Conception d'une famille de coprocesseurs parallèles intégrées pour le traitement d'images

Thierry Court

THESE

présentée par

THIERRY COURT

pour obtenir le titre de

DOCTEUR de l'INSTITUT NATIONAL POLYTECHNIQUE

de GRENOBLE

(Arrêté ministériel du 23 novembre 1988)

Spécialité : Microélectronique

CONCEPTION D'UNE FAMILLE DE COPROCESSEURS PARALLELES INTEGRES POUR LE TRAITEMENT D'IMAGES

Soutenue le 9 Décembre 1991

Composition du jury :

Président : A. CHEHIKIAN
Rapporteurs : M. JOURLIN
J. GALLICE
Examinateurs : B. COURTOIS
B. CHABERT
R. PESTY

Thèse préparée au sein du Service d'Electronique de Techniques Informatiques et d'Architectures de la Division LETI du Commissariat à l'Energie Atomique
(Centre d'Etudes Nucléaires de Grenoble)

Remerciements

Les idées rassemblées dans ce mémoire ne sont pas le fruit d'une personne isolée. Elles émanent d'un environnement à la fois scientifique et humain dont la richesse résulte de la diversité des personnes qui le constituent, et dont sa force est intimement liée à la mise en commun de compétences échangées à l'occasion de longues discussions, qui ont parfois été houleuses.

Ces trois années de thèse passées au LETI m'ont permis de satisfaire ma curiosité insatiable dans des domaines qui m'étaient jusqu'alors étrangers. Je garderai longtemps en mémoire cette ambiance humaniste, pleine de complicité, régnant dans l'équipe avec laquelle j'ai réalisé ces études. Au terme de cette aventure, je tiens à remercier tous ceux qui ont contribué à son accomplissement.

Je remercie Mr Darier, chef du service SETIA au LETI ainsi que son prédécesseur, Mr Monge pour m'avoir accueilli dans leur service.

Je tiens aussi à remercier Mr Courtois, Directeur du Laboratoire TIM3, pour avoir accepté la direction de cette thèse.

Je remercie Mr A. Chehikian pour m'avoir fait l'honneur de présider mon jury de thèse.

Je remercie Mr M. Jourlin et Mr J. Gallice pour m'avoir fait l'honneur d'être rapporteurs.

Je remercie également Mr B. Chabert, Mr B. Courtois et Mr R. Pesty pour leur participation au jury.

Je tiens à exprimer toute ma gratitude à Mr Chabert, pour m'avoir fait confiance et m'avoir soutenu tout le temps nécessaire à l'accomplissement de ce travail.

J'exprime ici ma plus profonde estime à l'égard de Dominique Massé et Dominique David pour leurs précieux conseils, leurs pragmatismes et leurs commentaires constructifs à propos des diverses solutions, parfois inattendues ou même incongrues, que je leur ai présentées.

Je tiens à remercier Jean-Luc Jacquot, Didier Varreau, Bernard Thevenin, pour leur contribution à ce projet, leur opiniatreté à le défendre et leur rigueur au niveau des réalisations.

Je remercie tout particulièrement Alain Pirson, avec lequel j'ai partagé un bureau pendant trois ans, qui a supporté mes humeurs et la fumée de mes cigarettes durant mes périodes de stress. Nous avons souvent refait le monde ensemble et je lui suis redevable de m'avoir initié aux arcanes de la programmation parallèle sur transputer.

Je salue toutes les personnes, les amis rencontrés dans le cadre de ce travail, je pense à Denise Brunet, Olivier Genvo, Rosolino Lionti et combien d'autres.....

Je remercie amoureusement Christine, ma compagne, qui m'a supporté et aidé dans la rédaction laborieuse de ce mémoire.

Résumé

La conception de systèmes de traitement d'images parallèles mariant dans une même architecture, des microprocesseurs évolués et des opérateurs spécialisés est une tâche délicate, du fait de la diversité des problèmes à prendre en compte. La présente étude identifie une certaine manière de réaliser et d'interfacer des opérateurs spécialisés à une unité centrale de type microprocesseur. Les deux orientations qui ont guidé ce travail sont la recherche d'opérateurs spécialisés polyvalents et reconfigurables et leurs connexions à un bus système, et non à des bus vidéo spécialisés. Ce travail de recherche propose une certaine architecture de circuits dédiés au traitement d'images et deux propositions de réalisation de ces derniers sous la forme de circuits ASIC. Un de ces circuits a pu être réalisé dans le cadre de cette étude en utilisant des outils de type compilateurs de silicium.

Ce travail s'intègre dans un projet plus vaste, dont le but est de développer un système pour le traitement d'image industriel, très performant, modulaire, basé sur la parallélisation dans des structures de type MIMD, d'une unité de traitement d'image élémentaire autonome composée d'un microprocesseur doté d'un coprocesseur parallèle adapté au traitement d'images.

Mots-clés

Traitement d'images, architecture parallèle, processeur spécialisé, parallélisation d'algorithmes, architecture pipe-line et systolique, ASIC.

Summary

The design of parallel image processing systems joining in a same architecture, sophisticated microprocessors and specialised operators is a difficult task, because of the various problems to be taken into account. The current study identifies a certain way of realizing and interfacing such dedicated operators to a central unit with microprocessor type. The two guide lines of this work are the search for polyvalent specialized and reconfigurated operators as well as their connections to a system bus, and not to specialized video buses. This research work proposes a certain architecture of circuits dedicated to image processing and two realization proposals of them. One of them was be realized in this study by using silicon compiler tools.

This work belongs to a more important project, whose aim is the development of an industrial image processing system, high performing, modular, based on the parallelization, in MIMD structures, of an elementary, autonomous image processing unit integrating a microprocessor equiped with a parallel coprocessor suited to image processing.

Keywords

Image processing, parallel architecture, dedicated processor, algorithm parallelization, pipe-line and systolic architecture, ASIC.

Sommaire

Pages

Pages

Pages

Introduction

Dans de nombreuses applications industrielles d'inspection, de reconnaissance de formes ou de localisation et positionnement d'objets, l'utilisation de méthodes numériques pour l'analyse d'images exige des systèmes informatiques très performants. Ces exigences résultent de la grande quantité d'informations devant être traitées (l'image numérisée), de la sophistication toujours croissante des méthodes de traitement utilisées, et de la rapidité de réponse imposée par les impératifs de "temps réel" associés aux cadences des processus industriels. La conception de systèmes d'analyse d'images répondant à ces besoins pour un coût acceptable, nécessite une excellente adéquation entre leurs architectures internes et les tâches de calculs qui leurs sont confiées. L'état des recherches dans le domaine et les machines actuellement commercialisées démontrent que ces systèmes, pour atteindre un niveau de performance suffisant, doivent obligatoirement utiliser des mécanismes de calcul parallèle mis en oeuvre par des architectures spécifiques utilisant des processeurs spécialisés intégrés.

La parallélisation des tâches de traitement de "bas niveau" (algorithmes transformant une image en une image résultat) a été abondamment étudiée, de nombreux systèmes parallèles basés sur de multiples structures (SIMD, MIMD, pipeline et systolique) ont été proposés ou sont même déjà commercialisés avec des fortunes diverses. La situation est beaucoup moins avancée en ce qui concerne la parallélisation des tâches de "moyen niveau", celles-ci étant plus nombreuses et souvent plus complexes que pour l'étape précédente (elles extraient des informations de l'image et les structurent, les résultats ne sont plus des images). A cours terme la sophistication des méthodes de traitement va rendre aussi indispensable que pour le "bas niveau", la parallélisation intensive de cette seconde classe de problèmes.

Cette diversification des tâches à paralléliser et, les inconnues qui subsistent encore au niveau des méthodes à adopter, conduisent à réviser et à remanier les structures existantes optimisées pour le bas niveau, qui utilisent souvent des concepts de parallélisation trop stricts voire rédhibitoires pour résoudre efficacement cette nouvelle gamme de problèmes. Ce travail de recherche s'intègre dans l'une des voies architecturales actuellement à l'étude : des systèmes à parallélismes multiples, intégrant dans une même structure différents concepts de parallélisation mis en oeuvre dans la plupart des cas dans des structures matérielles hétérogènes.

L'architecture explorée dans le cadre de ces travaux est globalement de type MIMD, basée sur l'association d'unités de traitement autonomes identiques traitant en parallèle différentes zones de l'image. Chaque unité est un système de traitement distribué associant un microprocesseur 32 bits et un accélérateur de calcul parallèle intégré qui augmente d'une manière très importante la vitesse de calcul pour certains traitements spécifiques du traitement d'images bas et moyen niveau. Ce type d'association a très peu été étudié dans le cadre de traitement d'images, mais il apparaît souvent au niveau de l'architecture des dernières stations de synthèse graphique haut de gamme commercialisées.

Deux des points les plus délicats de cette approche architecturale, ont fait l'objet de pré-études de faisabilité dans le cadre de deux thèses menées simultanément.

- les travaux de recherches de Mr Alain PIRSON "conception et simulation d'architectures parallèles et distribuées pour le traitement d'images" portent sur l'architecture macroscopique du système, les dispositifs de communications rapides entre unités, l'allocation des tâches et la gestion logicielle de la structure MIMD

- le présent travail de recherche porte, sur la parallélisation structurelle des algorithmes, l'étude architecturale de cet accélérateur de calcul parallèle (Coprocesseur de traitement d'image) et sa réalisation matérielle en utilisant des techniques de conception de type Compilateur de Silicium.

L'architecture adoptée pour ce "coprocesseur de traitement d'images" est une structure synchrone, utilisant des mécanismes de calcul parallèle "pipeline" et "systolique", à l'intérieur de laquelle se propagent les données et des informations de contrôle gérant le séquencement des calculs. Sa réalisation matérielle de petite taille est basée sur l'assemblage d'un petit nombre de processeurs intégrés, chacun de ces circuits étant une petite structure parallèle spécifique, capable de réaliser en fonction de sa programmation différents algorithmes de traitement d'images. Cette approche architecturale est validée sur deux études d'intégration de circuits pour cet accélérateur qui sont détaillées dans les deux derniers chapitres de ce mémoire de thèse.

Première partie

Etat de l'art et orientation de l'étude

Sommaire de la première partie

1.1) Architectures des systèmes de traitement d'images

1.1.1) Généralités

Les architectures de machines utilisées en traitement d'images dépendent de la nature des images étudiées et du type de traitements qui leurs sont appliqués. L'image digitale est obtenue par échantillonnage et numérisation d'une image analogique, elle est issue dans la majorité des cas de capteur tel que : tube vidicon, barette ou matrice CCD. Suivant la nature du capteur et le pas d'échantillonnage, l'image discrétisée est représentée par un (image monochrome) ou plusieurs (image multispectrale "couleur") tableaux d'entiers (256x256 à 2048x2048), chaque élément (pixel) de ces tableaux indiquant pour une bande spectrale déterminée, la mesure de l'intensité lumineuse (généralement codée sur 8 bits) d'un échantillon de l'image analogique.

Les problèmes rencontrés en traitement d'images sont très nombreux et extrêmement variés, on peut néanmoins les regrouper en quatre grandes catégories :

- l'archivage d'images et leur gestion dans des bases de données;
- la restauration d'images;
- la compression, transmission et décompression d'images;
- l'analyse et l'interprétation d'images.

Les frontières entre ces quatre types de traitement appliqués aux images sont assez floues au niveau des méthodes algorithmiques employées, mais nettement plus marquées pour des raisons économiques (performances / coût) au niveau de l'architecture des systèmes informatiques dédiés à leur mise en oeuvre. Dans le cadre de cette étude, nous nous intéresserons plus précisement aux systèmes d'analyse d'images utilisés dans le domaine de la production industrielle pour :

- le contrôle non destructif (inspection de défauts, contrôle d'aspect ou de conformité);
- l'identification et la localisation d'objets (en vue d'une manutention robotisée par exemple).

1.1.2) Les tâches à réaliser

Suivant la nature des données qu'ils manipulent et la complexité de leurs structures, les algorithmes utilisés en analyse d'images peuvent être rangés par commodité, de manière hiérarchique (fig.1.1) en trois classes de problèmes : les tâches de bas, moyen et haut niveau.

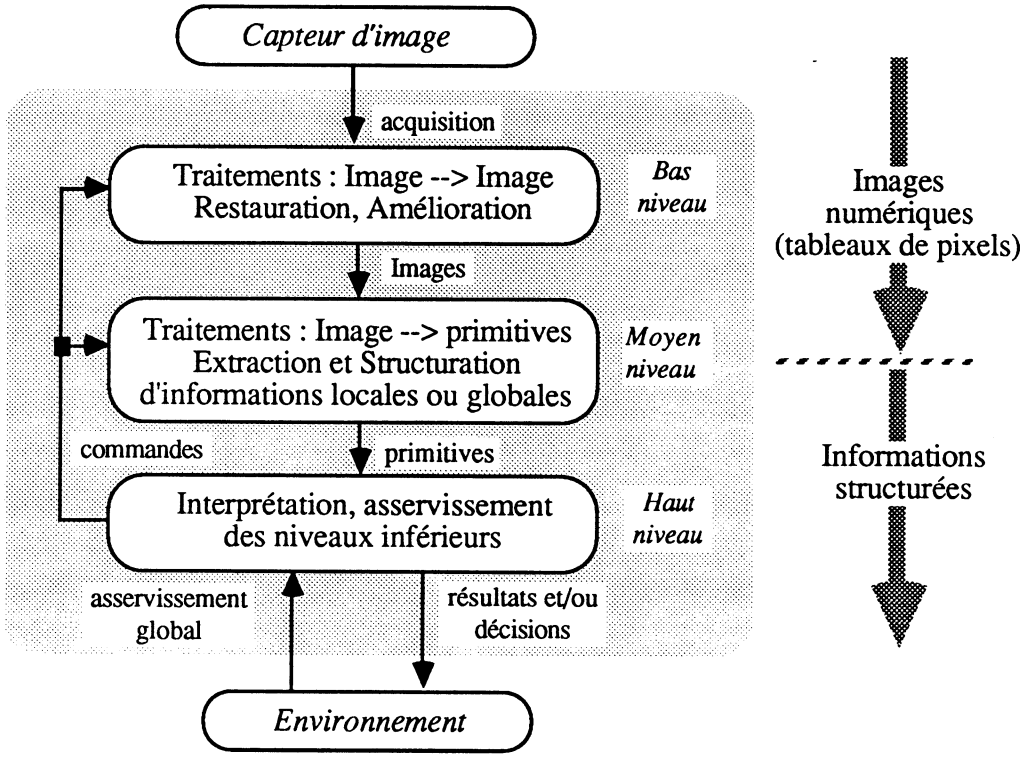


Figure 1.1: Interactions entre les tâches de bas, moyen et haut niveau

Le bas niveau peut être caractérisé de façon simple par quelques mots : "image en entrée, image en sortie", il englobe les algorithmes de traitement qui opèrent au niveau d'images discrétisées (tableau de pixels); on trouve par exemple les algorithmes de restauration d'image (filtrage, correction de déformations géométriques), d'extraction de contours, squelettes, textures, de morphologie mathématique (érosion, dilatation), d'étiquetages d'objets ou de régions (analyse des composantes connexes). Ces algorithmes ont des structures assez simples qui présentent peu ou pas de branchements, ils sont très répétitifs : on applique une même procédure de calcul pour chacun des 10^5 à 10^6 pixels que comporte l'image.

