_newL—
25:                 c_APGrandB?v_APGrandBL , c_NEgal? ;       —calcule 'v_newL<v_L'—
26:                 [ v_APGrandBL = "0"[2] =>         —etiquette +petite (on la garde)—
27:                   [ c_NpixG!v_newG , c_NpixL!v_newL ,
28:                     c_NInitOkN!'1'
29:                   ]
30:                   @v_APGrandBL = "1"[2] => —etiquette superieure ou egale (on change
    rien)—
31:                   [ c_NpixG!v_G , c_NpixL!v_L ,
32:                     c_NInitOkN!'1'
33:                   ]
34:                 ]
```

```
35:                ]
36:                @#c_CTopo = NM =>
37:                [ c_NpixG!v_G , c_NpixL!v_L ,
38:                  c_NInitOkN!'1'
39:                ]
40:              ]
41:              @v_APGrandBG = "1"[2] and v_NEgalG="1"[2] =>      —v_newG<v_G (l'etat APGB='0' et NE='0' est interdit)—
42:              [ c_NpixG!v_G , c_NpixL!v_L ,
43:                c_CReadOkN? , c_NReadOkN!'0' ,       —interdit une lecture sur ce port—
44:               c_NInitOkN!'1'
45:              ]
46:            ]
47:          ]
48:          @v_InitOk = "1"[2] =>
49:          [ c_NpixG!v_G , c_NpixL!v_L , c_NInitOkN!v_InitOk
50:          ]
51:        ]
52:      ]
53:      @#c_CEtat = COMP =>
54:      [ c_pNG?v_newG , c_pNL?v_newL ;
55:        [ #c_CTopo = MP =>
56:          [ v_newG13 := v_newG , v_G13 := v_G ,                  —conversion de type—
57:            v_newL13 := v_newL , v_L13 := v_L ;                  —copie de variables (synthetisable ?...)—
58:            c_A!v_newG13 , c_B!v_G13 ,
59:            c_APGrandB?v_APGrandBG , c_NEgal?v_NEgalG ;      —calcule 'v_newG<v_G'—
60:            [ v_APGrandBG = "0"[2] =>      —detection voisin inondation (v_newG<v_G)—
61:              [ c_NpixG!v_newG , c_NpixL!v_newL ,
62:                c_ONG!v_newG , c_OEG!v_newG , c_OSG!v_newG , c_OWG!v_newG ,      —propagation des donnees d'inondation—
63:                c_ONL!v_newL , c_OEL!v_newL , c_OSL!v_newL , c_OWL!v_newL,
64:                c_CTopo? , c_NTopo!NM ,                          —change de topologie—
65:                c_CReadOkN? , c_CReadOkE? ,            —acquitte les anciens masques—
66:                c_CReadOkS? , c_CReadOkW? ,
67:                c_NReadOkN!'1' , c_NReadOkE!'0' ,      —interdit toute lecture ailleurs—
68:                c_NReadOkS!'0' , c_NReadOkW!'0' ,
69:                c_pixReqN!              —requette uniquement suivant la dir inondation—
70:              ]
71:              @v_NEgalG="0"[2] =>                                  —v_newG=v_G—
72:              [ c_A!v_newL13 , c_B!v_L13 ,    —v_newL13 evite de consommer v_newL—
73:                c_APGrandB?v_APGrandBL , c_NEgal? ;          —calcule 'v_newL<v_L'—
74:                [ v_APGrandBL = "0"[2] =>            —etiquette +petite (on la garde)—
75:                  [ c_NpixG!v_newG , c_NpixL!v_newL ,
76:                    c_ONG!v_newG , c_OEG!v_newG , c_OSG!v_newG , c_OWG!v_newG ,  —propagation des donnees— d'unification
77:                    c_ONL!v_newL , c_OEL!v_newL , c_OSL!v_newL , c_OWL!v_newL ,
78:                    c_pixReqN!
79:                  ]
80:                  @v_APGrandBL = "1"[2] =>        —etiquette plus grande, on fait rien—
81:                  [ c_NpixG!v_G , c_NpixL!v_L , c_pixReqN!
82:                  ]
83:                ]
84:              ]
85:              @v_APGrandBG = "1"[2] and v_NEgalG="1"[2] =>    —v_newG>v_G (on fait rien)—
86:              [ c_NpixG!v_G , c_NpixL!v_L , c_pixReqN!
```

```
87:             ]
88:           ]
89:         ]
90:         @#c_CTopo = NM =>
91:         [ c_NpixG!v_newG , c_NpixL!v_newL ,
92:           c_ONG!v_newG , c_OEG!v_newG , c_OSG!v_newG , c_OWG!v_newG ,        —
    propagation des donnees d'inondation—
93:           c_ONL!v_newL , c_OEL!v_newL , c_OSL!v_newL , c_OWL!v_newL ,
94:           c_pixReqN!
95:         ]
96:       ]
97:     ]
98:   ]
99: ]
100: @#c_CReadOkN="0"[2] =>          —le pixel recoit une donnee alors qu'il ne la veut plus—
101: [ c_CReadOkN? , c_pNG? , c_pNL? , c_NReadOkN!"0"[2]
102: ]
```

C.3.2 Transition de l'état d'initialisation à l'état d'inondation

Dans ce programme, les variables `c_CInitOk` sont exploitées. Il décrit le passage de l'état d'initialisation (réception des données des quatre voisins et détermination de la topologie locale) vers l'état d'inondation (soit unification des étiquettes pour un plateau, soit inondation si la topologie du processeur est NM).

```
1: @#c_CInitOkN='1' and
2: #c_CInitOkE='1' and
3: #c_CInitOkS='1' and
4: #c_CInitOkW='1' and #c_CEtat=INIT =>                    —Initialisation terminee—
5: [ c_CEtat? , c_CpixG?v_G , c_CpixL?v_L ;
6:   c_NEtat!COMP ,                                              —changement d'état—
7:   c_NpixG!v_G , c_ONG!v_G , c_OEG!v_G , c_OSG!v_G , c_OWG!v_G , —propagation des
    donnees d'initialisation—
8:   c_NpixL!v_L , c_ONL!v_L , c_OEL!v_L , c_OSL!v_L , c_OWL!v_L ,
9:   c_pixActiv!PIX_ACTIVE ,                                    —le pixel se reveille—
10:  c_CReadOkN?v_ReadOkN , c_CReadOkE?v_ReadOkE ,
11:  c_CReadOkS?v_ReadOkS , c_CReadOkW?v_ReadOkW ;
12:  [ v_ReadOkN = "1"[2] => c_pixReqN!
13:    @v_ReadOkN = "0"[2] => skip
14:  ] ,
15:  [ v_ReadOkE = "1"[2] => c_pixReqE!
16:    @v_ReadOkE = "0"[2] => skip
17:  ] ,
18:  [ v_ReadOkS = "1"[2] => c_pixReqS!
19:    @v_ReadOkS = "0"[2] => skip
20:  ] ,
21:  [ v_ReadOkW = "1"[2] => c_pixReqW!
22:    @v_ReadOkW = "0"[2] => skip
23:  ] ,
24:  c_NReadOkN!v_ReadOkN , c_NReadOkE!v_ReadOkE ,   —Utilise lors du passage MP→NM
    car une requette peut-etre envoyee alors que le pixel est inonde entre temps—
25:  c_NReadOkS!v_ReadOkS , c_NReadOkW!v_ReadOkW
26: ]
```

C.3.3 Activité du cœur

Cette garde est traversée si aucune donnée n'est disponible sur les ports d'entrée du cœur : le pixel s'endort.

```
1: @not(#c_CInitOkN='1' and
2:      #c_CInitOkE='1' and
3:      #c_CInitOkS='1' and
4:      #c_CInitOkW='1' and #c_CEtat=INIT) =>
5:      [ c_pixActiv!PIX_SLEEP ]                    —le pixel s'endort—
```

Remarques:

- Le pixel risque de s'endormir alors que la phase INIT n'est pas encore terminée. Cependant *a priori*, au moins un pixel devrait être en-train de calculer, donc le reseau reste toutefois globalement actif,
- `PIX_SLEEP` est une constante fixée à "0".

C.3.4 Comparateur

Le comparateur analyse le résultat de la soustraction des opérandes `A` et `B` (figure V.9 page 161). À la sortie du comparateur, le canal `APGrandB` est à 1 si A⩾B, et le canal `NEqual` est à 1 si A≠B.

Construit à partir d'un *Full-Adder*, nous avons particularisé ce comparateur en effectuant le complément à 2 de `B` à l'intérieur même des gardes. La collaboration des deux processus *Generate-Propagate* et *Sub-Carry* modifiés forme notre comparateur. Le premier combine le $x^{\text{ième}}$ bit des opérandes `A` et `B`. Il propage alors un message `GP` de deux bits vers le soustracteur. Ce deuxième processus combine la retenue `Cin` avec `GP`.

L'algorithme suivant, identique pour les treize cellules du comparateur, décrit le processus *Generate-Propagate* modifié.

```
* process GP1
*   port( c_A1 , c_B1 : in DI DR ;
*         c_GP1 : out DI DR[2]
*       )
* variable a , b : DR ;
1: [
2:  *[ c_A1?a , c_B1?b ;
3:     [ a='0' and b='1'   => c_GP1!"0"[3]
4:       @a='1' and b='0'  => c_GP1!"1"[3]
5:       @a=b              => c_GP1!"2"[3]
6:     ]
7:   ]
8: ]
```

Nous avons supprimé la première retenue c_0, donc nous obtenons le premier "Sub-Carry" `SC0` différent des autres `SCn`, avec `n` allant de un à douze.

```
* process SC0
*   port( c_GP0 : in DI DR[2] ;
*         c_S0 , c_C1 : out DI DR
*       )
* variable g : DR[2] ;
1: [
2:  *[ c_GP0?g ;
3:     [ g=0 => [ c_C1!'0' , c_S0!'1' ]    —la premiere retenue est toujours egale a un—
```

```
4:       @g=1 => [ c_C1!'1' , c_S0!'1' ]
5:       @g=2 => [ c_C1!'1' , c_S0!'0' ]
6:     ]
7:   ]
8: ]
```

L'algorithme suivant décrit le processus `SC1`.

```
⋆ process SC1
⋆   port( c_GP1 : in DI DR[2] ;
⋆         c_C1  : in DI DR ;
⋆         c_S1 , c_C2 : out DI DR )
⋆ variable c : DR ;
⋆ variable g : DR[2] ;
1: [
2:  *[ c_GP1?g ;
3:     [ g=0 =>  [ c_C2!'0' , [ c_C1?c ; c_S1!c ] ]
4:       @g=1 => [ c_C2!'1' , [ c_C1?c ; c_S1!c ] ]
5:       @g=2 => [ c_C1?c ;
6:                 c_C2!c , c_S1!not(c) ]
7:     ]
8:   ]
9: ]
```

Le canal `APGrandB` véhicule la dernière retenue du soustracteur. Donc dans le dernier processus `SC12`, nous avons substitué le canal `c_C13` par le canal `APGrandB`.

Enfin, l'information d'inégalité des deux opérandes est fournie par la combinaison logique bit à bit du résultat de la soustraction. Une porte OU à treize entrées, ou une structure en «log» peut-être utilisée.

C.4 L'activité

Ce processus est un point mémoire de l'activité du pixel auquel il est rattaché et de l'activité combinée issue du pixel situé au nord.

Les noms `pA` et `cA` s'adressent respectivement à l'activité du **p**ixel courant et de la **c**olonne située au nord.

```
⋆ process pActivite
⋆ port( c_pixActiv , c_isColActiv : in DI DR ;
⋆       c_CpA : in DI DR ;                          —CpA ⇔ Courant-pix-Active—
⋆       c_CcA : in DI DR ;                        —CcA ⇔ Courant-column-Active—
⋆       c_isActive : out DI DR ;
⋆       c_NpA , c_NcA : out DI DR )
⋆ variable v_pAct, v_cAct, v_newAct, v_OrAct : DR ;
1: [ c_isActive!PIX_ACTIVE ;          —force le reseau a etre actif lors de l'initialisation—
2:  *[ #c_pixActiv =>                                —le pixel a change d'etat d'activite—
3:     [ c_pixActiv?v_newAct , c_CcA?v_cAct , c_CpA? ;
4:       v_OrAct := v_cAct or v_newAct ;        —effectue un OU logique des activites—
5:       c_isActive!v_OrAct , c_NpA!v_newAct ,       —memorise le nouvel etat local—
6:       c_NcA!v_cAct
7:     ]
8:     @#c_isColActiv =>              —le pixel du dessus a propage une info d'activite—
9:     [ c_isColActiv?v_newAct , c_CcA? , c_CpA?v_pAct ;
```

```
10:       v_OrAct := v_pAct or v_newAct ;          —effectue un OU logique des activites—
11:       c_isActive!v_OrAct , c_NpA!v_pAct ,
12:       c_NcA!v_newAct                           —memorise l'activite de la colonne—
13:     ]
14:   ]
15: ]
```

Le calcul combinatoire est réalisé aux lignes 4 et 10. Une implantation plus astucieuse permettrait certainement d'éviter l'utilisation de deux portes distinctes pour un même calcul.

Bibliographie

[ABOR98] Airiau (Roland), Bergé (Jean-Michel), Olive (Vincent) et Rouillard (Jacques). – *VHDL langage, modélisation, synthèse*. – Presses polytechniques et universitaires romandes et CNET-ENST, 1998, collection technique et scientifique des télécommunications édition. 153

[All99] Alleysson (David). – *Le traitement du signal chromatique dans la rétine : un modèle de base pour la perception humaine des couleurs*. – Thèse de PhD, UJF - Grenoble, 1999. `http://cepax6.lis.inpg.fr/these/Th_Alleysson.html`. 8

[Alt96] Altunbasak (Yucel). – *Object-Scalable, Content-based video representation and motion tracking for visual communications and multimedia*. – New York, Thèse de PhD, Université de Rochester, 1996. 14

[ANB01] Anwander (A.), Neyran (B.) et Baskurt (A.). – Gradient couleur multiéchelle pour la segmentation d'images. *Traitement du Signal*, vol. 18, n° 2, 2001. 19, 21

[ANR74] Ahmed (N.), Natarajan (T.) et Rao (K.R.). – Discrete cosine transform. *IEEE Transactions on Computers*, janvier 1974. 13

[Bau01] Baude (Françoise). – *Systèmes d'exploitation et Applications Répartis*. – Note de cours, Université de Nice et Sophia Antipolis, 2001. `www-mips.unice.fr/~baude/Systemes-Distribues/index.html`. 71, 73

[BB94] Beucher (Serge) et Bilodeau (Michel). – Road segmentation using a fast watershed algorithm. *Dans : Proc. ISMM 94 : Mathematical Morphology and its Applications to Image Processing*. pp. 29–30, septembre 1994. – Fontainebleau, septembre 1994. 20

[Beu90] Beucher (Serge). – *Segmentation d'images et morphologie mathématique*. – Thèse de PhD, Ecole Nationale des Mines de Paris, 1990. `http://cmm.ensmp.fr/~beucher/publi/SB_these.pdf`. 22, 27, 35, 113, 187, 188, 190, 196

[Beu94] Beucher (Serge). – Watershed, hierarchical segmentation and waterfall algorithm. *Dans : Proc. Mathematical Morphology and its Applications to Image Processing*, éd. par Serra (Jean) et Soille (Pierre). pp. 69–76, September 1994. – Kluwer Ac., Nld. 22, 94, 171

[Beu00] Beucher (Serge). – Ligne de partage des eaux et segmentation hiérarchique. – `http://cmm.ensmp.fr/~beucher/slideshow/cours2000fr.htm`, 2000. 25

[BL79] Beucher (S.) et Lantuejoul (C.). – Use of watershed in contour detection. *International Workshop on Image Processing : Real-time Edge and*

Motion detection/estimation, September 1979. – `http://cmm.ensmp.fr/~beucher/publi/watershed.pdf`. 22

[BLDS01] Buisson (Alexandre), Laurent (Nathalie), Demaret (Laurent) et Sentieys (Olivier). – Evaluation de la complexité et optimisation des algorithmes de codage vidéo par maillage. *Dans : CORESA*, novembre 2001. 15