Le moyen niveau regroupe les algorithmes qui extraient, à partir des images issues du bas niveau, des informations "primitives" qui caractérisent certains des éléments qu'elles contiennent. Ce niveau correspond à une phase où l'on réduit le volume d'informations et on en change la nature : les résultats ne sont plus des tableaux de pixels mais des structures de données évoluées (vecteurs, listes, graphes, arbres). On trouve par exemple des algorithmes de calcul de mesures statistiques (histogramme, moments) ou géométriques (aire, périmètre, .. d'objets), de détection de droites ou de courbes (transformée de Hough, projection), d'extraction et chainage des points contours, de codage de régions. Ces algorithmes ont des structures assez régulières parfois aussi simple que pour le bas niveau mais ils ne prennent pas nécessairement en compte tous les pixels de l'image.

Le haut niveau concerne l'interprétation des primitives qui ont été élaborées par les niveaux inférieurs, dans certains cas leur comparaison avec des modèles issus de base de connaissances, et la prise de décision. Ces algorithmes sont sophistiquées, dépendent fortement de la nature de l'application à résoudre, leurs structures sont complexes et nécessitent qu'ils soient formulés dans des langages informatiques de haut niveau.

Ces trois niveaux sont étroitement liés par des communications bidirectionnelles, les processus de bas et moyen niveau élaborant des mesures sur l'image pour les processus de haut niveau et éventuellement sous leur contrôle. Compte tenu du volume de données et de la quantité de calculs que représentent les tâches de traitement de bas et moyen niveaux, la tendance actuelle est d'utiliser des structures de calcul parallèles spécifiques pour leur mise en oeuvre et, de réaliser les tâches de haut niveau au moyen de calculateurs universels.

1.1.3) Architecture des systèmes d'analyse d'images

Mis à part les systèmes parallèles de type multiprocesseurs, la majorité des systèmes d'analyse d'images, qui seront détaillés dans la suite de ce mémoire, peuvent être macroscopiquement décomposés (fig. 1.2) en deux unités fonctionnelles interconnectées :

1) une structure matérielle spécifique adaptée aux problèmes de l'analyse d'images qui réalise la numérisation de l'image, sa mémorisation, certaines tâches de bas et moyen niveaux et comporte :

   * une chaine d'acquisition constituée d'une ou plusieurs caméras et d'un système de numérisation de l'image;
   * un ensemble de mémoires de grandes tailles destinées à stocker la ou les images numérisées et certaines images temporaires utilisées au cours des calculs;
   * un ou plusieurs dispositifs de calculs spécialisés utilisant des techniques de calcul parallèle pour accélérer l'exécution de certaines tâches de bas et moyen niveaux;
   * un dispositif de communication qui permet des échanges d'informations entre le système spécifique et le calculateur hôte;
   * pour les systèmes les plus évolués, cette structure spécifique dispose en plus d'un microprocesseur qui lui assure une certaine autonomie de fonctionnement (intelligence locale) et peut aussi réaliser certaines des tâches de traitement.

2) un calculateur hôte conventionnel réalisant les tâches de haut niveau et l'interface avec l'environnement (utilisateur, actionneurs, autre système informatique, ...).

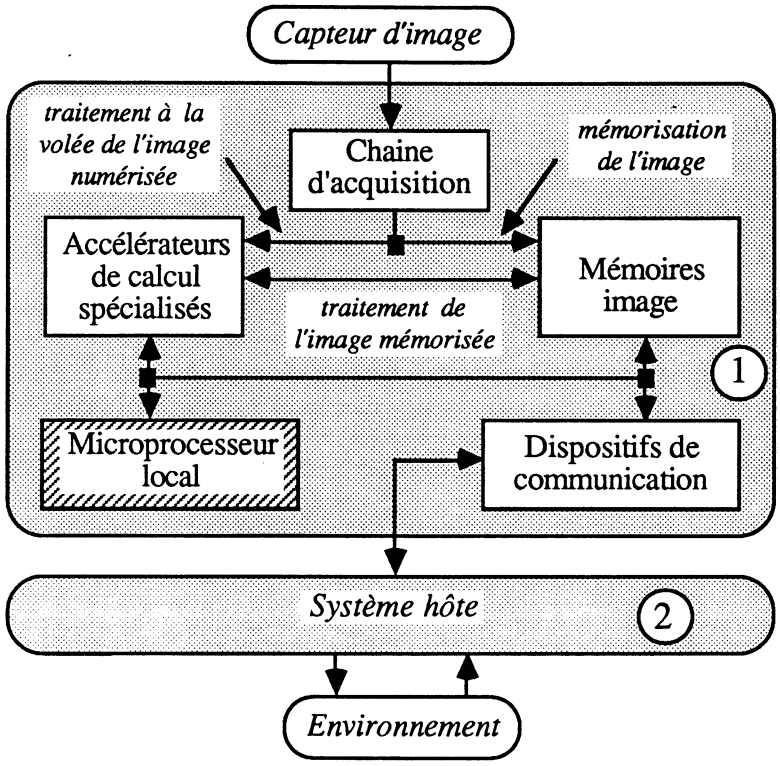


Figure 1.2 : Structure générale des systèmes d'analyse d'images

La majorité des différences entre les divers systèmes d'analyse d'images portent sur l'architecture des accélérateurs de traitements spécialisés, leurs performances et leurs mécanismes d'accés aux mémoires contenant l'image.

1.2) Parallélisme et traitement d'images : état de l'art

Les structures de calcul parallèle pour le traitement d'images sont très variées, et l'étude exhaustive du domaine dépasserait largement le cadre de cette présentation; nous citerons dans cet état de l'art, uniquement des systèmes dédiés au traitement d'images ayant fait l'objet d'une réalisation matérielle, leur classification sera réalisée en fonction des concepts de parallélisation et des mécanismes de contrôle qu'ils utilisent.

1.2.1) Définitions

Les différents termes techniques, se rapportant à l'architecture des systèmes informatiques parallèles, qui seront utilisés dans la suite de ce mémoire, sont définis ci aprés, ces définitions sont en majorité issues de l'ouvrage "Parallel Processing" (JOH86).

Architectures parallèles :
Elles mettent en oeuvre dans une même structure, un ensemble d'unités de traitement distinctes qui exécutent concurremment des tâches de calcul dépendantes les unes des autres (parallélisme explicite) et, qui échangent des informations au cours du traitement.

Architectures parallèles à forte granularité :
Elles comportent un nombre réduit (4 à 16) d'unités de traitement autonomes,trés puissantes et complexes (plusieurs circuits VLSI pour réaliser une unité).

Architectures parallèles à granularité moyenne:
Elles comportent un nombre variable d'unités de traitement (16 à 128), dont les performances et la complexité (un seul circuit VLSI) sont de l'ordre des microprocesseurs universels.

Architectures parallèles à faible granularité :
Elles comportent un nombre important (> 128) d'unités de traitement assez simples (plusieurs unités par circuit VLSI), ces unités sont très souvent toutes identiques.

Architecture parallèle homogène :
Toutes les unités de traitement qui la compose sont identiques et disposent de ressources identiques (chemins de communication et mémoire locale).

Architecture à traitement distribué :
Le traitement est découpé en différentes sous-fonctions qui sont assurées par des processeurs spécialisés (processeur scalaire, coprocesseur arithmétique, accélérateur vectoriel, ...).

Coprocesseur :
Dispositif de traitement spécialisé associé à un processeur standard autonome (ex: microprocesseur), qui augmente d'une manière très importante la puissance de calcul pour certaines fonctions bien spécifiques (arithmétique sur les réels, fonction de tri, ...).

Structure de donnée statique :
Organisation de variables (tableau, vecteur, ...) dont la taille est fixe, connue à priori, indépendante de résultats issus au cours des calculs.

Structure de donnée dynamique :
Organisation de variables (vecteur, liste, graphe, ...) dont la taille dépend de résultats issus au cours des calculs (ex. : dépend du nombre de points de contours ou de régions de l'image).

1.2.2) Les problèmes liés au parallélisme

Les performances d'une architecture parallèle dépendent étroitement de la manière, dont on peut alimenter les différentes unités de traitement qui la composent en travaux exécutables simultanément. La méthode la plus souple et la plus générale pour réaliser cette distribution du travail est issue de modèles formels tels que les graphes de flots de données et les réseaux de Petri, elle consiste à partitionner l'algorithme de traitement global en sous-tâches (processus) (fig. 1.3) plus ou moins élémentaires qui :

- peuvent s'exécuter concurremment de manière asynchrone;
- échangent des messages et/ou se partagent des données communes au cours de leur exécution;
- se synchronisent à des évènements déterminés.

Dans le cas idéal, il suffit d'allouer au niveau matériel, une unité de traitement et des médias de communication pour réaliser chacun de ces processus et échanges d'informations.

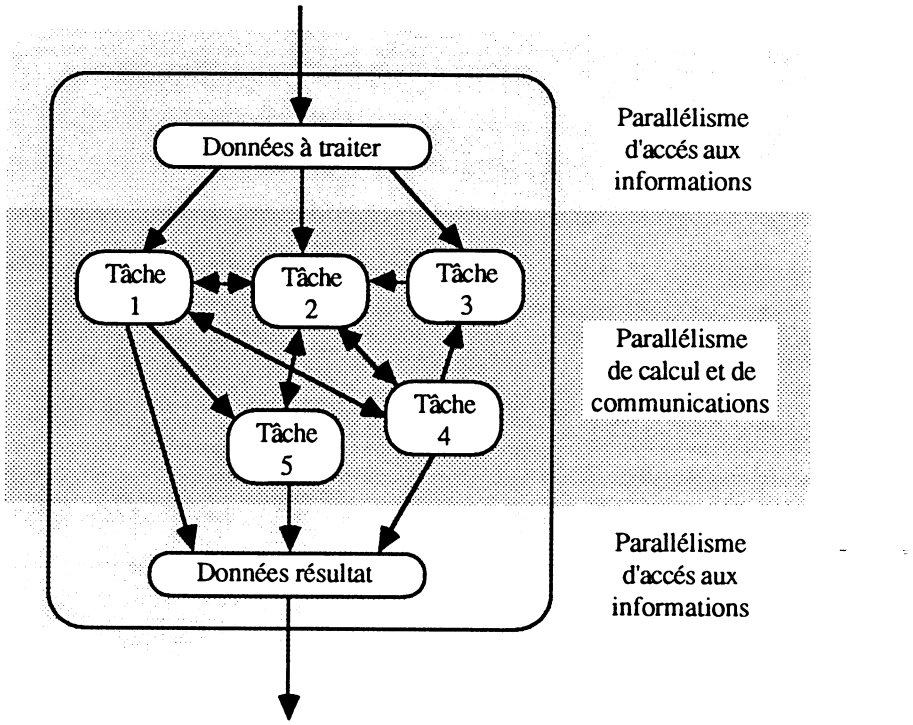


Figure 1.3 : Décomposition générale d'une tâche en processus parallèles

Dans la pratique, pour être efficace, cette division de travail (parallélisation), qui peut être réalisée de multiples manières, doit être adaptée à l'architecture du système cible et aux choix qui ont été faits lors de sa conception pour répondre au problème fondamental de toute architecture

parallèle: le rapport *calculs / communications*. Ces choix, sur les ressources allouées pour réaliser les calculs et les communications, peuvent se situer au niveau matériel ou temporel (temps alloué à chacune de ces deux tâches). Cette parallélisation peut être principalement orientée de deux manières :

- Parallélisme et systèmes à forte et moyenne granularité :

  Pour les systèmes qui comportent des unités de traitement puissantes dotées de ressources locales (mémoire de travail) et disposent de média de communication aux performances limitées, pour obtenir des vitesses de calcul maximales, il faut privilégier une décomposition du problème en de relativement gros processus qui maximisent le rapport :

  $$(\textit{calculs locaux} \,/\, (\textit{échanges} + \textit{accès aux données communes})) \quad >> \quad 1$$

- Parallélisme et systèmes à faible granularité :

  Pour les systèmes qui comportent un nombre important d'unités de traitement aux ressources et performances limitées et disposent de riches possibilités de communication (dialogues entre unités, accés aux données), il faut adopter une décomposition trés fine, en petits processus nécessitant de nombreux échanges d'informations et/ou accès aux données communes, en essayant d'équilibrer judicieusement le rapport :

  $$(\textit{calculs locaux} \,/\, (\textit{échanges} + \textit{accès aux données communes})) \quad >= \quad 1$$

Au fur et à mesure que diminue la granularité du système et de la parallélisation, cette division du travail et son allocation aux unités de traitement deviennent des problèmes de plus en plus fondamentaux. Ces tâches, qui sont déjà à la base difficiles, sont encore plus complexes à résoudre, si elles doivent prendre en compte certaines limitations particulières des systèmes parallèles cibles pour :

- la nature des tâches réalisables par les unités de traitement (spécialisées ou quelconques);
- l'organisation et les possibilités des dispositifs de communication et d'accès aux données communes;
- la nature des mécanismes de contrôle et de synchronisation (centralisés ou distribués) (autonomie de fonctionnement des unités de traitement).

Dans la pratique, des limitations matérielles interdisent la réalisation à un coût acceptable d'architectures parallèles performantes, qui minimiseraient les restrictions sur la parallélisation et l'allocation des tâches. Les principales d'entre elles sont par ordre d'importance :

1) Le nombre limité de dispositifs de communication entre unités de traitement, qui est directement lié au nombre de connexions électriques du système;

2) L'impossibilité de gérer rapidement des accès multiples et simultanés à une même mémoire contenant des données communes;

3) La complexité de réalisation de puissantes unités de traitement autonomes (contrôle, accès aux données, communications) et leurs tailles, qui ne permettent pas d'envisager actuellement plus d'une unité par circuit VLSI.

Cette situation conduit en général, lors de la conception de systèmes parallèles spécialisés à très hautes performances disposant de nombreuses unités de traitement, à privilégier l'aspect matériel et à orienter de manière plus ou moins stricte la façon de paralléliser les algorithmes, en fonction des choix techniques qui simplifient la réalisation des structures matérielles de ces systèmes.

De ce compromis (matériel/fonctionnel) dépendent les caractéristiques finales des systèmes, au niveau des performances, du coût et surtout de la diversité des problèmes qu'ils sont capables de résoudre en utilisant au mieux les différentes unités de traitement qui les composent.

1.2.3) Structures parallèles pour le traitement d'images

Au niveau des architectures parallèles dédiées à la réalisation de tâches de traitement d'image de bas et moyen niveaux, on rencontre principalement trois approches architecturales :

- deux approches très spécialisées qui utilisent des mécanismes de calcul parallèle synchrones, une unité de contrôle centralisée et, en général des circuits VLSI spécifiques qui intègrent plusieurs unités de traitement :
  - les architectures à structure de données (structures parallèles SIMD),
  - les architectures à propagation d'informations (structures pipelines et systoliques);

- une approche plus générale, utilisant des mécanismes de calcul parallèle asynchrones, qui est basée sur l'association dans des structures diverses d'unités de traitement autonomes qui sont le plus souvent des microprocesseurs : les architectures MIMD .

1.2.3.1) Les architectures à structure de données

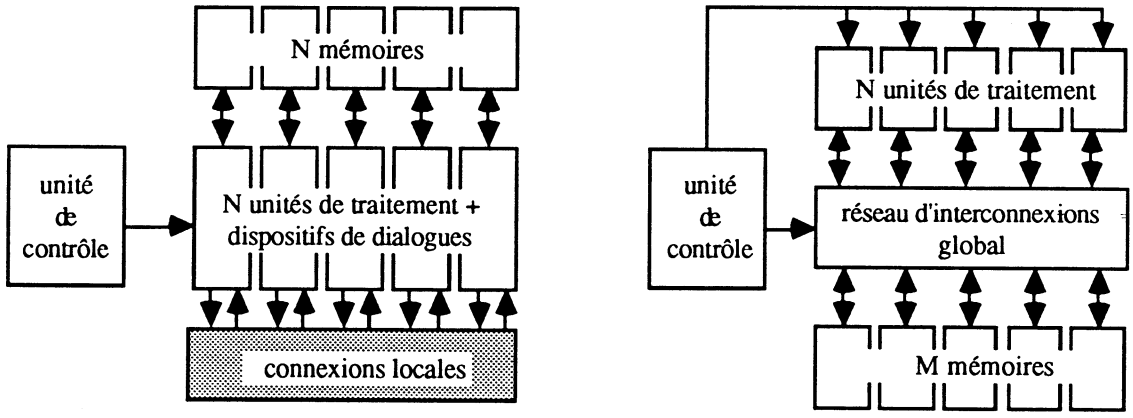
1.2.3.1.1) Principe

La majorité des algorithmes de traitement d'images de bas niveau étant appliqués de manière identique pour chacun des pixels de l'image à traiter, il est possible d'accroître les performances d'un facteur N, en traitant simultanément N pixels au moyen de N unités de traitement fonctionnant en parallèle (parallélisme lié à la multiplicité et l'organisation des données (structures)).

Une unité de contrôle et de commande centralisée émet des instructions aux différentes unités de traitement et, à chaque cycle de calcul chacunes d'entre elles exécutent la même instruction mais sur des données différentes (structure parallèle à commande centralisé SIMD).

Le système de calcul parallèle est organisé autour d'un nombre plus ou moins important d'unités opératives élémentaires identiques (de 4 à 16384 dans les systèmes actuels) qui échangent entre elles des informations et, qui accèdent **en parallèle** à un groupe de mémoires dans lesquelles sont distribuées les informations à traiter (image numérisée).

Deux types d'architectures sont les plus souvent rencontrés (fig. 1.4) : les tableaux de processeurs ("Processor Array") et les structures à réseau d'interconnexions.



"Processor Array"

Structure à réseau d'interconnexions

Figure 1.4 : Architectures parallèles à structure de données

1.2.3.1.2) Les tableaux de processeurs ("Array of Processors")

Ils sont composés d'un ensemble d'unités élémentaires interconnectées suivant des structures régulières (ligne ou matrice de processeurs), chaque unité élémentaire regroupant une unité opérative, une mémoire locale (le plus souvent privée : non partagée) et des dispositifs de communication pour les dialogues avec les unités voisines (dialogues locaux). Un dispositif de dialogue global permet de distribuer l'image ou des zones d'image dans les différentes mémoires locales et de restituer les résultats issus des traitements. La gamme des "processor array" pour le traitement d'image est très étendue, nous pouvons citer les exemples suivants qui se différencient essentiellement par la topologie de leur dispositifs d'interconnexion et les caractéristiques de leurs unités de traitement (performances, autonomie d'accès aux informations: calcul d'adresses). Nous citerons uniquement des tableaux de processeurs unis et bidimensionnels, mais il est à noter que de nombreuses études en cours s'orientent vers des structures aux connexions pyramidales.

- Tableau de processeurs unidimensionnel (ligne de processeurs) :

  LAP : "Linear Array Processor" (WAL88), de 64 à 4096 unités de traitement 1bit dialoguant avec leur 2 voisines.

  CAP II : "Cellular Array Processor", (MOR85, MOR86)
  16 unités de traitement arithmétiques 16 bits interconnectées par 3 bus.

  SYMPATI : "Système Multi-Processeur Adapté au Traitement d'Image" (CAS85, BAS86)
  16 à 128 unités de traitement 8 bits dialoguant avec leur quatre voisines.

- Tableau de processeurs bidimensionnel (matrice de processeurs) :

  MPP : "Massive Parallel Processor" (DUF81, POT83, ROS83)
  128x128 unités de traitement 1 bit dialoguant avec leurs quatre voisines.

  CLIP : "Cellular Logic Image Processor" (DUF81, DAW88)
  96x96 unités de traitement 1 bit dialoguant avec leurs huit voisines.

  AAP : "Adaptative Array Processor" (DUF81, KID83)
  64x64 unités de traitement 1 bit dialoguant avec leurs huit voisines.

Exemple :

SYMPATI "Système Multi-Processeur Adapté au Traitement d'Image" (CAS85, BAS86) a été développé au Laboratoire CERFIA de l'université Paul Sabatier de Toulouse, il comporte de 16 à 128 unités 16 bits interconnectées suivant une structure unidimensionnelle (Fig. 1.5) "Processeur ligne". L'intégration de ces unités de traitement sous forme de VLSI a été réalisée, au Laboratoire DTA/LETI/DEIN du Commissariat à l'Energie

Atomique (CEA, Saclay), (Précaractérisé CMOS 1,2 µm , 224 broches, 90000 transistors, 4 unités par circuit).

Chaque unité de traitement élémentaire regroupe :

- un multiplieur 8x8 bits, une unité arithmétique et logique 16 bits, 16 registres de travail et une unité de calcul d'adresse (chaque unité génère les adresses d'accès à sa mémoire locale);
- une mémoire statique externe non partagée (16 à 128 Kilo Octets);
- des dispositifs de communication permettant à chaque unité de dialoguer avec les quatre unités les plus proches;
- un registre à décalage global permettant de distribuer les informations dans le réseau de processeurs et de restituer les résultats.

La gestion du séquencement du système et la génération des instructions sont réalisées par une unité microprogrammée, basée sur des composants microprogrammés standards, qui fonctionne sous le contrôle d'un microprocesseur 16 bits (Intel 80286).

Une distribution particulière (hélicoïdale) de l'image dans les différentes mémoires locales, permet aux unités de calcul d'accéder en parallèle à des groupes de pixels voisins sur une même ligne ou une même colonne de l'image.

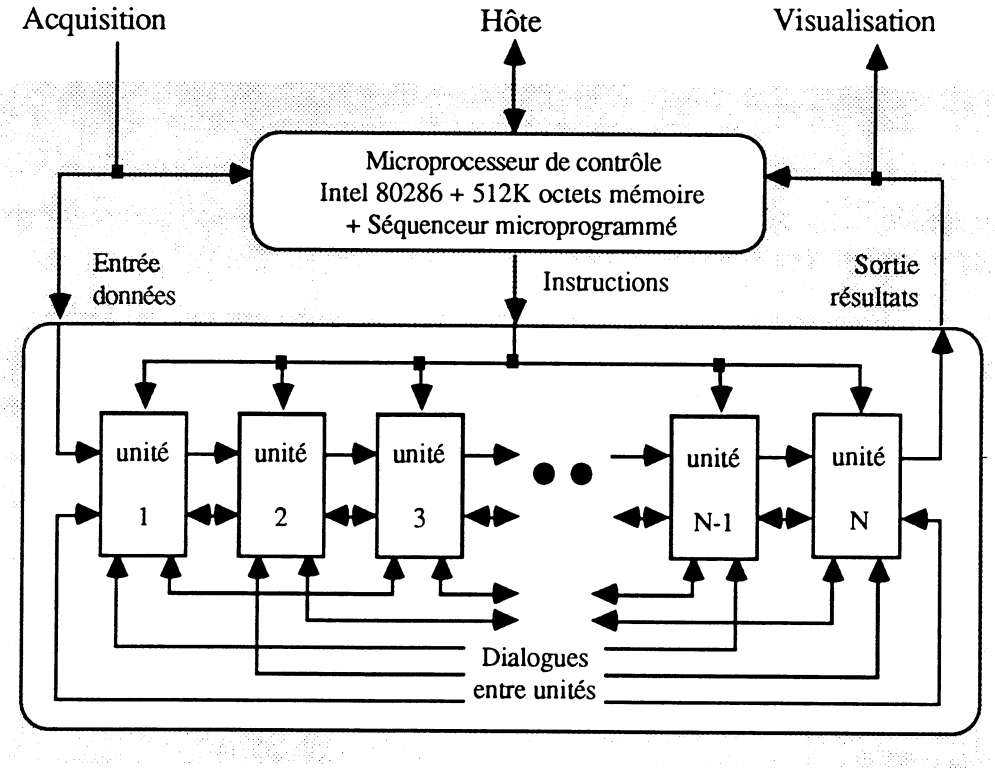


Figure 1.5 : Architecture de SYMPATI

1.2.3.1.3) Les structures à réseaux d'interconnexions

Elles sont composées d'un ensemble d'unités de traitement et de mémoires qui peuvent être mises en relation par un réseau d'interconnexions (exemple: interconnexion matricielle "Cross Bar"), toutes les communications de la structure transitent via les mémoires. La réalisation de grands réseaux d'interconnexions étant difficile (nombre très important de connexions), ces structures comportent en général un nombre d'unités de traitement beaucoup plus faible que les tableaux de processeurs, celles-ci sont par contre souvent plus puissantes. Nous avons par exemples :

PICAP 2 : (processeur de voisinage) (TAN80), 4 unités de traitement 8 bits (unité arithmétique + table de transcodage) interconnectées à 4 mémoires de travail locales.

PIP : "Parallel Image Processor" (PFE87), 8 unités de traitement 8 bits (unité arithmétique + multiplieur) interconnectées à une mémoire centralisée organisée en mots de 64 bits.

MPA : "Massively Parallel Architecture" (GRO87)
1024 unités de traitement 4 bits interconnectées via un réseau à permutation à 10 étages ("Omega network"), les mémoires sont organisées en mots de 4 bits et sont distribuées au niveau de chaque unité.

Exemple :

PIP "Parallel Image Processor" (PFE87) a été développé par Visual Information Technology Inc. (USA), il est avec LAP (WAL88) l'une des rares structures SIMD de traitement d'image qui semble avoir été utilisées pour des applications industrielles d'analyse d'image. Sa structure est composée de trois blocs fonctionnels (Fig. 1.6) :

- un circuit VLSI (170.000 transistors, CMOS 1,5 µm, 179 broches) conçu au moyen de compilateur silicium, qui intègre huit unités de traitement et le réseau d'interconnexions pour l'accès à la mémoire :
  * chaque unité de traitement comporte un multiplieur 8x8 bits, une unité arithmétique et logique 16 bits et 32 registres de travail à quadruple accès;
  * le réseau d'interconnexions est à permutation ("Shuffle Network"), il permet à chaque unité de traitement d'accéder à l'un des 8 octets du mot de 64 bits.

- une mémoire dynamique centralisée pour stocker l'image, organisée en mots de 64 bits (1 mot = 8 pixels voisins sur une même ligne) et, accessible par le système hôte;

- le contrôle du séquencement, la génération des instructions et le calcul des adresses pour l'accés à la mémoire, sont réalisés par une unité microprogrammée basée sur des composants standards.

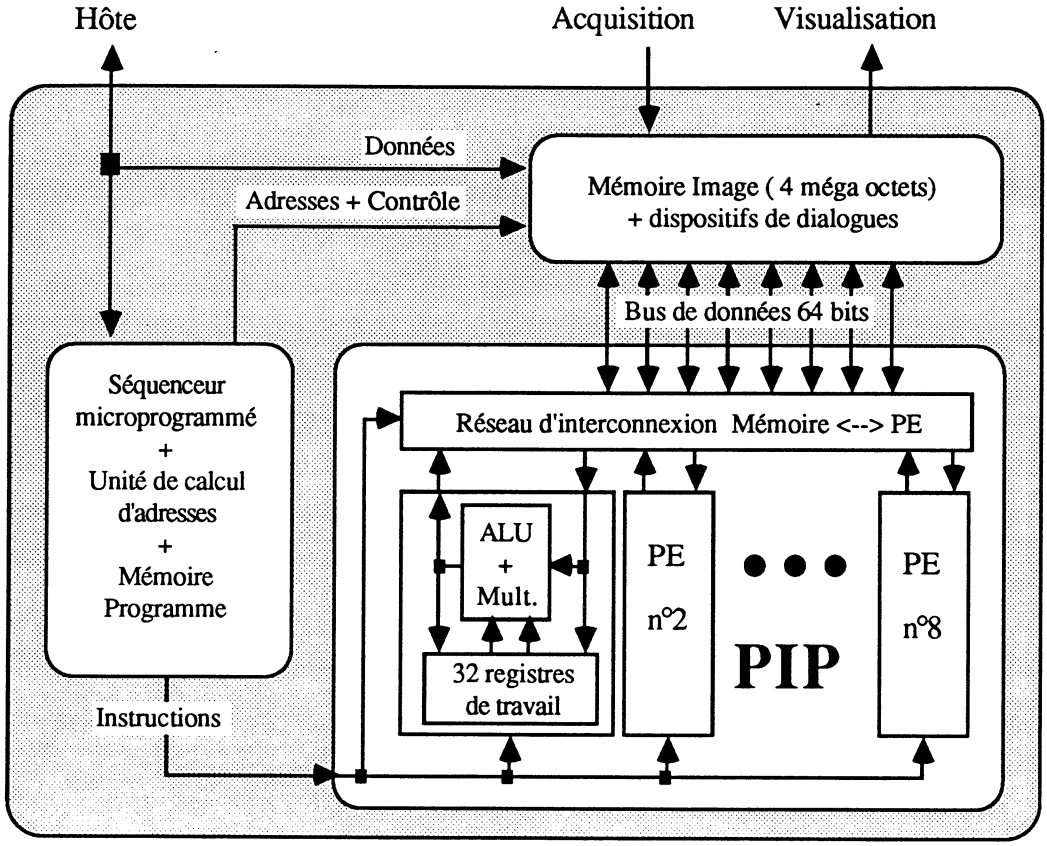


Figure 1.6 : Architecture de PIP

1.2.3.2) Les architectures à propagation d'informations

1.2.3.2.1) Principe

Ces architectures mettent en oeuvre des combinaisons de modèles de calcul parallèle "pipeline" (ZAV84, GOT85, DAW88), "systolique" (KUN79, KUN80, KUN82), qui utilisent pour la parallélisation des problèmes certaines de leurs propriétés structurelles ("parallélisme lié à la formulation algorithmique"). Elles traitent "séquentiellement" les pixels issus de mémoires centralisées, réalisent pour chacuns d'entre eux, en un temps identique, une même procédure de calcul (N cycles d'horloge), les méthodes de parallélisation utilisées ayant pour buts :

- la minimisation du temps d'exécution de cette procédure
  (un seul cycle de l'horloge de calcul dans le meilleur des cas);
- la maximisation de la taille de cette procédure et de fait la minimisation du nombre d'accès à l'image (l'ensemble des tâches de traitement de bas et moyen niveaux dans certains cas très favorables).

La méthode de parallélisation utilisée est classique (division en processus et allocation d'une unité de traitement et de chemins de communication pour la réalisation de chacuns d'entre eux), mais cette partition du travail est réalisée en privilégiant les dialogues entre processus et en minimisant et centralisant les accés aux données de départ et aux résultats. Dans le cas du traitement d'images bas et moyen niveaux, cette division du travail peut être réalisée simultanément à deux niveaux algorithmiques :

- **Niveau macroscopique** (techniques d'enchaînement pipeline (ZAV84))
  La tâche de traitement globale peut souvent se décomposer sous la forme d'une suite d'étapes élémentaires (ex: rotation, filtrage, extraction d'informations, ...) appliquées séquentiellement sur l'image, chaque étape utilisant les résultats issus d'étapes précédentes.
  On réalise en parallèle plusieurs étapes. (recouvrement de tâches, "Overlapping")

- **Niveau microscopique** (systolisation et parallélisation structurelle (KUN82, KUN84))
  Les étapes élémentaires consistent en des algorithmes de calcul de petites tailles, réguliers et décomposables à un niveau atomique (opération diadique élémentaire) sous forme de graphes de calculs statiques dans lesquels circulent rythmiquement les données.
  On réalise en parallèle certaines ou la totalité des opérations élémentaires d'une même étape.

Pour mettre en oeuvre efficacement ce type de formulation parallèle, l'architecture interne de ces systèmes favorise la circulation orientée des informations entre les différentes unités de traitement, pour cette raison ils sont couramment appelés **"architectures à propagation d'informations"**.