[BM98] Bieniek (Andreas) et Moga (Alina). – A connected component approach to the watershed segmentation. *Mathematical Morphology and its Applications to Image and Signal Processing*, vol. 12, 1998, pp. 215–222. 24, 190, 194, 196

[BM00] Bieniek (Andreas) et Moga (Alina). – An efficient watershed algorithm based on connected components. *Pattern recognition*, vol. 6, n° 33, 2000, pp. 907–916. 26

[BN97] Bernard (T.M.) et Nguyen (R.). – VAMPIRE : Guidage d'un véhicule miniature par un système de vision à rétine programmable. *Dans : Proc. Conf. Rétine electronique, ASIC-FPGA et DSP pour la vision et le traitement d'image en temps réel*, éd. par Ni (Y.), pp. 1–6, mai 1997. – Evry, France, mai 1997. `www.ensta.fr/~tbernard/Publis/1997/read/Index.html`. 147

[Bon98] Bonnaud (Laurent). – *Schémas de suivi d'objets vidéo dans une séquence animée :application à l'interpolation d'images intermédiaires.* – Thèse de PhD, Université de Rennes 1, 1998. 21

[Bra95] Braviano (Gilson). – *Logique floue en segmentation d'images : seuillage par entropie et structures pyramidales irrégulières.* – Thèse de PhD, TIMC-IMAG, Grenoble, 1995. `http://www-mediatheque.imag.fr/Mediatheque.IMAG/collections-electroniques/publications/theses/Mathdoc/theses-ftp/1995/Braviano.Gilson/notice-francais.html`. 21

[Bre99] Brelot (Marc). – *Représentations orientées-objets de scènes visuelles pour la composition flexible.* – Thèse de PhD, INP Grenoble, 1999. 12, 13, 15

[Bri95] Brigger (Patrick). – *Morphological shape representation using skeleton decomposition : application to image coding.* – Thèse de PhD, Ecole polytechnique fédérale de Lausanne, 1995. 13

[BT89] Bertsekas (Dimitri P.) et Tsitsiklis (John N.). – *Parallel and distributed computation.* – Prentice-Hall, 1989, *numerical methods-Englewood Cliffs.* 3, 31

[BZD93] Bernard (Thierry), Zavidovique (Bertrand) et Devos (Francis). – A programmable artifical retine. *IEEE Journal of solid-state circuits*, vol. 28, n° 7, Juillet 1993, pp. 789–798. 147

[Cap98] Capello (Franck). – *Une introduction aux architectures parallèles.* – Notes de cours, LRI, Université Paris-Sud, Orsay, 1998. `penarvir.univ-brest.fr/~llagadec/ENSEIGNEMENT/ARCHIPAR/COURS/cours1.html`. 70

[Car99] Carissimi (Alexandre da Silva). – *ATHAPASCAN-0 : exploitation de la multiprogrammation légère sur grappes de multiprocesseurs.* – Grenoble,

Thèse de PhD, Laboratoire LMC-IMAG et ID-IMAG, 1999. www-id.imag.fr/Laboratoire/Theses. 71

[CJ98] Corbel (Annie) et Jegou (Roland). – *Algorithmique séquentielle et parallèle.* – Notes de cours, Ecole des Mines de Saint Etienne, 1998. www.emse.fr/~corbel/PARALLEL/asp.ps. 71

[CP96] Cocquerez (Jean-Pierre) et Philipp (Sylvie). – *Analyse d'images : filtrage et segmentation.* – MASSON, Février 1996. 21

[DC99] Delmas (Patrice) et Coulon (Pierre-Yves). – Automatic snakes for robust lip boundaries extraction. *Dans : IEEE International Conference On ACOUSTICS, SPEECH, AND SIGNAL PROCESSING*, mars 1999. – http://www.citr.auckland.ac.nz/~patrice/pub/icassp99.ps.gz. 13

[DDR+01] Dinh Duc (Anh Vu), , Renaudin (Marc), Rigaud (Jean-Baptiste) et Rezzag (Amine). – *DTL : The foundations of a general design methodology for asynchronous circuits.* – Rapport technique, Grenoble, France, TIMA Laboratory, 2001. 153, 156

[DDRR01] Dinh Duc (Anh Vu), , Renaudin (Marc) et Rigaud (Jean-Baptiste). – *CHP syntax.* – Rapport technique, Grenoble, France, TIMA Laboratory, 2001. 153

[DDRR+02] Dinh Duc (Anh Vu), Rigaud (Jean-Baptiste), Rezzag (Amine), Sirianni (Antoine), Fragoso (Joao), Fesquet (Laurent) et Renaudin (Marc). – Tast cad tools. *Dans : Second ACiD-WG Workshop, Munich, Allemagne*, Janvier 2002. 152, 153

[Del00] Delmas (Patrice). – *Extraction des contours de lèvres d'un visage parlant par contours actifs : Application à la communication multimodale.* – Thèse de PhD, INPG - Grenoble, 2000. http://cepax6.lis.inpg.fr/these/Th_Delmas.html. 13

[DM01] Ducourthial (Bertrand) et Mérigot (Alain). – *Parallel asynchronous computations for image analysis.* – Rapport technique, IEF, université Paris-Sud, 2001. Rapport interne, http://www.hds.utc.fr/~ducourth/BIBLIO/rap_par_async_im.ps. 40, 183

[D.S00] D.Schroeder (Marc). – Jpeg compression algorithm and associated data structures. – http://www.cs.und.edu/~mschroed/jpeg.html, 2000. 11

[Duc99] Ducourthial (Bertrand). – *Un modèle de programmation à parallélisme de données, pour algorithmes et données irréguliers, à primitive de calcul asynchrones.* – Thèse de PhD, Univ. Paris-Sud — U.F.R. scientifique d'Orsay, 1999. 39, 40, 42

[Dul96] Dulac (Didier). – *Contribution au parallélisme massif en analyse d'image : une architecture SIMD fondée sur la reconfigurabilité et l'asynchronisme.* – Thèse de PhD, Univ. Paris-Sud — U.F.R. scientifique d'Orsay, 1996. 39, 40, 73, 93

[DVG94] Dobrin (Bogdan P.), Viero (Timo) et Gabbouj (Moncef). – Fast watershed algorithms :analysis and extensions. *SPIE on Non linear image processing*, vol. 2180, 1994, pp. 209–220. 22

[Fas01] Fassino (Jean-Philippe). – *THINK : vers une architecture de systèmes flexibles*. – Thèse de PhD, École Nationale Supérieure des Télécommunications, Decembre 2001. `sardes.inrialpes.fr/papers/phd.php`. 76

[Fau95] Fauvel (Marc-Noël). – *Contribution à la stéréovision : Une approche connexionniste*. – Thèse de PhD, Université de Paris VI, spécialité Robotique, 1995. `perso.wanadoo.fr/mnfauvel/Publications`. 75

[Fly72] Flynn (M.). – Some computer organizations and their effectiveness. *IEEE Trans. Comput.*, vol. C-21, 1972, p. 94. 70

[Fos94] Foster (Ian). – *Designing and building parallel programs*. – Addison-Wesley Publishing Company, 1994. `www-unix.mcs.anl.gov/dbpp`. 109

[Fre74] Freeman (Herbert). – Computer processing of line-drawing images. *ACM Computing Surveys*, vol. 6, n° 1, 1974, pp. 57–97. 13

[FSR02] Fragoso (João Leonardo), Sicard (Gilles) et Renaudin (Marc). – Générateur d'opérateurs arithmétiques asynchrones. *Dans : 5ème Journées Nationales du Réseau Doctoral de Microélectronique (JNRDM)*, Avril 2002. – Grenoble, Avril 2002. 161

[GDB00] Guezguez (Saloua), Dulac (Didier) et Bertrand (Gilles). – Parallel segmentation based on topology with the associative net model. *Proceedings of the Fifth IEEE International Workshop on Computer Architectures for Machine Perception (CAMP)*, 2000. 22

[GDM98] Guezguez (Saloua), Dulac (Didier) et Mérigot (Alain). – Parallélisation d'une segmentation par ligne de partage des eaux sur la maille associative d'orsay. *Dans : Rencontres Francophones de l'Intelligence Artificielle, RFIA'98*, 1998. 33, 40, 183