La majorité des architectures parallèles dédiées au traitement d'images rélisées à l'heure actuelle, reposent sur de tels principes architecturaux, leurs structures matérielles sont en général assez proches les unes des autres, et peuvent être décomposées en quatre blocs fonctionnels (fig. 1.7):

- une unité de contrôle centralisée qui séquence les actions des différentes unités, via un nombre réduit d'horloges et de signaux de contrôle;
- une ou plusieurs mémoires centralisées de grandes tailles pour stocker les images à traiter et les images temporaires de calcul;
- une unité de calcul d'adresses qui gère l'accès séquentiel aux informations en mémoire suivant des lois qui sont le plus souvent de type balayage vidéo (ligne à ligne);
- un nombre plus ou moins important de dispositifs de traitements spécialisés, interconnectés par des réseaux programmables, qui réalisent des tâches de bas et moyen niveaux ; chacun de ces dispositifs de traitement est le plus souvent une petite structure de calcul parallèle dédiée à la réalisation d'un ou plusieurs algorithmes de traitement (un seul dans la majorité des cas).

Les tâches de traitement sont réalisées en faisant circuler une ou plusieurs fois les images à travers les dispositifs de traitement (PRA 82).

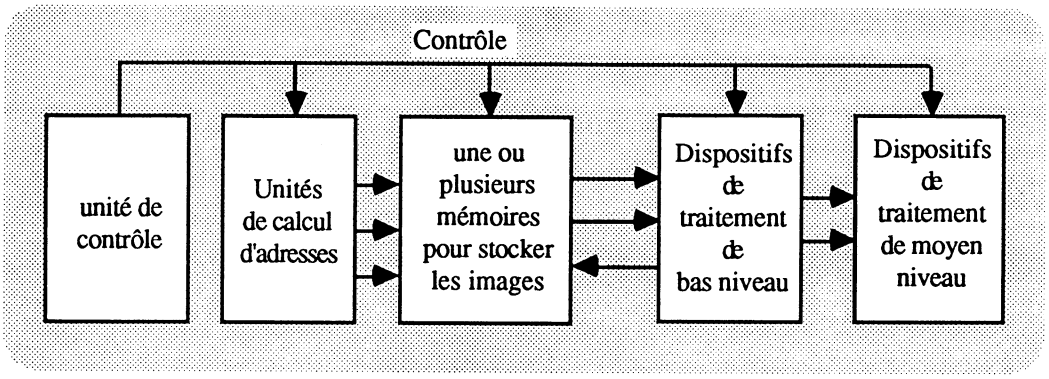


figure 1.7 : structure générale des architectures à propagation d'informations

Nous pouvons citer pour exemples les systèmes ci-dessous, qui utilisent le plus souvent des dispositifs de traitement analogues (processeur de filtrage linéaire, de morphologie mathématique, d'histogramme, ...) et se différencient surtout par les lois qu'ils peuvent utiliser pour accéder aux informations.

- Systèmes traitant les informations uniquement suivant un balayage vidéo (suivant les lignes) CYTOCOMPUTER (LOU85, CUB85), RAPAC (ELP87).

- Systèmes traitant les informations suivant des balayages horizontaux ou verticaux (lignes ou colonnes) : VICOM (PRA82, PRA85), VIPS (GOT85), VISTA (PAU88).

- Systèmes traitant les informations suivant des lois de balayages complexes (rotation, réinterpolation, ...) : TEMAP (MOL88), IP (TAK88, HIT88).

Le développement très important de cette approche architecturale est dû en grande partie, à la disponibilité d'un grand nombre de ces dispositifs de traitement sous forme de circuits intégrés VLSI, dont une grande partie sont conçus et commercialisés par des fabricants de composants (LSI logic, INMOS, PLESSEY Semiconductor, NEC, HITACHI, TOSHIBA, TRW Microelectronic). Nous pouvons citer pour exemples :

- des circuits pour les tâches de bas niveau :
  * convolution (PLE86, ABE87, RUE87, WIL87, TRW87,TUR87, RUE88, TUR88);
  * filtrage médian et morphologie mathématique (RUE87, WIL87, RUE88);

* recherche de motifs binaires "Pattern Matching" (PEL87, RUE88);
* étiquetage des composants connexes (WEI87, TAK88);

- des circuits pour les tâches de moyen niveau :
  * histogramme, transformée de Hough (DOL89);
  * calcul de moments bidimensionnels (HAT86);
  * calcul de mesures (aire, périmètre, barycentre) sur des connexes (WEI87, TAK88);

- des circuits de calcul d'adresses pour générer des lois particulières d'accès aux images (rotation, réinterpolation) (STR85, ELD88, TAK88).

1.2.3.2.2) Exemple

IP "Image Processor" (TAK88, HIT88) est un système industriel de traitement d'images commercialisé par Hitachi depuis la fin de l'année 1989. Sa structure est issue d'un ensemble de travaux sur les processeurs parallèles intégrés (FUK84, FUK85, FUK86, KOB87) réalisés à l'Hitachi Research Laboratory (Japon). Son architecture préfigure selon nous l'évolution à court terme de ce type de systèmes, elle est trés classique au niveau des mécanismes de séquencement employés (traitement d'un pixel à chaque cycle d'horloge: 83 nanosecondes) mais remarquable sur quatre points :

- la gamme très étendue d'algorithmes pris en compte,
- les lois d'accès aux images mémorisées,
- l'utilisation systématique de structures parallèles intégrées multifonctions (capable de réaliser suivant leur programmation différents algorithmes de traitement),
- les techniques utilisées pour sa réalisation matérielle, conception de huit circuits sur mesure: 7 prédiffusés CMOS 2 µm et un circuit Bi-CMOS 1,8 µm "Full Custom" ISP).

Son architecture générale (Fig. 1.8) peut être découpée en trois blocs fonctionnels :

- une intelligence locale basée sur un microprocesseur 16 bits (Motorola MC68000) qui gère le système et les communications avec l'hôte,
- un ensemble de mémoires images 512x512 pixels (3 images organisées 8 bits/pixel, 3 images binaires) accessibles par le microprocesseur local et l'accélérateur de calcul),
- un accélérateur de calcul synchrone réalisé au moyen de ces huit circuits spécifiques (6 dispositifs de traitement, 2 réseaux d'interconnexion).

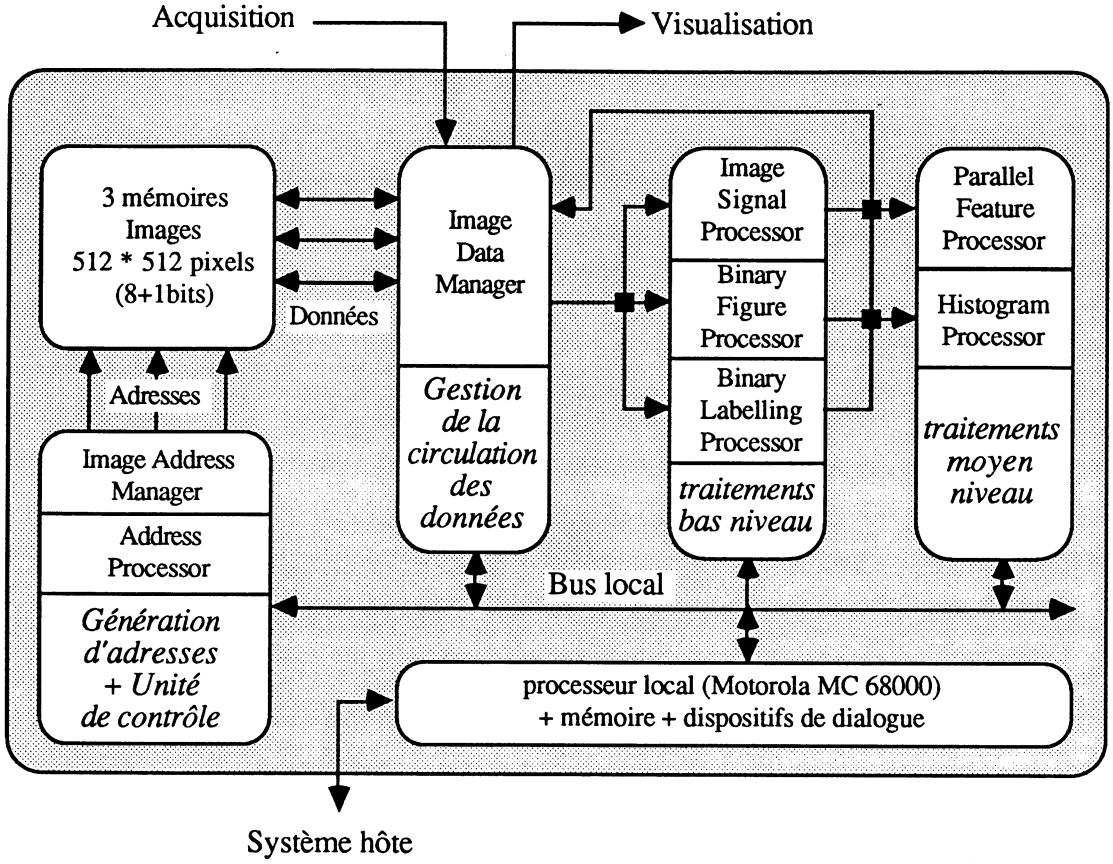


Figure 1.8 : Architecture d'IP "Image Processor"

Cet accélérateur de calcul utilise les deux niveaux de parallélisme précédemment décrits, il traite séquentiellement des zones rectangulaires d'image. Il réalise en parallèle, pour chaque accès aux informations image, en utilisant des techniques d'enchaînement pipeline (niveau macroscopique) :

- une loi d'accès particulière aux données à traiter (ex: rotation d'image),
- une suite de traitements de bas niveau (image -> image résultat),
- la mémorisation de l'image résultat,
- l'extraction en parallèle d'un groupe de primitives sur cette image résultat (tâches de moyen niveau).

Les traitements réalisables par les différents circuits spécialisés que comporte cet accélérateur de calcul sont les suivants:

"Image Signal Processor"

Convolution, morphologie en gris, opérations arithmétiques et logiques entre deux images, corrélation, recherche de motifs binaires.

"Binary Figure Processor"

Erosion, dilatation, amincissement de contours (image binaire).

"Binary Labelling Processor"

Etiquetage d'objets (composantes connexes d'une image binaire).

"Parallel Feature Processor"

Calcul de mesures sur des objets (aire, périmètre, centre de gravité).

"Histogram Processor"

Histogramme, projection suivant les lignes ou les colonnes.

"Address Processor"

Génération des adresses d'accès aux pixels, balayage suivant les lignes ou les colonnes, rotation d'image, rééchantillonnage spatial, correction des déformations géométriques de type erreurs de parallaxe et contrôle du séquencement de l'accélérateur de calcul.

Pour la parallélisation au niveau microscopique, nous citerons uniquement comme exemple le processeur de traitement de voisinage ISP "Image Signal Processor" (KOB 87, TAK88), qui est utilisé dans la machine décrite ci-dessus.

Ce circuit VLSI (> 150000 transistors, technologie BiCMOS 1,8 µm, 120 broches) traite des images disponibles séquentiellement suivant un balayage de type vidéo (ligne à ligne) et, est plus particulièrement destiné à la réalisation d'algorithmes de traitement de bas niveau mettant en oeuvre des voisinages de pixels (exemple : convolution, morphologie en gris, ...).

Un circuit ISP peut réaliser à chaque cycle d'horloge, un calcul mettant en oeuvre un voisinage de trois pixels appartenant à une même ligne ou une même colonne, il est possible d'accroître la taille du voisinage en associant en cascade plusieurs circuits (fig 1.9). Sa structure interne (fig. 1.10) peut être décomposée en deux blocs fonctionnels programmables :

- une unité de gestion de voisinage qui réalise l'interface avec l'entrée séquentielle des pixels, mémorise temporairement les deux dernières lignes de pixels et, émet en parallèle vers l'unité de calcul, les trois pixels du voisinage à traiter;
- une unité de calcul composée d'un réseau d'unités opératives (7 unités arithmétiques, 3 multiplieurs) dans lesquelles se propagent les informations.

La programmation des différents blocs fonctionnels du circuit permet de définir la nature de l'algorithme de traitement réalisé, en agissant sur :

- la longueur des lignes,
- les opérations réalisées par les unités opératives,
- les constantes de calcul,
- l'orientation du voisinage (suivant les lignes ou les colonnes).

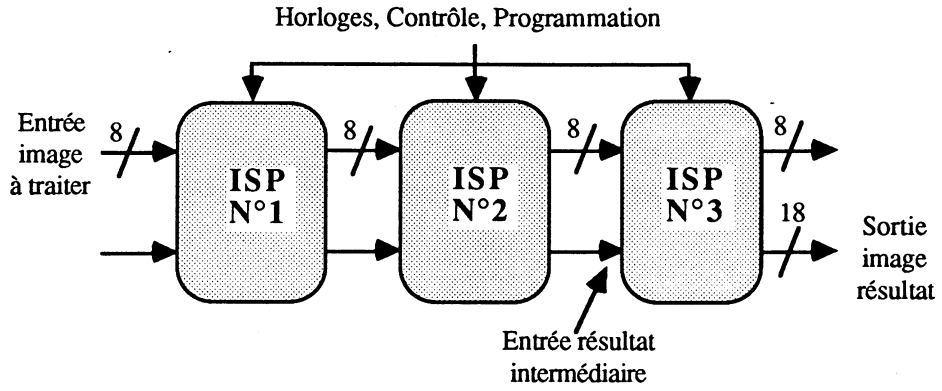


Figure 1.9 Association en cascade de plusieurs ISP

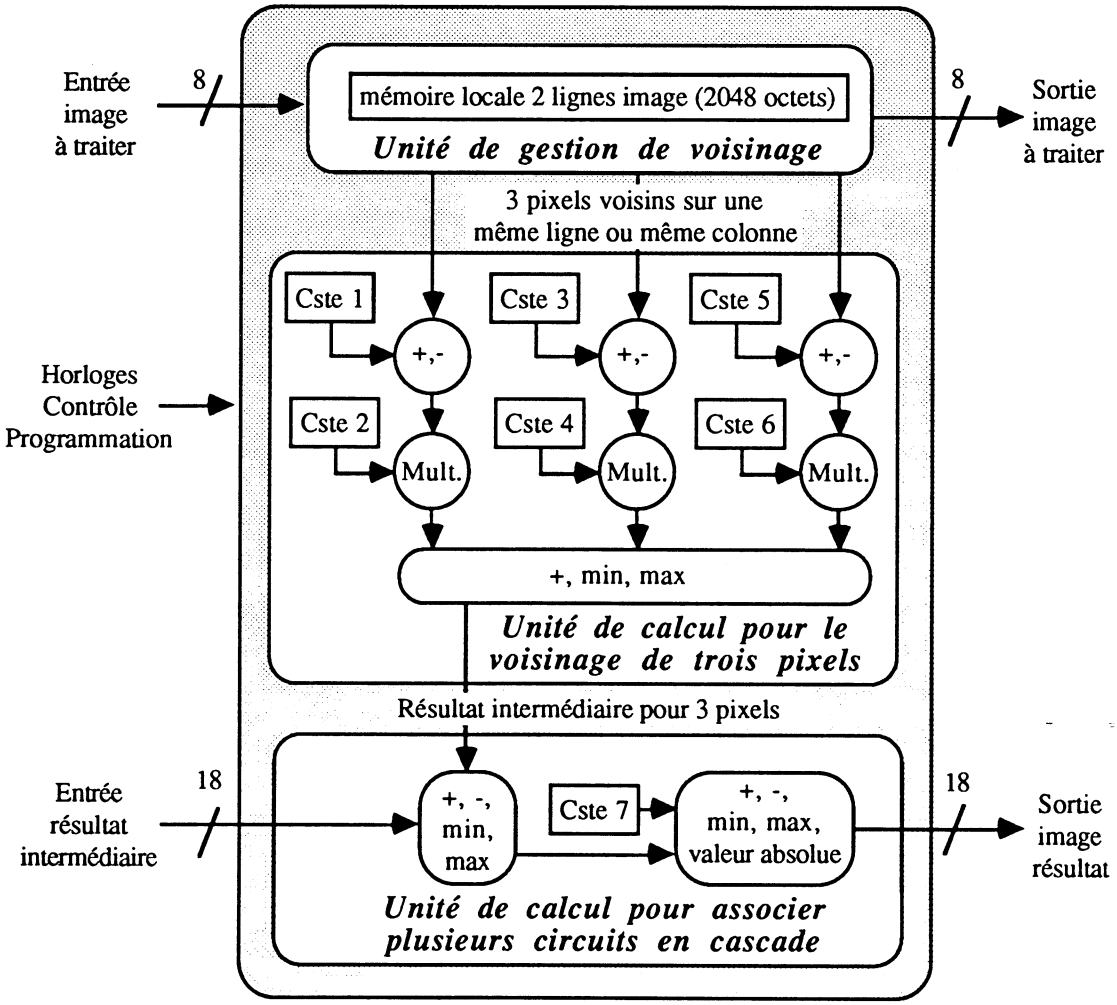


Figure 1.10 : Structure d'ISP "Image Signal Processor"

1.2.3.3) Les architectures MIMD

Principe

Les structures MIMD sont les architectures parallèles les moins strictement définies, que ce soit au niveau de leur architecture matérielle ou des techniques de parallélisation qu'elles mettent en oeuvre. Elles sont en général organisées autour d'un nombre assez faible (4 à 64) d'**unités de traitement autonomes** (identiques ou non) dotées de mémoires de travail pour des variables de calcul locales, qui échangent des informations et exécutent de manière asynchrone les processus qui leur sont alloués.