[Gel98] Gelgon (Marc). – *Segmentation spatio-temporelle et suivi dans une séquence d'images : application à la structuration et à l'indexation vidéo*. – Thèse de PhD, Rennes I, UFR Structure et Propriétés de la Matière, 1998. 16, 18

[GER00] G.Milanova (Mariofanna), Elmaghraby (Adel) et Rubin (Stuart). – Cellular neural networks for segmentation of image sequence. *Dans : 11th Portuguese Conference on Pattern Recognition*, éd. par A. (Campilho) et Mendonça (A.), pp. 49–54, 2000. – `http://www.louisville.edu/speed/emacs/mrl/publications/`. 21, 32

[GHP+95] Graffigne (Christine), Heitz (Fabrice), Pérez (Patrick), Prêteux (Françoise), Sigelle (Marc) et Zerubia (Josiane). – Hierarchical markov random field analysis models applied to image analysis : a review. *Dans : Proc. of the Conf. on Neural, Morphological and Stochastic Methods in Image Processing*. SPIE's International Symposium on Optical Science, Engineering and Instrumentation, 9-14 Juillet 1995. – San Diego, 9-14 Juillet 1995. `http://tsi.enst.fr/~sigelle/publis2-eng.html`. 32

[GL98] Gu (Chuang) et Lee (Ming-Chieh). – Semiautomatic segmentation and tracking of semantic video objects. *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, n° 5, septembre 1998, pp. 572–584. 17, 21

[GM00] Gomila (C.) et Meyer (F.). – Automatic video object generation tool : segmentation and tracking of persons in real time. *Annales des télécommunications*, vol. 55, n° 3-4, mars-avril 2000, pp. 172–183. 17, 20, 183

[Gom01] Gomila (Cristina). – *Mise en correspondance de partitions en vue du suivi d'objets.* – Thèse de PhD, Ecole nationale supérieure des mines de Paris, 2001. 10, 18, 19, 20, 27, 183, 189

[GS98] Garrido (Luis) et Salembier (Philippe). – Region based analysis of video sequences with a general merging algorithm. *Dans : eusipco*, 1998. 18

[Gu95] Gu (Chuang). – *Multivalued morphology and segmentation-based coding.* – Thèse de PhD, Ecole polytechnique fédérale de Lausanne, 1995. `http://ltswww.epfl.ch/text/publications/phd.html`. 8, 13, 21

[Hau95] Hauck (Scott). – Asynchronous design methodologies : An overview. *Proceedings of the IEEE*, vol. 83, n° 1, Janvier 1995, pp. 69–93. – `http://paradise.ucsd.edu/class/ece284`. 153

[Hoa78] Hoare (C.A.R.). – Communicating sequential processes. *Commun. ACM 21*, no8, Août 1978, pp. 666–677. 152

[Hoa85] Hoare (C.A.R.). – Communicating sequential processes. *Prentice Hall International Series in Computer Science*, 1985. 152

[Hur89] Hurlbert (Anya C.). – *The computation of color.* – Rapport technique n° 1154, MIT Artificial Intelligence Laboratory, Septembre 1989. 20

[KB99] Kerfoot (Ian B.) et Bresler (Yoram). – Theoretical analysis of multispectral image segmentation criteria. *IEEE TRANSACTIONS ON IMAGE PROCESSING*, vol. 8, n° 6, juin 1999, pp. 798–820. 21, 32

[KH02] Kim (Changick) et Hwang (Jenq-Neng). – Fast and automatic video object segmentation and tracking for content-based applications. *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, n° 2, février 2002. 17

[KS98] Kauff (Peter) et Schüür (Klaas). – A shape-adaptive dct with block-based dc-separation and delta-dc-correction. *IEEE Trans. CVST*, vol. 8, n° 3, juin 1998, pp. 237–242. 14

[KS99] Konrad (Janusz) et Stasiński (Ryszard). – A new class of fast shape-adaptive orthogonal transforms and their application to region-based image compression. *IEEE Trans. on Circuits Syst. Video Technologie*, vol. 9, février 1999, pp. 16–34. – `http://www.inrs-telecom.uquebec.ca/users/konrad/publications.html`. 14

[Lau98] Laurent (Christophe). – *Conception d'algorithmes parallèles pour le traitement d'images utilisant la morphologie mathématique : Application à la segmentation d'images.* – Thèse de PhD, Université Bordeaux I, 1998. 33, 83

[LC02] Lei (Ming-Han) et Chiueh (Tzi-Dar). – An analog motion field detection chip for image segmentation. *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, n° 5, Mai 2002. 21, 32

[LD91] Lu (Cheng-Chang) et Dunham (James-George). – Highly efficient coding schemes for contour lines based on chain code representations. *IEEE Transactions on Communications*, vol. 39, n° 10, octobre 1991, pp. 1511–1514. 13

[LD99] Luthon (Franck) et Dragomirescu (Daniela). – A cellular analog network for mrf-based video motion detection. *IEEE transactions on circuits and systems : fundamental theory and applications*, vol. 46, n° 2, Février 1999, pp. 281—293. 21, 32

[Lec99] Lechat (Patrick). – *Représentation et codage de séquences vidéo par maillages 2D déformables.* – Thèse de PhD, Rennes I, TdS et Télécommunications, 1999. 14

[Lem96] Lemonnier (Fabrice). – *Architecture électronique dédiée aux algorithmes rapides de segmentation basés sur la Morphologie Mathématique.* – Thèse de PhD, Ecole Nationale des Mines de Paris, 1996. 33, 119

[Lev02] Levner (Ilya). – Shape detection, analysis and recognition. *CMPUT 615*, Avril 2002. – `http://www.cs.ualberta.ca/~ilya/courses/c615-PerceptualSystems`. 17

[Lie00] Lievin (Marc). – *Analyse entropico-logarithmique de séquences vidéo couleur : application à la segmentation markovienne et au suivi de visages parlants.* – Thèse de PhD, INP Grenoble, 2000. 18, 19, 20, 21

[LLS99] Laurent (Nathalie), Lechat (Patrick) et Sanson (Henri). – Scalable image coding with fine granularity based on hierarchical mesh. *Dans : Visual Communications and Image Processing*, pp. 1130–1142, 1999. 14

[LMM02] Louchnikova (Tatiana) et Marchand-Maillet (Stéphane). – Flexible image decomposition for multimedia indexing and retrieval. *Dans : Proceedings of SPIE Photonics West, Electronic Imaging 2002, Internet Imaging III*, 2002. – San Jose, USA, 2002. `http://viper.unige.ch/segmentation`. 17

[Lop00] Lopez (Pierre). – Cours de graphes. – `http://www.laas.fr/~lopez/cours/GRAPHES/Spedago.html`, Octobre 2000. 177

[LS00] Lecussan (Bernard) et Sainrat (Pascal). – *L'architecture du noeud de la grappe : état de l'art et prospective.* – Rapport technique, Ecole d'hiver iHPerf 2000 : Applications Haute Performances, 2000. `www.irisa.fr/iHPerf2000/documents.html`. 71

[Mal89] Mallat (S.G.). – Multifrequency channel decompositions of images and wavelet models. *IEEE Trans. on ASSP*, vol. 37(12), December 1989, pp. 2091–2110. 14

[Mar91] Martin (Alain). – *Synthesis of Asynchronous VLSI Circuits.* – Rapport technique, Caltech Computer Science Technical Reports, 1991. `http://resolver.library.caltech.edu/caltechCSTR:1991.cs-tr-93-28`. 152

[Mar96] Marcotegui (Beatriz). – *Segmentation de Séquences d'Images en Vue du Codage.* – Thèse de PhD, Ecole nationale supérieure des mines de Paris, 1996. `http://cmm.ensmp.fr/~marcoteg/these.html`. 13, 16, 18, 20, 21, 188, 190

[MB00] Moga (Alina) et Bieniek (Andreas). – An efficient watershed algorithm based on connected components. *Patttern Recognition*, vol. 6, n° 33, 2000, pp. 907–916. 33, 51, 196

[Mey87] Meyer (Fernand). – Algorithmes séquentiels. *Dans : 11ème GRETSI*, juin 1987. – Nice, France, juin 1987. 23, 187

[Mey91] Meyer (Fernand). – Un algorithme optimal de ligne de partage des eaux. *Dans : Actes du 8ème ccongrès AFCET*, pp. 847–859, 1991. – Lyon-Villeurbane, France, 1991. 22, 33, 111, 115, 196

[Mey94] Meyer (Fernand). – Topographic distance and watershed lines. *Signal processing*, no38, 1994, pp. 113–125. 22, 23, 24, 25, 26, 34, 36, 50, 190, 192, 195, 196

[Mey98] Meyer (Fernand). – Levelings for image simplification and segmentation. – Juin 1998. `http://cmm.ensmp.fr/Recherche/pages/pdf/leveling.pdf`. 189

[Mey00] Meyer (Fernand). – Nivellements et zones plates. – 2000. Support de cours, Ecole d'été du centre de morphologie mathématique. 189

[Mit97] Mitchell (Joan L.). – *MPEG Digital video compression standard*. – W.B. PENNEBAKER. - New York : Van Nostrand Reinhold, 1997. 13

[MN98] Meier (Thomas) et Ngan (King N.). – Automatic segmentation of moving objects for video object plane generation. *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, n° 5, septembre 1998. 16

[Mog97] Moga (Alina). – *Parallel Watershed Algorithms for Image Segmentation*. – Thèse de PhD, Tampere University of Technology, Finland, 1997. 22, 24, 32, 33, 83, 109

[MPE] The MPEG home page. – `http://mpeg.telecomitalialab.com`. 2, 12, 13

[MPE95] Iso/iec/jtc1/sc29/wg11, Juillet 1995. MPEG-4 Proposal Package Description (PPD). 12

[MPE02] Mpeg-4 overview - (v.21 - jeju version), mars 2002. Editor : Rob Koenen, `http://mpeg.telecomitalialab.com/standards/mpeg-4/mpeg-4.htm`. 2, 12

[Mér92] Mérigot (Alain). – *Associative nets : a new parallel computing model*. – Rapport technique, IEF, université Paris-Sud, 1992. `http://www.ief.u-psud.fr/~sicard/OLD/ANET/anet-anet.html`. 39, 41

[MR95] Meijster (Arnold) et Roerdink (Jos B.T.M.). – A proposal for the implementation of a parallel watershed algorithm. *CAIP'95 proceedings*, 1995, pp. 790–795. xi, 28, 29

[Mér97] Mérigot (Alain). – Associative nets : A graph-based parallel computing model. *IEEE Transactions on Computers*, vol. 46, n° 5, mai 1997, pp. 558–571. 39

[MR98] Meijster (Arnold) et Roerdink (Jos B.T.M.). – A disjoint set algorithm for the watershed transform. *Dans : EUSIPCO*, 1998. – `www.cs.rug.nl/~roe/publications/eusipco.html`. 24, 196

[ND97] Nowick (Steve) et Davis (Al). – *An introduction to asynchronous circuit design*. – Rapport technique, Université d'Utah (Salt Lake City, USA), 1997. `www.cs.man.ac.uk/async/background/index.html`. 78, 81, 151, 153

[Nog98] Noguet (Dominique). – *Architectures parallèles pour la morphologie mathématique géodésique*. – Thèse de PhD, INPGrenoble, 1998. 22, 33, 34, 115, 194

[NS96] Najman (Laurent) et Schmitt (Michel). – Geodesic saliency of watershed contours and hierarchical segmentation. *IEEE Trans. Pattern Anal. & Machine Intell.*, vol. Vol.18, n° No.12, December 1996, pp. 1163–1173. – `http://www.laurentnajman.org/papers.htm`. 195

[NYK+02] Nakayama (Hiroshi), Yoshitake (Toshiyuki), Komazaki (Hiroshi), Watanabe (Yasuhiro), Araki (Hisakatsu), Morioka (Kiyonori), Li (Jiang), Peilin (Liu), Lee (Shinhaeng), Kubosawa (Hajime) et Otobe (Yukio). – An MPEG-4 video LSI with an error-resilient codec core based on a fast motion estimation algorithm. *Dans : IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 368–369, 2002. 2

[OP98] Ojanperä (Tero) et Prasad (Ramjee). – An overview of air interface multiple access for IMT-2000/UMTS. *IEEE Communications magazine*, september 1998, pp. 82–95. 2

[Pat98] Pateux (Stéphane). – *Segmentation spatio-temporelle et codage orienté-régions de séquences vidéo basés sur le formalisme MDL*. – Thèse de PhD, Rennes I, informatique, TdS et Télécommunications, 1998. 16, 18, 32

[Pel02] Pelissier (Patrick). – *Algorithme massivement parallèle asynchrone de segmentation d'images suivant un critère de volume*. – Rapport technique, France Telecom R&D, Juin 2002. Rapport de stage ingénieur ENSERG – Grenoble. 170

[PHKU01] Petrou (Maria), Hou (Peixin), Kamata (Sei-ichiro) et Underwood (Craig Ian). – Region-based image coding with multiple algorithms. *IEEE Transactions on Geoscience and Remote Sensing*, vol. 39, n° 3, mars 2001, pp. 562–570. 14

[Pla94] Planet (Patricia). – *Algorithmes et architectures cellulaires pour le traitement d'images*. – Thèse de PhD, INP Grenoble, 1994. 91

[Ren00] Renaudin (Marc). – Asynchronous circuits and systems : a promising design alternative. *Microelectronic Engineering*, vol. 54, n° 1–2, December 2000, pp. 133–149. – Elsevier. 79, 81, 151, 153, 164

[Rez02] Rezzag (Amine). – Méthodologie de synthèse logique de circuits asynchrones micropipeline. *Dans : 5ème Journées Nationales du Réseau Doctoral de Microélectronique (JNRDM)*, Avril 2002. – Grenoble, Avril 2002. 153

[RM01] Roerdink (Jos B.T.M.) et Meijster (Arnold). – The watershed transform : Definitions, algorithms and parallelization strategies. *Fundamenta Informaticae*, no41, 2001, pp. 187–228. – IOS Press. 22, 24, 32, 195

[Rob97] Robin (Frédéric). – *Etude d'architecture VLSI numérique parallèles et asynchrones pour la mise en œuvre de nouveaux algorithmes d'analyse et rendu d'images*. – Thèse de PhD, Ecole nationale supérieure des télécommunications de Paris, 1997. 34, 67, 73, 76, 79, 81, 83, 84, 85, 93, 128, 131, 157, 168, 183

[Rou02] Roux (Sébastien). – *Adéquation algorithme-architecture pour le traitement multimédia embarqué*. – Grenoble, Thèse de PhD, INPG, 2002. `http://tima.imag.fr/news/defense/all_theses.asp`. 12, 15, 19, 20, 21

[RQFR01] Rigaud (Jean-Baptiste), Quartana (Jérôme), Fesquet (Laurent) et Renaudin (Marc). – Modeling and design of asynchronous priority arbiters for on-chip communication system. *Dans : IFIP International Conference On Very Large Scale Integration (VLSI-SOC'01)*, Décembre 2001. – Le Corum, Montpellier, France, Décembre 2001. 158

[RVR97] Renaudin (Marc), Vivet (Pascal) et Robin (Frédéric). – AAAA : asynchronisme et adéquation algorithme architecture. *Traitement du Signal*, vol. 14, n° 6, 1997. 157