Ces systèmes utilisent des techniques de parallélisation à fortes ou moyennes granularités privilégiant les calculs et minimisant les dialogues et les synchronisations souvent côuteuses en temps dans les architectures parallèles asynchrones. Les unités de traitement utilisées sont le plus souvent des processeurs intégrés apparentés aux microprocesseurs.

- Les mémoires contenant l'image peuvent être globales ou locales (partagées ou non).
- Les communications entre unités peuvent être réalisées via des bus, une mémoire commune, des réseaux, des chemins de dialogues locaux analogues aux dispositifs rencontrés dans les tableaux de processeurs SIMD.

Ces structures utilisent, pour les traitements de bas et moyen niveaux, des techniques de calcul parallèle qui sont des généralisations voire des combinaisons asynchrones des méthodes employées pour les systèmes synchrones cités dans les deux paragraphes précédents.

- pour les méthodes de parallélisation à "structures de donnée", on utilise des techniques voisines dites : SPMD "Single Procedure stream, Multiple Data stream" (ETC89) ou SCMD "Single Code Multiple Data" (WAL89) où, chaque processeur exécute un même algorithme (tâche) sur des blocs de données différents (zones d'image (RUB82)). Pour minimiser les communications en cours de traitement, les dialogues et les synchronisations entre processeurs sont souvent centralisés et réalisés lorsque tous les processeurs ont terminé l'exécution de leur tâche (mécanismes de "Fork & Join" global).

- pour les méthodes de parallélisation à "propagation d'informations", on utilise des techniques dites : "Wavefront" ou "pipeline asynchrone" (ANA87, KUN84, FUJ88) où, le temps d'exécution des différents processus réalisés par les unités de traitement est variable voire dépendant des données.

La première génération de systèmes MIMD pour le traitement d'images (avant 1984) était basée sur des microprocesseurs d'usage général, et plutôt orientée "architectures à structures de données". Nous pouvons citer pour exemple :

ZMOB : 64 processeurs 8 bits (Zilog Z80) dialoguant par réseaux (ROS78, KUS81)

ROMUALD : 8 processeurs 16 bits (Zilog Z8001) dialoguant par un bus centralisé de grande largeur (BRE78, RUB82)

EDDY : tableau bidimensionnel de 16 couples de processeurs 16 bits (Zilog Z8001), chaque unité de calcul est composée d'un processeur de calcul et d'un processeur gérant des dialogues avec les 8 unités voisines par des entrées-sorties parallèles (TAK83)

La seconde génération (à partir de 1984) de ces systèmes a eu des performances améliorées en utilisant des processeurs commercialisés plus puissants et dédiés au calcul numérique (processeur de traitement du signal (DSP)) ou même dans certains cas, en concevant des processeurs spécifiques.

Systèmes orientés vers des "architecture à structures de données" :

CAPITAN : 4 à 64 DSP 16 bits (Texas Instrument TMS320X) (GAI87) dialoguant par des bus et des réseaux

ODYSSEY : 4 à 32 DSP 16 bits (Texas Instrument TMS32020) dialoguant par bus (GAS87)

ICS : 4 à 8 DSP 16 bits (Analog Device, ADSP2100) dialoguant par bus (THR87)

VPP : 64 processeurs spécifiques 32 bits dialoguant par des réseaux (INO88)

EWC : 4 processeurs spécifiques 32 bits intégrés (T9506) dialoguant par bus (KOB88)

Systèmes orientés vers des "architecture à propagation d'informations" :

TIP3 et 4 : 8 à 64 DSP 16 bits (NEC µPD7281) dialoguant par des réseaux (FUJ88)

WARP : 10 ou plus processeurs spécifiques 32 bits interconnectés linéairement (KUN84, ANA87), INTEL a débuté en 1990 l'échantillonnage d'une version monocircuit VLSI (>500.000 transistors) de ce processeur.

Exemple :

ICS "Image Computing System" est un système parallèle MIMD à forte granularité développé et commercialisé par ANDROX Corp (USA), il se présente sous la forme d'une monocarte de traitement interfaçable à des systèmes hôtes de type Station de travail SUN. Il comporte quatre ou huit unités de traitements autonomes, une mémoire image centralisée et deux bus de communication (fig. 1.11) :

- un bus 32 bits synchrone (40 Méga Octets / Sec.) pour l'accès rapide à la mémoire image;
- un bus 16 bits asynchrone pour les dialogues entre processeurs.

Le nombre de transactions sur le bus mémoire est réduit par la présence au niveau de chaque unité de traitement, d'une mémoire cache qui mémorise localement des zones bidimensionnelles d'image. Ce cache est rempli ou vidé par une DMA rapide concurremment au fonctionnement du processeur. Chaque unité de traitement comporte :

- un microprocesseur de traitement du signal (ADSP2100 Analog Device);
- 48 Koctets de mémoire locale pour le code et les variables locales;
- 64 Koctets de mémoire cache;
- un circuit intégré spécifique (Prédiffusé) intégrant la DMA spécialisée, le contrôle du cache et l'interface avec le bus mémoire;
- un dispositif de dialogue interprocesseur par passage de messages.

Un microprocesseur dédié aux applications de synthèse graphique (TMS34010 Texas Instrument) gère les dispositifs d'acquisition et de visualisation de l'image.

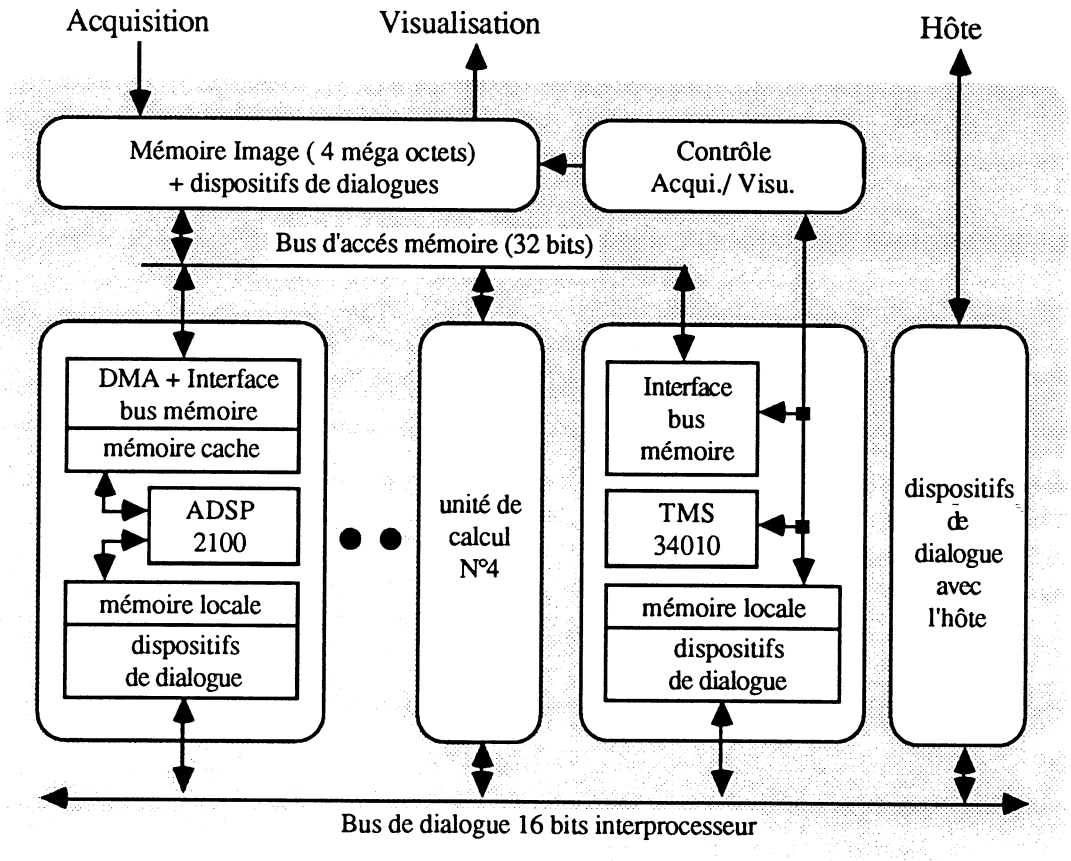


Figure 1.11 : Architecture d'ICS

1.2.4) Conclusion

La comparaison globale des approches architecturales et des réalisations matérielles associées, qui ont été examinées dans les trois derniers paragraphes, est très difficile voire impossible, si l'on désire simultanément prendre en compte de multiples critères au niveau : fonctionnel (performances, polyvalence, limitations), matériel (coût, complexité) ou logiciel (complexité de programmation et de formulation du parallélisme).

Les performances des systèmes de traitement d'images actuellement réalisés ou en projets, sont fortement liées au nombre d'unités de traitement qu'ils comportent et aux techniques utilisées pour leur réalisation. Ils évoluent entre, la monocarte électronique produite en petite série, utilisant des composants standards, et d'énormes machines très proches des supercalculateurs utilisant intensivement des circuits VLSI sur mesure, dont la diffusion est extrèmement restreinte voire unitaire. Une telle disparité matérielle rend fallacieuse toute comparaison qui se baserait uniquement sur les performances pures.

Si l'on se restreint aux systèmes de traitement d'images utilisés en "productique" industrielle, qui nous intéresse plus particulièrement, les critères de comparaison les plus importants sont essentiellement d'ordre économique :

- le coût des systèmes (logiciel + matériel) dont le prix moyen est de l'ordre de 200000 francs;

- les rapports "performances / coût" et "performances / complexité de programmation".

Dans le cas où la comparaison des approches architecturales est réalisée en privilégiant les critères ci-dessus, il est possible de regrouper en cinq points, les diverses conclusions et constatations des nombreux travaux de synthèses réalisés sur les architectures parallèles pour le traitement d'images, au niveau logiciel (YAL85, LOU85, DEW86, DUF88, MAR88) ou matériel (ZAV84, CAS85, BAS86, THO87, DAW88).

Pour les traitements de bas niveau et de nombreux traitements de moyen niveau manipulant des structures de données statiques, les systèmes les plus performants sont actuellement des architectures parallèles synchrones : système pipe-line et systolique, tableau de processeurs SIMD "processor array".

Une énorme majorité des systèmes actuellement commercialisés utilise des mécanismes de calcul parallèle à propagation d'informations, la prépondérance de cette approche architecturale est principalement due à deux facteurs :

- la disponibilité sur la marché des composants électroniques, de nombreux circuits VLSI correspondant à ces modèles de calcul, qui permettent de réaliser des systèmes avec d'excellents compromis (performances/coût);

- le fonctionnement de ces structures parallèles est proche des systèmes informatiques traditionnels et de leurs méthodes de programmation, ils sont donc faciles à appréhender (formulation "séquentielle" des algorithmes, centralisation des données dans une seule mémoire de très grande taille).

Les tableaux de processeurs sont relativement simples à intégrer sous forme de VLSI, ils offrent pour le bas niveau d'excellentes performances et des possibilités intéressantes du fait de leur fonctionnement programmé, mais leur développement est handicapé : par des outils de développement logiciel assez frustres et une grande complexité de formulation du parallélisme pour les tâches de moyen niveau (changement de la structure des données), qui les réservent à un public de spécialistes.

La parallélisation des algorithmes de traitement de moyen niveau les plus sophistiqués (présentant des branchements multiples, des accès non déterministes aux informations, et gérant des structures de données dynamiques ou des pointeurs) est extrêmement difficile si l'on doit utiliser des formulations parallèles synchrones.

Les structures les plus souples à utiliser, que ce soit au niveau de la formulation parallèle des algorithmes existants ou à venir, ou de l'utilisation intensive des différentes unités de traitement, sont évidemment les systèmes MIMD, ces structures sont efficaces pour l'ensemble des tâches de moyen niveau, mais leur rapport "performances / coût" pour les tâches de bas niveau ne peut pas rivaliser avec les structures parallèles synchrones beaucoup plus spécialisées.

Au vu des progrès actuels des travaux sur la parallélisation d'algorithmes et de la technologie microélectronique, cet "état de fait" entre les différentes approches architecturales devrait être conservé à court terme, ce malgré la montée en puissance extrêmement rapide des systèmes MIMD, qui bénéficient de processeurs VLSI de plus en plus puissants, d'assez faible coût, développés pour des domaines connexes à l'analyse d'images (par exemple le traitement du signal ou la synthèse d'image).

1.3) Notre approche

1.3.1) L'idée de base

Pour réaliser des systèmes d'un coût acceptable, et pallier aux limitations des approches architecturales présentées dans les paragraphes précédents, certains concepteurs de systèmes parallèles d'analyse d'images (DUF88, WAL89) ont proposé d'orienter les recherches vers des systèmes "hybrides" qui associeraient dans une même structure matérielle :

- des architectures parallèles synchrones compactes et très performantes pour certaines gammes de problèmes de traitement d'image bas et moyen niveaux;

- des architectures parallèles asynchrones MIMD offrant une grande souplesse de parallélisation pour les problèmes les plus complexes, et de bonnes perspectives pour la mise en oeuvre des problèmes à venir.

Cette approche offre en théorie des fonctionnalités très séduisantes, mais la conception de tels systèmes est nettement plus complexe que les approches architecturales classiques. Il est en effet nécessaire de prendre en compte en plus des problèmes de base (calculs/communications) liés à tout système parallèle, des aspects architecturaux qui sont spécifiquement dus à cette dichotomie "synchrone/asynchrone" :

- les communications, accès aux données communes, et synchronisation entre deux structures parallèles qui sont à la base foncièrement différentes;

- les redondances entre ces deux structures au niveau des calculs et des ressources matérielles (mémoire de travail, dispositifs de communication).

Les premières études (DUF88, WAL89) dans cette voie se sont orientées vers des systèmes, pour lesquels la séparation "synchrone / asynchrone" était très nette, et dont la structure correspondait à l'association de deux architectures parallèles distinctes dialoguant entre elles via des dispositifs de communication rapide ou une mémoire centralisée partagée.