[RVR99] Renaudin (Marc), Vivet (Pascal) et Robin (Frédéric). – ASPRO : an asynchronous 16-bit RISC microprocessor with DSP capabilities. *Proceedings of the 25th european solid-state circuits conference, ESSCIRC'99*, September 1999, pp. 428–431. – Duisburg, Germany. 161

[R.Z96] R.Zaiane (Osmar). – Cours de la sfu school of computing science : 4.2. video compression. – `http://fas.sfu.ca/cs/undergrad/CourseMaterials/CMPT479/material/notes/Chap4/Chap4.2/Chap4.2.html`, Juin 1996. Trouvé sur internet. 11

[SBJ98] Shi (Zhuoer), Bao (Zheng) et Jiao (Licheng). – Arbitrary shape wavelet packet image processing. *soumis à IEEE Transactions on Image Processing*, 1998. – `http://www.uh.edu/~zshi/publications`. 14

[SDS96] Stollnitz (Eric J.), DeRose (Tony D.) et Salesin (David H.). – *Wavelets for computer graphics (Theory and Applications)*. – Morgan Kaufmann, San Francisco, 1996. 14

[Ser82] Serra (Jean). – *Image Analysis and Mathematical Morphology*. – Academic Press, 1982. 184

[Ser88] Serra (Jean). – *Image Analysis and Mathematical Morphology. Theoretical Advances*. – Academic Press, 1988. 22, 184

[Ser00] Serra (Jean). – Cours de morphologie mathématique. – `http://cmm.ensmp.fr/~serra/cours/index.htm`, 2000. xii, 22, 80, 184, 187, 193

[Sic99] Sicard (Gilles). – *De la biologie au silicium : une rétine bio-inspirée analogique pour un capteur de vision «intelligent» adaptatif*. – Thèse de PhD, INP Grenoble–Microélectronique, 1999. 147

[Sik95] Sikora (Thomas). – Low complexity shape-adaptive dct for coding of arbitrarily shaped image segments. *Signal Processing : Image Communication*, vol. 7, 1995, pp. 381–395. 14

[Sir02] Sirianni (Antoine). – *Le compilateur CHP2VHDL*. – Rapport technique, Grenoble, Laboratoire TIMA, 2002. 153

[SK02] Schreer (O.) et Kauff (P.). – An immersive 3d video-conferencing system using shared virtual team user environments. *Dans : ACM Collaborative Environments, CVE*, Sept./Oct. 2002. – Bonn, Germany, Sept./Oct. 2002. `http://bs.hhi.de/SPAG/SPAG-publications.htm`. 12

[SM99] Salembier (Philippe) et Marqués (Ferran). – Region-based representations of image and video : Segmentation tools for multimedia services. *IEEE transactions on circuits and systems for video technology*, vol. 9, n° 8, décembre 1999, pp. 1147–1167. – `http://gps-tsc.upc.es/imatge/pub/Journal_data.html`. 17, 18, 21

[SP01] Sohn (Young Wook) et Park (Rae-Hong). – Lot coding for arbitrarily shaped object regions. *IEEE Transactions on Image Processing*, vol. 10, n° 2, février 2001, pp. 317–322. – Correspondence. 14

[SS02] Shishikui (Yoshiaki) et Sakaida (Shinichi). – Region support dct (RS-DCT) for coding of arbitrarily shaped texture. *IEEE Transactions on circuits and systems for video technology*, vol. 12, n° 5, Mai 2002. 14

[Sys] SystemC home page : a modeling platform that enables, promotes and accelerates system-level co-design and IP exchange. – `http://www.systemc.org`. 87

[SZC+00] Szirányi (TamaÂs), Zerubia (Josiane), Czúni (LaÂszLó), Geldreich (David) et Kato (ZoltaÂn). – Image segmentation using markov random field model in fully parallel cellular network architectures. *Real-Time Imaging*, vol. 6, n° 3, 2000, pp. 195–211. – `http://www.sztaki.hu/~sziranyi/publ_szt.html`. 21, 32

[Thi94] Thiel (Tamiko). – The design of the connection machine. *DesignIssues*, vol. 10, n° 1, 1994. – `http://mission.base.com/tamiko/cm/cm-articles.html`. 75

[Tro94] Tron (Cécile). – *Modèles quantitatifs de machines parallèles : les réseaux d'interconnexion.* – Thèse de PhD, Institut National Polytechnique de Grenoble, Novembre 1994. `http://www-id.imag.fr/Laboratoire/Theses`. 72, 76

[Vac95] Vachier (Corinne). – *Extraction de Caractéristiques, Segmentation d'Image et Morphologie Mathématique.* – Thèse de PhD, Ecole nationale supérieure des mines de Paris, 1995. 19, 21, 103, 170, 184

[VBMZ+99] Van Beek (Peter), Murat (Tekalp A.), Zhuang (Ning), Celasun (I sl) et Xia (Minghui). – Hierarchical 2-d mesh representation, tracking and compression for object-based video. *IEEE Trans. On Circuits and Syst. For Video technol.*, vol. 9, n° 2, septembre 1999, pp. 353–369. 14

[VD94] Van Droogenbroeck (Marc). – *Traitement d'images numériques au moyen d'algorithmes utilisant la morphologie mathématique et la notion d'objet : application au codage.* – Thèse de PhD, Université catholique de Louvain, 1994. 15, 21

[VdSD01] Van der Steen (Aad J.) et Dongarra (Jack J.). – *Overview of Recent Supercomputers.* – Rapport technique, Université d'Utrecht, Hollande, Dept. of Computational Physics, 2001. `http://www.euroben.nl/reports/INDEX`, fichier overview01.ps.gz. 71

[Vin92] Vincent (Luc). – Morphological area openings and closings for grayscale images. *Proc.NATO Shape in Picture Workshop, Driebergen. Springer Verlag*, 1992. 189

[Vin93] Vincent (Luc). – Morphological grayscale reconstruction in image analysis : Applications and efficient algorithms. *IEEE Transactions on Image Processing*, vol. 2, n° 2, April 1993. 35, 114, 191

[Viv01] Vivet (Pascal). – *Une méthodologie de conception de circuits intégrés quasi-insensibles aux délais : application à l'étude et à la réalisation d'un processeur RISC 16-bit asynchrone.* – Thèse de PhD, INPG, Grenoble, 2001. `http://tima.imag.fr/publications/theses.asp`. 79, 81, 91, 153, 155, 161, 199

[VS91] Vincent (Luc) et Soille (Pierre). – Watershed in digital spaces : An efficient algorithm based on immersion simulations. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, vol. 13, n° 6, June 1991, pp. 583–598. 27

[Wan98] Wang (Demin). – Unsupervised video segmentation based on watersheds and temporal tracking. *IEEE Transactions On Circuits and Systems For Video technology*, vol. 8, n° 5, septembre 1998, pp. 539–546. 21, 188

[Wan01] Wang (Demin). – Unsupervised segmentation of color-texture regions in images and video. *Dans : PAMI*, 2001. – `http://maya.ece.ucsb.edu/JSEG`. 20, 21

[XLLZ01] Xing (Guiwei), Li (Jin), Li (Shipeng) et Zhang (Ya-Qin). – Arbitrarily shaped video-object coding by wavelet. *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, n° 10, octobre 2001, pp. 1135–1139. 14

[YJA02] Yang (Ming-Hsuan), J. Kriegman (David) et Ahuja (Narendra). – Detecting faces in images : a survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, n° 1, Janvier 2002, pp. 34–58. 19

[YM01] Yining et Manjunath (B.S.). – Unsupervised segmentation of color-texture regions in images and video. *Dans : PAMI*, 2001. – `http://maya.ece.ucsb.edu/JSEG/`. 20, 21

[YMS99] Yining, Manjunath (B.S.) et Shin (Hyundoo). – Color image segmentation. *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 1999. – `http://maya.ece.ucsb.edu/JSEG/`. 20