Ces études ont démontré l'efficacité du concept au niveau des performances et de la souplesse de parallélisation, et mis en évidence le goulot d'étranglement de ce type de systèmes :

les communications entre dispositifs synchrones et asynchrones et la gestion de celles-ci.

Cette limitation et un souci d'homogénéité et de réduction des coûts au niveau de la structure globale du système, nous a conduit à nous orienter vers une autre approche architecturale de ce type d'association, pour laquelle ces dialogues seraient fragmentés et distribués dans toute la structure :

des systèmes parallèles MIMD homogènes à forte granularité composés d'unités de traitement autonomes identiques où chaque unité est un système de traitement distribué associant un microprocesseur 32 bits et un accélérateur de calcul parallèle synchrone (coprocesseur spécialisé) pour réaliser rapidement certaines tâches spécifiques du traitement d'image bas et moyen niveaux.

A notre connaissance, cette approche architecturale n'a pas encore été étudiée dans le cadre des systèmes parallèles pour l'analyse d'images, mais des idées similaires sont en cours de développement pour la synthèse d'images (DIE88), la compression d'image couleur (KAN88) et certains dispositifs de traitement du signal pour des applications militaires. L'émergence de ce type de systèmes parallèles spécialisés, qui à notre avis va connaître à court terme un développement très important dans de multiples domaines, est très récente et est selon nous due principalement à quatre raisons :

- la démocratisation des techniques d'intégration VLSI sur mesure, qui permettent de réaliser des accélérateurs de calcul très compacts;
- la possibilité de concevoir des systèmes parallèles MIMD spécialisés, tout en utilisant au mieux les microprocesseurs du commerce de plus en plus puissant et de coûts modérés, qui disposent d'outils de développement logiciel très élaborés;
- la montée en puissance des microprocesseurs du commerce qui permet de réaliser de plus en plus de tâches auparavant réservées à des systèmes spécialisés;
- cette approche est facile à appréhender et est proche des systèmes informatiques traditionnels avec tous les avantages qui en découlent (programmabilité, intelligibilité du système, appel à des composants grand public connus,...).

1.3.2) Les choix architecturaux

Les orientations architecturales de base que nous avons décidées d'adopter pour la mise en oeuvre matérielle de cette approche, sont étroitement liées aux différents types de parallélisation macroscopique utilisable au niveau du traitement d'images industriel. Il est possible par exemple d'utiliser simultanément plusieurs unités de traitement autonomes pour :

- traiter différentes régions d'une même image;
- appliquer sur une même image des algorithmes de traitement différents;
- traiter des images acquises à des temps différents (traitement à la chaine);
- traiter, en utilisant un même algorithme, différentes parties d'une même image (techniques "SCMD", BRE82, YAL85, WAL82, cf 1.3.3.3) en échangeant des informations en cours de traitement.

Pour utiliser facilement ces diverses possibilités de parallélisation, nous avons orientés nos recherches vers des systèmes parallèles composés de modules de traitement autonomes interconnectés, où chacun de ces modules est une petite machine d'analyse d'images complète (microprocesseur, accélérateur de calcul, mémoire image, dispositifs de communications).

Les deux principaux intérêts de cette approche sont d'une part de pouvoir moduler la puissance des systèmes en utilisant plus ou moins de modules de traitement identiques, et d'autre part, de réduire les contraintes au niveau des performances d'un module qui ne doit pas nécessairement, à lui tout seul, résoudre l'ensemble de l'application.

1.3.2.1) Architecture macroscopique

Pour permettre la mise en oeuvre des différentes méthodes de traitement parallèle citées dans le paragraphe précédent, l'architecture macroscopique que nous avons décidé d'adoptée pour notre système MIMD, est schématiquement organisée de la manière suivante (fig 1.12) :

- une module réalisant l'interface avec le capteur et la numérisation de l'image;

- un nombre variable de modules de traitement;

- des dispositifs de communications asynchrones entre les différents modules, par passage de message, de deux types :
  * des dispositifs très rapides, transférant les données par paquets, pour les échanges rapides de gros volumes d'informations (image, ou zone d'image), ils permettent entre autres de distribuer l'image à traiter dans les mémoires locales des différents modules de traitement;
  * des dispositifs plus lents, pour les communications diverses (données, contrôle), la synchronisation des dialogues rapides entre modules, les dialogues avec l'hôte et le téléchargement des programmes vers les unités de traitement.

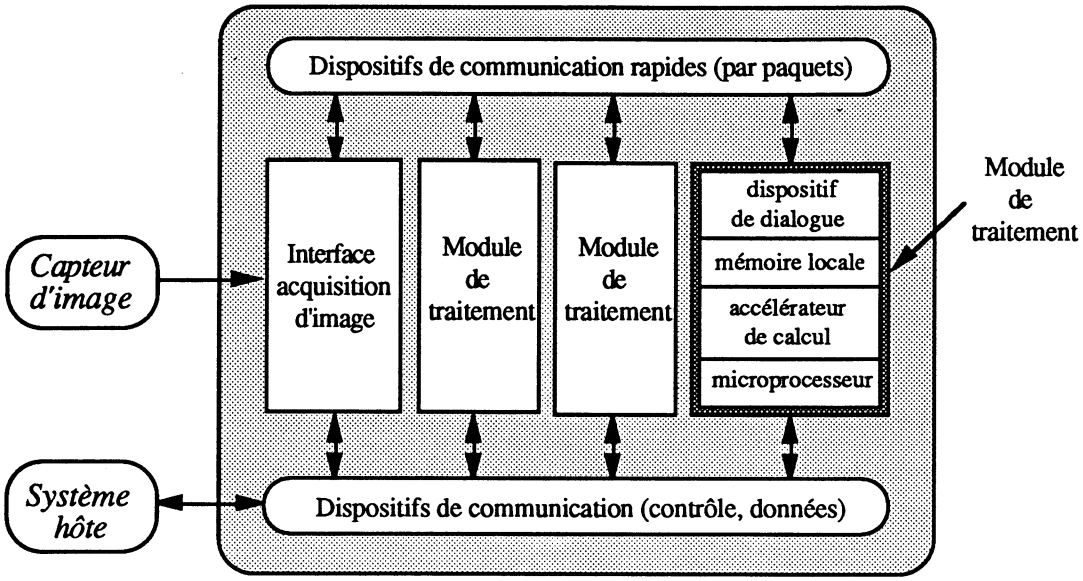


Figure 1.12 : Architecture macroscopique de type MIMD des systèmes cibles

Cette architecture doit permettre de réaliser des systèmes comportant un nombre variable d'unités de traitement. La distribution de l'image à traiter dans les mémoires locales des différentes unités de traitement doit être entièrement programmable.

1.3.2.2) Architecture microscopique

L'architecture que nous avons décidé d'adopter au niveau de la structure interne du module de traitement est globalement organisée autour des blocs fonctionnels suivants (fig 1.13) :

- un accélérateur de calcul spécialisé qui utilise en interne des mécanismes de calcul parallèle synchrone à propagation d'informations (techniques pipeline et systolique), et est capable de réaliser certaines tâches courantes du traitement d'images bas et moyen niveaux ; il traite des informations disponibles séquentiellement, qui ont été préalablement mémorisées dans les mémoires images du module (pas de traitement à la volée);

- un dispositif qui réalise l'interface entre l'accélérateur de calcul et les mémoires images, réalise les calculs d'adresses et l'accès aux informations en mémoire, et gère le contrôle du séquencement de l'accélérateur de calcul;

- un microprocesseur 32 bits qui se charge des phases de traitement ne pouvant être réalisées par l'accélérateur (exemple : les tâches de traitement d'images de haut niveau), et assure la gestion du module (communications, allocation de tâches à l'accélérateur de calcul);

- un dispositif d'interface, doté en interne de tampons de mémorisation, assure les échanges rapides d'informations entre les différents modules que comporte le système, ces échanges sont réalisés de mémoire à mémoire et sont effectués sous le contrôle des microprocesseurs locaux des modules concernés. Le module peut, au moyen de ce dispositif, acquérir et stocker dans sa mémoire locale, l'image ou la zone d'image qu'il doit traiter;

- une ou plusieurs mémoires de grandes tailles pour stocker les images, les programmes du microprocesseur local et les variables de travail, elles sont accessibles de manière aléatoire par : le microprocesseur, le dispositif d'interface de l'accélérateur de calcul, l'interface de communications rapides;

- un bus système interne par l'intermédiaire duquel sont réalisées toutes les communications entre les différentes unités du module, des protocoles particuliers d'utilisation de ce bus (transferts en rafales) permettent de réaliser des échanges d'informations très rapides entre la ou les mémoires images et l'accélérateur de calcul et les dispositifs de communication rapides inter modules.

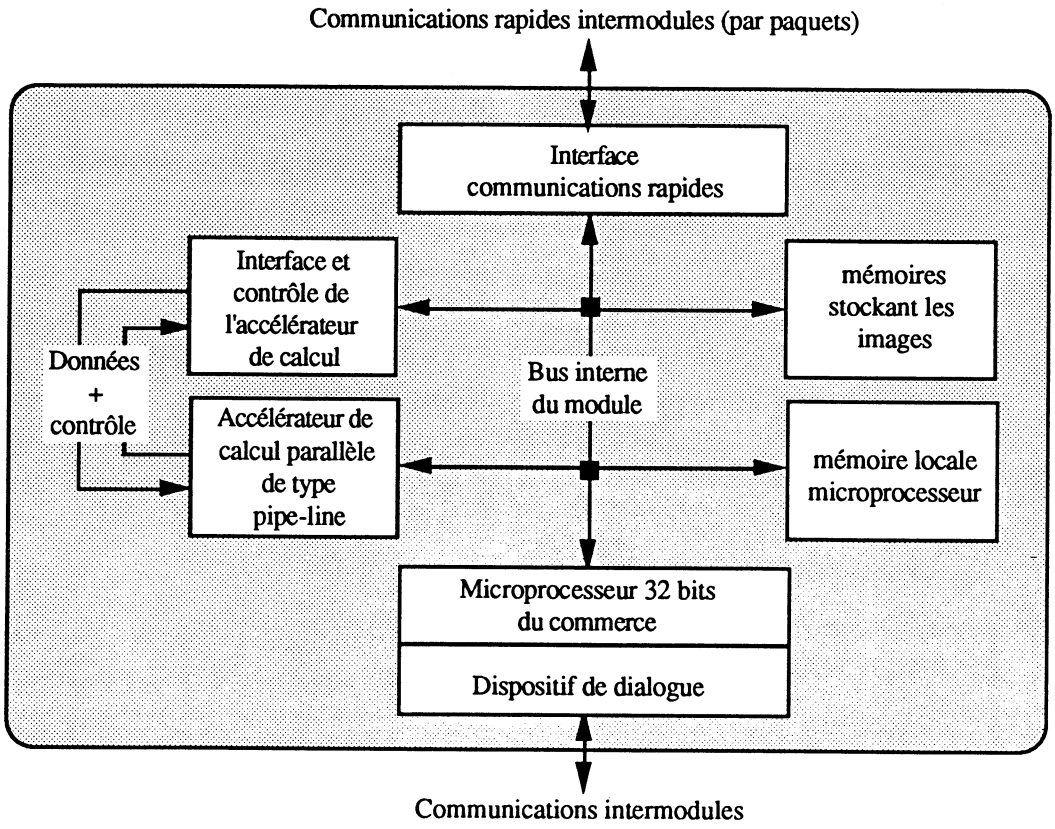


Figure 1.13 : Architecture du module de traitement

Au niveau de l'accélérateur de calcul parallèle synchrone, nous avons préféré nous orienter vers des architectures à "propagation d'informations" plutôt que vers des architectures à "structures de données" pour trois raisons : elles sont très compactes, nécessitent des dispositifs de contrôle ainsi que des outils logiciels réduits, et permettent d'utiliser des mémoires images centralisées, ce qui simplifie les dispositifs permettant à d'autres entités d'y accéder (ex: microprocesseur local).

1.3.3) Les difficultés et l'orientation de l'étude

Le système parallèle qui a été brièvement présenté dans les deux paragraphes précédents, est un projet de machine dont la mise en oeuvre des choix architecturaux soulève un grand nombre de problèmes combinés, matériel ou logiciel, qui doivent être étudiés en détail. Au vu des travaux actuels sur les systèmes parallèles pour le traitement d'images et des machines déjà commercialisées, cette approche est économiquement viable si le module de traitement satisfait à deux conditions :

- ses performances doivent être comparables avec celles des systèmes de traitement d'images actuels de milieu de gamme (nous prendrons comme base de comparaison pour la suite de cette étude le système Hitachi IP "Image Processor" (TAK88) cité en 1.2.3.2) qui, à notre avis, est le système actuel proposant le meilleur rapport performances/coût;
- sa réalisation matérielle doit être de petite taille, une monocarte électronique de format "standard" (environ 6 dm^2).

En l'absence d'études antérieures sur ce type d'architecture, il n'était pas raisonnable d'essayer de prendre en compte, en une seule fois, la totalité des problèmes. Nous avons décidé dans un premier temps, pour démontrer la faisabilité de cette approche et nous donner des briques de base pour la suite de cette étude, de réaliser des préétudes sur les trois groupes de problèmes qui semblent être les clefs de cette approche :

- l'architecture macroscopique du système (communications) et sa gestion logicielle;
- les bus rapides d'accès aux mémoires images et l'interface avec l'accélérateur de calcul;
- l'architecture de l'accélérateur de calcul et ses méthodes de parallélisation.

Le premier groupe de problèmes a été étudié par Mr A. Pirson dans le cadre de sa thèse "Conception et simulation d'architecture parallèle distribuée pour le traitement d'images". Ce présent travail de recherche porte sur la parallélisation des algorithmes et la réalisation matérielle de l'accélérateur de calcul.

1.4) Organisation de la thèse

Nous avons présenté dans ce premier chapitre un état de l'art sur les systèmes et architectures parallèles dédiés au traitement d'images et, défini le cadre et les principales orientations de ce travail de recherche. La suite de ce manuscrit de thèse est organisée de la manière suivante :

la deuxième partie est consacrée à l'étude architecturale du coprocesseur de traitement d'images, elle présente les choix qui ont été adoptés au niveau de son interfaçage avec le module de traitement et des concepts de parallélisation qu'il utilise, son architecture générale, une première décomposition de sa structure interne, ses mécanismes de séquencement et d'accès aux données, les inter-actions et les protocoles de communications entre les différents blocs fonctionnels qui le composent;

les troisième et quatrième parties présentent deux études architecturales (parallélisation d'algorithmes, structure matérielle) de processeurs spécialisés intégrables sous forme de circuits VLSI, pouvant être utilisés pour réaliser l'accélérateur de traitement d'images :

- "Image Neighborhood Processor" pour des traitements de bas niveau;
- "Image Feature Processor" pour des traitements de moyen niveau;

enfin, nous présentons brièvement en conclusion les performances d'une version embryonnaire de cet accélérateur de traitement d'images (dispositif de test de la première version intégrée du processeur pour les traitements bas niveau INP20 décrit dans le paragraphe n° 3), discutons des avantages et des inconvénients de cette approche et indiquons les améliorations envisagées pour l'avenir.

Deuxième partie

Etude architecturale d'un coprocesseur de traitement d'images

How should I arrange
My garden, now that the walls
Around it are gone ?

David W. Mizell

Sommaire de la seconde partie

2.4.3) Architecture des unités de calcul

2.4.3.1) Principes généraux

2.4.3.2) Interfaces externes

2.4.3.3) Architecture interne

2.5) Conclusion

2.1) Introduction

Au coeur de la majorité des algorithmes de traitement d'images de bas et moyen niveau manipulant des structures de données statiques (tableaux de pixels), on retrouve une partie de calcul répétitif ("corps" ou "noyau") qui est appliquée à chacun des pixels à traiter et dont la structure bien qu'assez simple, nécessite globalement un très grand nombre d'opérations. Les techniques "pipeline" et "systolique" de calcul parallèle, pour accélérer l'exécution des noyaux de calcul spécifiques au traitement d'images, ont été abondamment étudiées et ces travaux de recherches ont débouchés sur la réalisation, sous forme de circuits VLSI, de nombreux opérateurs adaptés à ce type de tâches (cf. paragraphe 1.3.2.2).

Mis à part les circuits conçus pour le système Hitachi IP200 (TAK88, HIT88), la plupart de ces opérateurs intégrés déjà réalisés sont des structures parallèles unifonctions, qui réalisent une seule tâche de calcul avec des performances maximales et/ou une complexité matérielle minimale. Pour réaliser des systèmes de traitement d'images qui soient à la fois performants et polyvalents, il est nécessaire d'utiliser un grand nombre de ces circuits très spécialisés et donc, de concevoir des réseaux d'interconnexions complexes destinés à reconfigurer les échanges d'informations entre ces différents opérateurs. En outre, la plupart de ces circuits intégrés ont été conçus pour des systèmes où l'on traite directement, sans mémorisation, l'image numérique issue du capteur qui est disponible suivant des standards vidéos destinés à une visualisation de l'image.

Cette approche conceptuelle, utilisant des processeurs unifonctions, nous a semblé inadaptée pour la conception de notre accélérateur de calcul car, même en utilisant des circuits à haute intégration, le volume d'électronique (opérateurs de traitement + réseaux d'interconnexions) nécessaire pour réaliser un "coprocesseur" relativement polyvalent est incompatible avec la taille limitée que nous nous sommes fixés pour le module élémentaire de traitement (monocarte électronique) (cf. paragraphe 1.3.3).

Pour respecter ces contraintes d'encombrement tout en conservant un large éventail de possibilités, nous avons du faire trois choix de base pour orienter les études de l'architecture interne de notre "coprocesseur de traitement d'images" :

- réduire la taille et la complexité du coprocesseur en minimisant voire en supprimant les dispositifs d'interconnexions entre les différents opérateurs qui le composent;

- réduire le nombre et la diversité des opérateurs de traitement en utilisant des dispositifs programmables ou configurables, qui soient capables de réaliser plusieurs types d'algorithmes de traitements;

- définir pour le coprocesseur une architecture modulaire permettant d'intégrer un nombre variable d'opérateurs de traitement, suivant la complexité, la taille et les techniques de réalisation de ces derniers.

L'analyse structurelle, les spécifications et l'optimisation de l'architecture d'un système électronique (module de traitement et coprocesseur), qui devra être réalisé en interconnectant entre eux, plusieurs circuits intégrés sur mesure, est un travail complexe et de longue haleine. Dans notre cas cette tâche est rendue encore plus complexe, car les fonctionnalités et les structures de ces circuits spécifiques sont elles-mêmes à définir.

Si la structure générale de notre module de traitement correspond à une architecture classique de système à base de microprocesseur, et donc ne pose pas de problème majeur de réalisation, la définition et la réalisation du coprocesseur de traitement d'images suivant les orientations définies dans le paragraphe précédent est un tout autre problème.

L'objectif de ce chapitre est d'étudier l'interfaçage du coprocesseur de traitement d'images dans le module de traitement, de spécifier les tâches qu'il doit réaliser et de proposer pour celui-ci une architecture externe et interne capable de répondre aux choix énoncés ci-dessus.

2.2) Environnement externe du coprocesseur

Notre coprocesseur de traitement d'images doit s'intégrer dans un environnement déjà prédéfini (PIR90) tout en minimisant son encombrement. Il est donc impératif de prendre en considération dès les premières phases de sa conception, certaines contraintes d'interfaçage dûes à son environnement externe (architecture du module) afin de minimiser la logique électronique nécessaire à son intégration dans le module. Le but de ce chapitre est de présenter plus en détail l'architecture du module de traitement et ses intérêts et influences au niveau de l'architecture externe et des fonctionnalités du coprocesseur.

2.2.1) Architecture macroscopique du module

Le module de traitement est un petit système de traitement d'images autonome qui est organisé autour d'un bus système local 32 bits (adresses, données et contrôles) sur lequel se greffent différentes unités (cf figure 2.1). Ce bus permet de véhiculer les informations entre les différentes unités. L'unité centrale (transputer) est le maître du module et arbitre ce bus qu'il cède sur requêtes aux différentes unités qui le sollicitent.

Deux types d'unités sont connectés à ce bus : les unités actives et les unités passives. Nous appelons unités actives celles qui sont capables de contrôler le bus afin d'accéder aux ressources qui y sont connectées ou, qui sont utilisées pour échanger des informations d'une unité passive à une autre (exemples d'unités actives : unité centrale, contrôleur d'accès à la mémoire, ...). Nous appelons unités passives, celles qui sont incapables d'utiliser seules le bus et qui requièrent pour leurs échanges d'informations avec l'extérieur, l'intervention d'une unité active (exemples d'unités passives : mémoires, ports d'entrée/sortie,....).

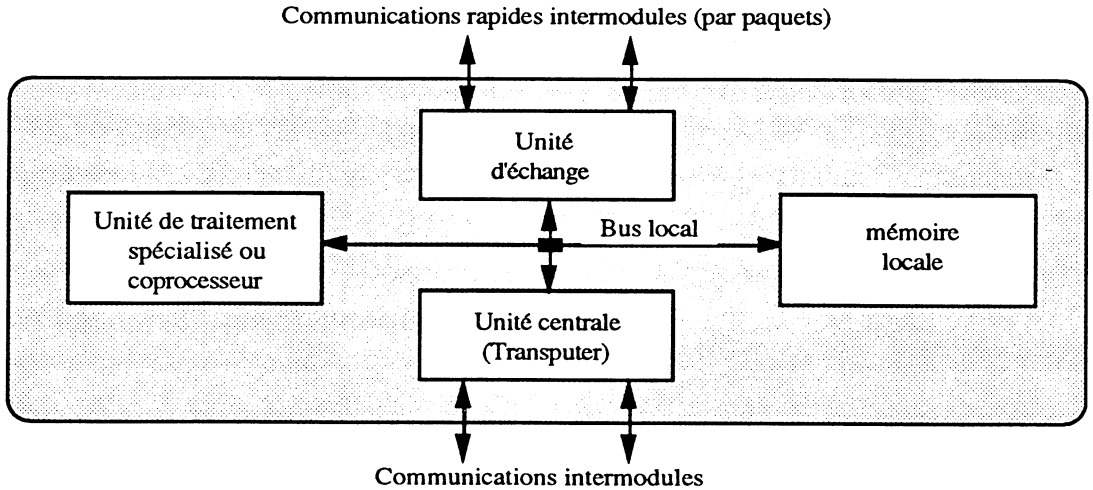


Figure 2.1 : Architecture du module élémentaire de traitement

2.2.2) Architecture interne du module

2.2.2.1) Unité d'échange

L'unité d'échange réalise les dialogues synchrones à haute vitesse entre deux modules. Ces dialogues sont basés sur des transferts en parallèle (32 bits) par paquets, de données structurées (images ou zones d'images), entre les mémoires locales des différents modules. Ces communications sont réalisées sous le contrôle de l'unité centrale qui gère la synchronisation des échanges via des dialogues par les liens série du transputer.

Cette unité particulière permet entre autres des dialogues entre le module et des modules spécialisés dans l'acquisition ou la visualisation des images. Les communications gérées par l'unité d'échange sont basées sur le principe des macro-liens qui peuvent être considérés comme des extensions haute vitesse des liens de communication des "transputers" INMOS (PIR 90) tout en respectant le modèle OCCAM.

2.2.2.2) Mémoire locale

La mémoire locale est organisée en mots de 32 bits de large et supporte tous les modes d'accès de l'unité centrale, chacun des octets qui composent un de ses mots est accessible individuellement. Elle est banalisée et permet de stocker des images ou des zones d'images, des résultats intermédiaires de calcul et le code du programme exécuté par le microprocesseur du module. Nous avons préféré une mémoire banalisée pour stocker les images aux classiques mémoires image de la plupart des sytèmes, car celle-ci permet une plus grande latitude au niveau du nombre d'images mémorisées, de leurs tailles ou de la dynamique de représentation des pixels qui les composent.

Cette mémoire est accessible de manière totalement aléatoire par toute unité active et permet de stocker indifféremment toutes données codées sur un octet ou plus, les images ayant des pixels en général codés sur 8, 16 ou 32 bits.

Si l'unité de traitement spécialisée est active, cette mémoire peut être accessible par deux entités distinctes (unité centrale, unité de traitement), il y a donc des risques de conflit d'accès ou d'incohérence de l'information. La technique qui a été adoptée au niveau du module pour supprimer ce type de problèmes, est qu'une seule unité (l'unité centrale) gère les ressources mémoire et alloue celles-ci aux autres unités. En outre, l'unité centrale sera toujours la plus prioritaire pour le contrôle du bus système et, de fait, pour l'accès à la mémoire.

2.2.2.3) Unité centrale

L'unité centrale du module (transputer doté d'une mémoire locale de travail non partagée) est chargée d'une quadruple mission, elle assure :

- la gestion des différentes unités du module : elle gère la mémoire et l'unité d'échange en allouant des zones aux paquets d'informations entrant, l'emplacement des paquets sortant et les zones de travail réservées à l'unité de traitement.
- l'exploitation des différentes unités dans le cadre des traitements : elle pilote entre autres l'unité de traitement spécialisée comme un coprocesseur auquel elle alloue des tâches et des ressources mémoire.
- la réalisation de certaines tâches de calcul qui ne peuvent être réalisées par l'unité de traitement spécialisée (tâches de haut niveau).
- la gestion des dialogues et des synchronisations avec les autres modules.

Les mécanismes de gestion multi-tâches intégrés du transputer et une programmation appropriée de celui-ci lui permettent de réaliser en concurrence les quatre activités citées ci-dessus.

2.2.2.4) Bus local du module

Le module s'articule autour d'un bus système partagé par toutes ses ressources. Ce bus est principalement utilisé pour les accès à la mémoire par les unités actives. Les échanges d'informations sur le bus sont de deux types, des cycles de transfert simple (un seul mot de 8, 16 ou 32 bits) ou des cycles de transferts plus sophistiqués pour un groupe de mots de 32 bits ("Burst transfert", SCH89)

La politique de gestion (arbitrage) adoptée pour ce bus consacre l'unité centrale (transputer) comme le maître du bus et les autres unités du module sont considérées comme des périphériques esclaves plus ou moins sophistiqués. Ces périphériques peuvent uniquement ravir à l'unité centrale le bus après une autorisation de celle-ci, cette méthode permet d'éliminer les conflits d'accès.

Le bus est le seul moyen dont dispose l'unité centrale pour accéder aux ressources qu'il pilote, privé de celui-ci, il perd tout contrôle du module, il est donc impératif d'assurer au niveau des périphériques une libération du bus automatique. Nous avons pour cela défini l'unité centrale comme l'unité la plus prioritaire pour le contrôle du bus, ses demandes d'accès au bus forcent automatiquement la libération de celui-ci par les autres unités.

2.2.2.5) Coprocesseur spécialisé

L'unité de traitement spécialisée est l'organe de calcul du module qui est chargé des tâches de calcul lourdes, c'est donc elle qui détermine la spécificité du module. Dans notre cas, nous nous sommes orientés vers une unité de traitement spécialisée dédiée aux opérations de traitement d'images bas et moyen niveaux (coprocesseur de traitement d'images).

Dans cette architecture du module déjà prédéfinie, cette unité spécialisée peut être connectée au bus du module de deux manières (cf figure 2.2) :

- cette unité de traitement est active, capable de gérer de manière autonome le bus et d'accéder aux informations à traiter stockées dans la mémoire locale.
- cette unité est passive et il est donc nécessaire d'utiliser une unité active externe (contrôleur de DMA ou unité centrale) pour réaliser tous les échanges d'informations avec la mémoire locale.

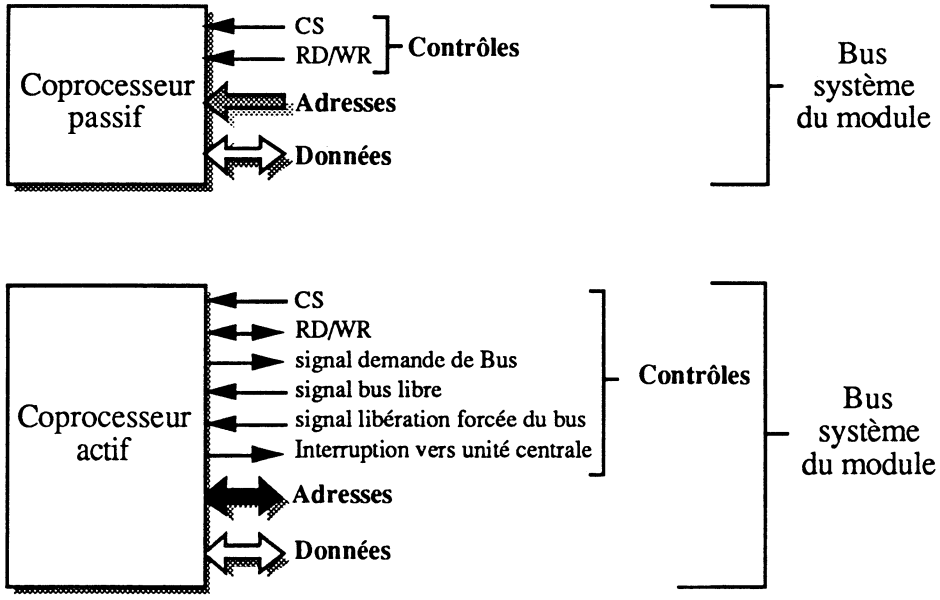


Figure 2.2 : Interface externe des unités de traitement actives ou passives

Dans le cas où cette unité de traitement est active, elle doit tolérer un dispositif d'arbitrage pour son accès à la mémoire locale du module partagée. Cela signifie qu'elle doit être suspensive (son mécanisme de séquencement doit permettre une interruption temporaire de ses tâches de calcul) et tolérer des vols de cycle par l'unité centrale qui est plus prioritaire qu'elle.

L'unité de traitement passive est dépourvue de mécanisme d'adressage à la mémoire locale du module, tous les transferts d'informations doivent être réalisés par une unité active externe, elle se comporte dans ce cas comme un coprocesseur fortement couplé avec cette dernière (OKL88).

Du fait de la structure du bus et de ses mécanismes d'accès, l'unité de traitement, quelle soit active ou passive doit être capable de traiter des informations disponibles de manière "asynchrone" (les échanges d'informations avec la mémoire locale ne sont pas réalisés à des moments déterminés et dépendent des transactions en cours sur le bus du module), donc cette unité doit pouvoir suspendre temporairement l'exécution des tâches de traitement qu'elle réalise.

2.2.3) Intérêts et contraintes liés à cet environnement

2.2.3.1) Accès aux informations images

Les données à traiter (images ou sous-images) sont disponibles dans la mémoire locale du module. Cette mémoire à accès aléatoire permet donc une grande latitude d'accès aux informations par l'unité centrale ou les unités de traitement spécialisées actives. Cette possibilité de traiter une image stockée dans une mémoire, et d'y accéder librement, permet principalement dans le cas des architectures parallèles à propagation d'informations :

- un fonctionnement asynchrone des systèmes de traitement spécialisés et de la source d'informations (capteur d'images);
- le traitement de l'image à des cadences plus rapides que l'acquisition de celle-ci par le capteur d'images, si la mémoire locale est suffisamment rapide;
- la possibilité de traiter sélectivement certaines zones de l'image et non la totalité de celle-ci;
- la possibilité de traiter les informations (pixels) dans un ordre quelconque.