[ZMM99] Zanoguera (Francisca), Marcotegui (Beatriz) et Meyer (Fernand). – A toolbox for interactive segmentation based on nested partitions. *ICIP (Kobe-Japon)*, 1999. – `http://cmm.ensmp.fr/~marcoteg/momusys/vogue`. 17, 183

Index et liste des symboles

Index

Liste des symboles

Sigles et abbréviations

$\neg$	inversion logique d'un bit, page 112
$\star$	données requises par un algorithme, page 112
ASIC	*Application Specific Integrated Circuits* : circuit microélectronique dédié, page 141
ASPRO	*ASynchronous PROcessor* : processeur RISC asynchrone 16 bits, page 161
CB	*Catchment Bassin* : bassin d'attraction associé à un minimum, page 196
CHP	*Communicating Hardware Processes*, page 152
CIF	Acronyme[1] de *Common Intermediate Format* : format d'images (352×288 pixels), page 2
CNN	*Cellular Neuronal/Nonlinear Network* : réseau de neurones analogique, page 32
CSP	*Communicating Sequential Processes*, page 152
DCT	*Discret Cosine Transform* : transformée en cosinus discret, page 11
DSP	*Digital Signal Processor* : processeur dédié au traitement du signal, page 110
DTL	*Data Transfer Level* : niveau de programmation pour la synthèse de circuits asynchrones, page 153
DWT	*Discret Wavelet Transform* : transformée en ondelettes discrètes, page 14
FAH	File d'Attente Hiérarchique. Ensemble de FIFO ordonnées par leur priorité, page 33
FIFO	*First In First Out* : file d'attente, page 77
GP	*Generate/Propagate* : un des deux processus destiné à la soustraction, page 162
GSM	*Global System for Mobile communication*, page 2

[1] Acronymes utilisés dans la visiocommunication `http://www.aurif.fr/visio-u_acronymes.html`

HDL *Hardware Description Language* : langage de prototypage matériel, page 87

JPEG *Joint Picture Expert Group*, page 11

JSEG Méthode de segmentation couleur. Les valeurs d'une "J-image" correspondent à un critère d'homogénéité basé texture, page 21

LPE Ligne de Partage des Eaux ou *Watershed.*, page 195

LPGS Localement Parallèle Globalement Séquentiel : granularité intermédiaire, page 84

LSGP Localement Séquentiel Globalement Parallèle : granularité intermédiaire, page 84

MDL *Minimum Description Length*, page 32

MIMD *Multiple Instruction stream, Multiple Data stream* : parallélisme de processus), page 71

MISD *Multiple Instruction stream, Single Data stream* : parallélisme de flux, page 71

MMX *MultiMedia eXtended* : instruction permettant de paralléliser les traitements en encapsulant plusieurs données pour une seule instruction, page 71

MPEG *Moving Picture Expert Group*, page 12

MRA Modèle de Réseau Associatif, page 40

MRF *Markov Random Field* : Champs aléatoires de Markov, page 32

QCIF Quart-CIF : format d'images standard pour les visiophones (176×144 pixels), page 2

QDI *Quasi Delay Insensitive* : circuit asynchrone où la fourche isochrone est supposée vérifiée, page 153

RAM *Random Access Memory* : mémoire volatile, page 5

RGB *Red Green Blue* : espace de représentation de la couleur, utilisé pour les images vidéo, à partir des trois couleurs Rouge, Vert et Bleue, page 8

RISC *Reduced Instruction Set Computer*, page 110

RS-DCT *Region Support-DCT* : DCT suivant le support de la région (fenêtrage), page 14

RTL *Register Transfert Level* : niveau de programmation pour la synthèse de circuits synchrones, page 153

SA-DCT *Shape Adaptive-DCT* : DCT s'adaptant à la forme de la région, page 14

SA-DWT *Shape Adaptive-Discret Wavelet Transform* : DWT s'adaptant à la forme de la région, page 14

SC *Sub/Carry* : un des deux processus destiné à la soustraction, page 162

SIMD *Single Instruction stream, Multiple Data stream :* parallélisme de données, page 71

SISD *Single Instruction stream, Single Data stream*, page 71

SP Accélération (implantation parallèle), page 132

SPMD *Single Program stream, Multiple Data stream* : particularisation du modèle MIMD, page 73

SQCIF Sub-QCIF : format d'images standard pour les visiophones (88×72 pixels), page 2

T Temps d'exécution (implantation parallèle), page 132

TAST TIMA *Asynchronous digital circuit Synthesis Tools*, page 152

UMTS *Universal Mobile Telecommunications Service* (débits atteignant 384 kbits/s en voix descendante et 64 kbits/s en voix montante), page 2

VHDL *Very high scale integration Hardware Description Language*, page 153

VLSI *Very Large Scale Integration circuit* : circuit microélectronique à forte densité d'intégration, page 78

YUV Luminance Y, chrominance rouge U, chrominance bleue V : espace de représentation de la couleur où l'intensité lumineuse est décorrélée des composantes de couleur, page 8

Notations CHP - Microélectronique

@ Garde déterministe, page 153

@@ Garde indéterministe, page 153

, Opérateur de composition parallèle, page 153

; Opérateur de composition séquentielle, page 153

C Préfixe d'un canal véhiculant l'état courant (*Current*) d'une donnée, page 155

E Canal d'entrée, page 155

N Préfixe d'un canal véhiculant l'état suivant (*Next* d'une donnée, page 155

S Canal de sortie, page 155

=> Garde, page 153

Opérateur de test de l'activité d'un canal, page 153

? Opérateur de lecture, page 153

! Opérateur d'écriture, page 153

$\tilde{E}$ Energie, estimation de consommation du pixel synchrone, page 148

$\tilde{P}$ Puissance moyenne, estimation de consommation du pixel synchrone, page 148

Graphes

$\overline{\Gamma^{\blacktriangle}_G}(v)$ (MRA) Ancêtres d'un nœud v et lui-même, page 40

$\Gamma^{\blacktriangle}_G(v)$ (MRA) Ancêtres d'un nœud v dans la grille G, page 40

$\overline{\Gamma_G}(v)$ (MRA) Ascendants d'un nœud v et lui-même, page 40

$\Gamma_G(v)$ (MRA) Ascendants d'un nœud v dans la grille G, page 40

$\diamond$ (MRA) Opérateur binaire, page 41

$\lhd$ (MRA) Opérateur d'inondation, page 44

$\rightsquigarrow$ Symbole indiquant que deux sommets d'un graphe sont joignables via un chemin, page 179

Ω Fonction de pondération d'un graphe, page 44

$\oplus$ (MRA) s-opérateur, page 42

$\bot_\oplus$ (MRA) Plus petit élément d'un ensemble au sens de $\preccurlyeq_\oplus$, page 42

π Chemin dans un graphe, page 179

$\Pi_f^{\downarrow}$ Ensemble des chemins strictement descendants le long d'un relief topographique f, page 182

$\Pi_f^{\uparrow}$ Ensemble des chemins strictement ascendants le long d'un relief topographique f, page 182

$\preccurlyeq_\oplus$ (MRA) Relation d'ordre partiel pour $\oplus$, page 42

$\circledcirc$ (MRA) Opérateur d'unification, page 44

A Ensemble contenant les arcs d'un graphe, page 177

a Arc d'un graphe, page 177

$\bar{a}$ Arc opposé de a, page 177

$A^i_{péri}$ Arcs entrants et sortants de la périphérie du $i^{\text{ème}}$ minimum, page 56

d^-_G Degré de connexité (s'adresse suivant le contexte, à une grille ou à un nœud), page 179

$e_\diamond$ (MRA) Elément neutre de l'opérateur $\diamond$, page 42

f Variable parallèle contenant le couple (h,l), page 44

$\mathcal{G}$ Graphe équivalent à G où tous les plateaux minima sont substitués par un nœud, page 57

G Structure de graphe, page 177

G^* Sous-graphe, page 178

$G^*_{\mathcal{M}}$ Sous-graphe composé de tous les plateaux minima d'un graphe G, page 47

$G^*_{\mathcal{M}}{}^i$ $i^{\text{ème}}$ plateau minimum d'un graphe G, page 47

G' Graphe partiel, page 178

h Fonction retournant l'altitude (luminance) de chaque pixel, page 43

$h^i_{\mathcal{M}}$ Altitude du $i^{\text{ème}}$ plateau minimum, page 47

l Fonction retournant l'étiquette de chaque pixel, page 43

$\mathcal{N}(v)$ Ensemble des voisins (***N****eighbors*) du pixel v, page 178

$\overline{\mathcal{N}}(v)$ Ensemble des voisins de v et lui-même, page 178

$\mathcal{N}^=(v)$ Ensemble des voisins du pixel v tel que que leur valeur soit égale, page 36

$\mathcal{N}^I(v)$ Voisin d'**I**nondation : s'il existe, voisin du pixel v de valeur strictement inférieure et la plus petite, page 36

$\mathcal{N}^{\geqslant}(v)$ Ensemble des voisins du pixel v tel que que leur valeur soit supérieure ou égale, page 36

net_G (MRA) Variable parallèle utilisée pour coder la connexité de chaque nœud d'un graphe, page 40

P^*_i $i^{\text{ème}}$ plateau d'un graphe, page 52

$\mathbb{S}$ (MRA) Ensemble de définition sur lequel s'applique les opérateurs associatifs, page 40

t^x_{prop} Temps de propagation de la donnée x sur un arc (pondération d'une image digitale), page 44

V Ensemble contenant les sommets (*vertex*) d'un graphe, page 177

V^{MP} Ensemble de nœuds à l'état MP, page 53

V^{NM} Ensemble de nœuds à l'état NM, page 53

Morphologie Mathématique

Δ Gradient morphologique symétrique, page 188

Δ^+ Gradient morphologique asymétrique supérieur, page 188

Δ^- Gradient morphologique asymétrique inférieur, page 188

δ_B Dilatation d'un ensemble ou fonction par l'élément structurant B, page 186

ϵ_B Erosion d'un ensemble ou fonction par l'élément structurant B, page 186

γ_B Ouverture d'un ensemble ou fonction par l'élément structurant B, page 187

Γ	Ensemble des voisins de plus grand pente, page 194
ψ	Transformation morphologique s'appliquant, suivant le contexte, sur un ensemble ou une fonction, page 184
φ_B	Fermeture d'un ensemble ou fonction par l'élément structurant B, page 187
$\wedge, \vee$	inf et sup, page 184
B	Elément structurant, ensemble de points définits par rapport à son centre, page 186
CB	*Catchment Bassin* : bassin d'attraction associé à un minimum, page 196
cost(.,.)	Coût de passage d'un pixel à son voisin, page 193
D^g_X	Distance géodésique par rapport à un ensemble X., page 192
D_h	Distance topographique sur un relief défini par la fonction h., page 193
I	Image : représente l'espace où s'applique différents opérateurs morphologiques., page 10
INIT	Etat initial : premier état de la machine à état pour l'algorithme de *Hill-Climbing*, page 37
L	Treillis, structure fondamentale en morphologie mathématique, page 184
LPE	Ligne de Partage des Eaux (*Watershed*). Ligne de contour délimitant les régions d'une image, page 22
$LS(.)$	Valeur de la pente de descente maximale (*Lower Slope*), page 192
ML	Minimum local, point ayant tous ses voisins d'altitude strictement supérieure, page 35
MP	Etat Minimum ou Plateau : deuxième état de la machine à état, page 37
NM	Etat Non Minimum : troisième état de la machine à état, page 37
PE	Pixel extérieur, point ayant au moins un voisin d'altitude strictement inférieure et un autre de même altitude, page 35
PI	Pixel intérieur, point ayant tous ses voisins d'altitudes supérieures où au moins un de ses voisins est de même altitude, page 35
R	Région : composante connexe, fraction d'une image., page 10
SKIZ	*SKeleton by Influence Zone* : squelette par zone d'influence, page 196
ZPE	Zone de Partage des Eaux, page 23

Résumé

Cette thèse fait partie d'un projet exploratoire souhaitant répondre à la question suivante : «Est-il possible d'intégrer une chaîne de codage vidéo orienté-objet dans un terminal multimedia portable ?» Afin d'apporter un élément de réponse à ce large problème, cette thèse est une étude d'adéquation algorithme-architecture de la brique de segmentation nécessaire au système complet. Nous proposons une version totalement désynchronisée de l'algorithme de segmentation *Hill-Climbing*, et son implantation microélectronique asynchrone.

L'état de l'art sur les algorithmes de segmentation une fois établi, nous présentons une nouvelle version réordonnancée de l'algorithme de *Hill-Climbing* dans lequel chaque pixel est rendu autonome. Nous démontrons que son comportement aboutit à une segmentation correcte de l'image. La validation et l'adéquation de cet algorithme pour un spectre d'architectures allant du grain le plus fin (un processeur asynchrone par pixel) jusqu'au plus gros sont démontrées grâce à la bibliothèque de prototypage SystemC^{TM}. Enfin, la conception de bas niveau en langage CHP et VHDL montre la faisabilité d'une telle architecture.

Mots clés

MPEG4, visiophonie portable, segmentation, *Hill-Climbing*, architecture parallèle, processeur asynchrone, adéquation algorithme-architecture.

Abstract

This thesis aims to answer to the following question : «Is it possible to implement an object oriented video coder into a multimedia handset ?» In order to provide a piece of answer to this large problem, this thesis aims to study the algorithm architecture adequacy of the segmentation operator needed by the whole system. We propose a totaly unsynchronized version of the segmentation algorithm *Hill-Climbing*, and its asynchronous microelectronic implementation.

After a state of the art about segmentation algorithms, we propose a new asynchronous version of *Hill-Climbing* algorithm where each pixel is autonomous. We demonstrate that its comportment get to a correct picture segmentation. With the SystemC^{TM} hardware modeling environment, we simulate and validate architectures from the finest granularity (one asynchronous processor per pixel) to the coarsest, and we demonstrate that the algorithm-architecture couple is appropriate. Finally, a low level conception in CHP and VHDL language shows the feasibility of such a circuit.

Key words

MPEG4, mobile video visiophony, segmentation, Hill-Climbing, parallel system, asynchronous processor, algorithm architecture adequacy.

ISBN : 2-84813-002-4
ISBNE : 2-84813-003-2