Cette dernière possibilité est sans aucun doute la plus intéressante, car elle permet une simplification de la mise en oeuvre de certains algorithmes de traitement dans les architectures à propagation d'informations, en utilisant des lois d'accès aux données beaucoup plus sophistiquées que le classique balayage vidéo ligne à ligne (WEI87, TAK88, DAV80,MOL88, HIT88).

La mémoire locale est organisée en mots de 32 bits où chaque octet est accessible séparément, elle est banalisée et est vue comme toute mémoire système d'un microprocesseur évolué, ce qui permet de stocker des données de structures et de natures différentes, par exemple des images

dont les pixels sont codées sur 8 ou 16 bits, ou des informations géométriques (positions dans l'image représentées par des couples d'entiers 16 bits). La multiplicité des données qui peuvent être stockées dans cette mémoire complexifie la gestion de cette dernière par une unité de traitement spécialisée active. Par contre, nous considérons que cette organisation mémoire est la plus souple qu'il existe, que ce soit au niveau des possibilités d'accès aux informations images ou au niveau de la gestion de la mémoire par l'unité centrale. Cette approche correspond bien à l'évolution technologique des mémoires dont la capacité est nettement supérieure aux besoins actuels de stockage d'images.

2.2.3.2) Interfaçage de l'unité de traitement

Si l'architecture et les possibilités des systèmes parallèles à propagation d'informations ont été abondamment étudiés, il n'en est pas de même pour leur interfaçage avec des mémoires à accès aléatoire et des bus système partagés. Les principaux problèmes que nous allons rencontrer, au niveau de l'architecture du coprocesseur et de son interfaçage avec le bus système du module de traitement, sont les suivants :

- s'adapter à un bus de données de largeur 32 bits alors que la majorité des données et résultats manipulés par le coprocesseur sont codés sur 8 ou 16 bits;
- s'interfacer avec un bus mémoire disponible de manière asynchrone et devoir suspendre temporairement le fonctionnement du coprocesseur pour permettre l'accès à la mémoire par l'unité centrale plus prioritaire;
- calculer les adresses de lecture ou d'écriture en mémoire des données à traiter et des résultats et piloter le bus système du module dans le cas où le coprocesseur est actif;
- lire ou écrire dans une mémoire unique, disposant d'un seul port d'accès, des données à traiter et des résultats de calcul qui sont présents simultanément sur la plupart des dispositifs parallèles à propagation d'informations.

A notre avis, le dernier problème cité sera, sans aucun doute, le plus complexe à résoudre.

Cette méthode d'interfaçage est beaucoup plus sophistiquée que celles qui sont actuellement utilisées dans les systèmes de traitement d'images comportant des unités pipe-line ou systoliques où, des mémoires particulières stockent les images (mémoires vidéo VRAM à accès aléatoire et séquentiel) et transférent les informations vers ces unités via des bus spécifiques.

Le dispositif d'interfaçage du coprocesseur de traitement d'images est tout aussi important que la structure parallèle interne de ce dernier, car il détermine la vitesse de transfert des données et

des résultats dans la mémoire locale et, de fait, la cadence maximale de calcul.

Heureusement l'évolution actuelle des mémoires dynamiques de grande capacité et l'organisation parallèle (32 bits) de notre bus de données nous permettent d'une part d'espérer des débits mémoire avoisinant ou dépassant les 30 mégaoctets/seconde qui correspondent à des échanges de données à des cadences bien supérieures aux classiques cadences vidéo et d'autre part, de pouvoir, en un seul cycle de lecture ou d'écriture, transférer entre le coprocesseur et la mémoire, plusieurs informations (données à traiter ou résultats) simultanément.

2.3) Spécifications des tâches réalisées par le coprocesseur

2.3.1) Nature des tâches

Le coprocesseur pour des raisons économiques (minimisation de la complexité), doit réaliser exclusivement les tâches pour lesquelles l'unité centrale du module n'a pas des vitesses de traitements suffisantes. Actuellement, ces problèmes se situent principalement au niveau des tâches de traitement d'images bas et moyen niveau (cf. 1.1.2) pour lesquelles il est nécessaire de prendre en compte de gros volumes de données (la ou les images à traiter).

La parallélisation de ce type de tâches a déja été abondamment étudiée (cf. 1.2.3.2.1) et il existe, pour chacun des algorithmes de traitement d'images les plus connus, une ou plusieurs architectures spécialisées capables de les réaliser à des cadences suffisantes pour la plupart des applications de vision industrielle. Malheureusement, la majorité de ces systèmes spécifiques ne sont pas du tout polyvalents et, leurs performances se dégradent énormément, dès que l'utilisateur les reconfigure pour réaliser d'autres tâches de traitement d'images que celles pour lesquelles ils ont été initialement conçus.

Les dispositifs de traitement d'images intégrés représentent une faible part du marché des composants microélectroniques et leur extrême spécialisation réduit d'autant plus le nombre d'applications qui peuvent leur être dévolues. Cette situation est incompatible avec la réduction des coûts et l'orientation de la microélectronique vers la production de masse.

Si l'objectif est d'intégrer ce style de dispositifs, il faut privilégier les fonctionnalités et la diversité d'utilisation du circuit au lieu de tendre vers la recherche de performances optimales qui déjà, commencent à dépasser les besoins (PLE86, TRW87, RUE88). Notre problème est donc, de privilégier les fonctionnalités, plus que la performance théorique crête, qui pourra toujours

être atteinte, au niveau de notre architecture globale de système, en parallélisant des modules de traitement, chacun d'entre eux travaillant sur une partie de l'image (parallélisme à structure de données).

A partir d'un tour d'horizon sur les méthodes de traitement d'images utilisées sur des systèmes industriels déjà en activité, nous pouvons définir un ensemble d'algorithmes de traitement que notre coprocesseur doit impérativement réaliser pour qu'il puisse être utilisé dans de nombreuses applications de vision industrielle. Ces algorithmes doivent pouvoir s'appliquer sur des images complètes, des sous-images de forme rectangulaire ou, dans le cas le plus général, sur des zones d'images de formes quelconques.

- ***tâches de bas niveau (images -> images)***

  filtrages linéaires (convolutions)
  morphologie mathématique binaire
  morphologie mathématique en gris (transformée en chapeau plat)
  addition, soustraction, minimum ou maximum de deux images pixel à pixel
  opérations logiques entre deux images
  opérations arithmétiques entre une image et une constante
  seuillage d'une image par une constante
  transformation d'une image pixel à pixel par une loi tabulée (LUT)
  transformation géométrique de l'image (rotation, rééchantillonnage)
  marquage de certains pixels répondant à une condition

- ***tâches de moyen niveau***

  histogrammes
  projections dans une direction quelconque
  calculs de moments
  codage d'une image binaire sous la forme d'une suite de segments
  extraction des indices ligne et colonne de certains pixels répondant à une condition

Le principal intérêt des systèmes parallèles à propagation d'informations est de permettre la parallélisation d'une tâche de calcul, mais aussi d'enchaîner, dans une même phase de traitement, des tâches appliquées successivement sans passer par une mémorisation intermédiaire des informations (images).

Notre coprocesseur doit donc être capable d'enchaîner dans une même phase de traitement, plusieurs des tâches citées ci-dessus. En traitement d'images, seul un petit nombre

d'enchaînements a un intérêt, on peut, sans trop de limitation, se contenter du type d'enchaînement classique suivant (cf. figure 2.3) :

- d'une loi particulière d'accès aux données (rotation, rééchantillonnage)
- d'une ou plusieurs tâches de bas niveau générant une image résultat à partir d'une ou deux images d'entrée
- d'une ou plusieurs tâches de moyen niveau (extraction d'informations sur l'image résultat)
- d'une loi particulière de rangement dans la mémoire locale, des résultats issus des tâches de bas ou moyen niveau, sous forme de tableaux (images) ou de listes.

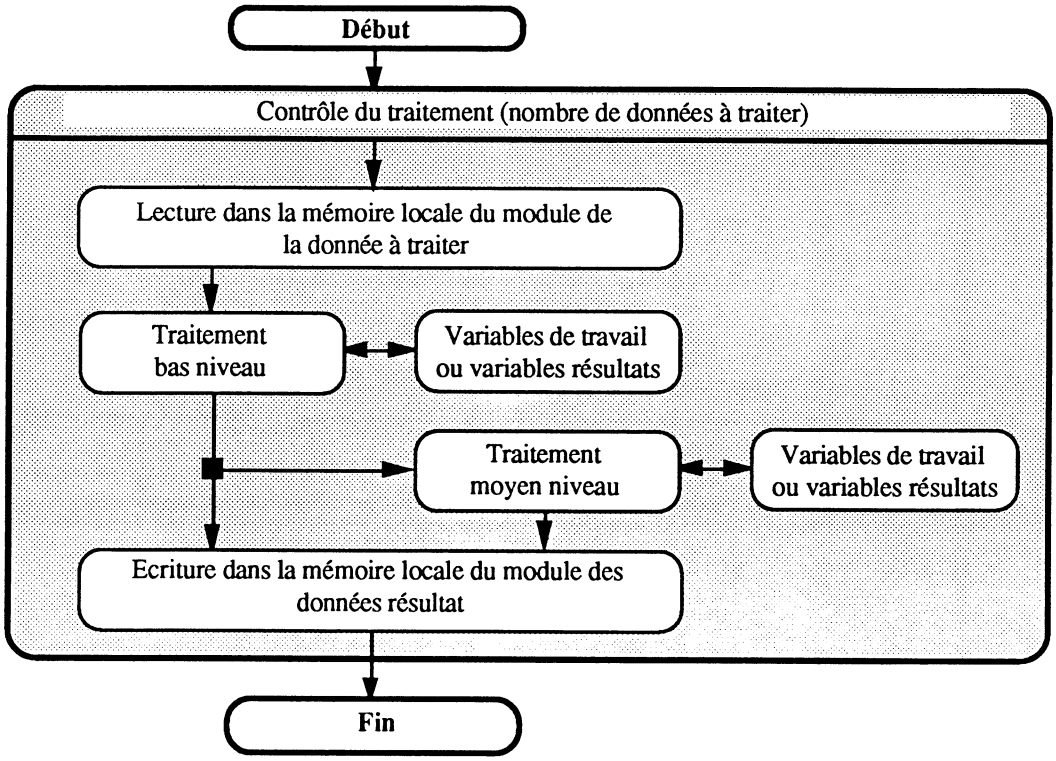


figure 2.3 : Structure générale d'une phase de traitement standard

2.3.2) Structure générale des algorithmes cibles

Les algorithmes de traitement que nous avons définis pour notre coprocesseur peuvent, quel que soit leur nature et la manière dont ils sont enchaînés, être représentés par trois processus de calcul échangeant entre eux des informations sous la direction d'un processus de contrôle (figure 2.4).

Le premier processus lit séquentiellement dans la mémoire locale tous les opérandes nécessaires pour réaliser le traitement.

Le second processus traite ces groupes d'opérandes et génère un groupe de résultats en sortie.

Le troisième processus écrit séquentiellement les résultats des calculs dans la mémoire locale du module.

Le processus de contrôle gére le nombre d'itérations (d'exécution) des trois premiers (nombre de données à traiter et/ou nombre de résultats à générer)

Ces trois premiers processus correspondent à ce que nous avons appelé les "noyaux de calcul" (cf 2.1), ils sont exécutés pour chacune des informations élémentaires (pixels) à traiter.

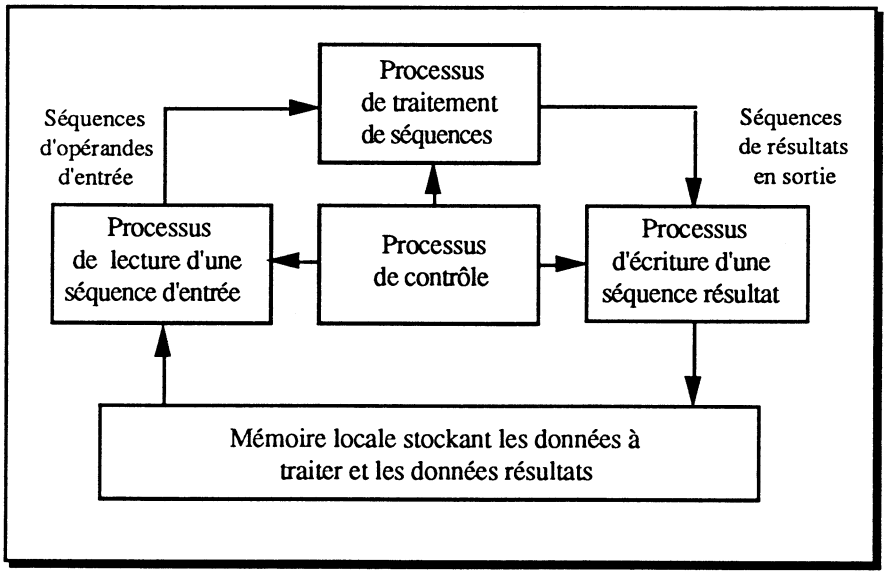


figure 2.4 :
Structure générale des tâches de calcul réalisées par le coprocesseur

Chacun de ces trois processus connaît la loi régissant l'ordre des séquences d'opérandes d'entrées et de résultats en sorties : que ce soit a-priori ou à partir d'informations spécifiques intégrées dans ces séquences. Dans certains cas le processus de traitement de séquences peut être totalement indépendant de l'ordre d'arrivée des données à traiter (pixels).

Si l'on détaille les fonctionnalités et les caractéristiques des algorithmes de traitement d'images bas et moyen niveau sous la forme de processus traitant des séquences, deux cas possibles peuvent se présenter :

Algorithmes de traitement de bas niveau :

la structure générale des tâches de traitement d'images de bas niveau représentée sous la forme de processus traitant des séquences peuvent être décrits de la manière suivante (cf. figure 2.5) :

- ces algorithmes de calcul transforment une séquence d'entrée de données à traiter en une séquence de données résultat;
- ils nécessitent pour leurs calculs des variables locales temporaires, en nombre fini (en général quelques lignes image au maximum);
- le temps d'exécution pour traiter chaque élément de la séquence d'entrée est borné (le plus souvent un cycle de l'horloge de calcul dans les réalisations actuelles);
- les séquences d'entrée et de sortie sont composées d'informations disponibles séquentiellement suivant un ordre prédéfini qui ne sera pas modifié au cours de l'exécution;
- les séquences d'entrée et les séquences de sortie ont quasiment toujours la même longueur;
- la séquence d'entrée peut, dans certains cas particuliers (ex. : images issues d'une caméra linéaire inspectant des objets sur un tapis roulant), avoir une longueur non bornée.

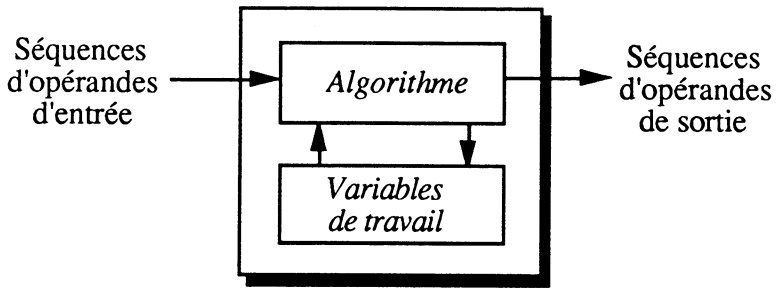


figure 2.5 :
Structure générale des algorithmes réalisant des tâches de bas niveau

Exemple :

Une opération de convolution sur une image appliquée sur un voisinage de trois points suivant les lignes peut être par exemple représenté sous la forme d'un processus traitant des séquences de pixels en entrée disponibles suivant un ordre de balayage ligne à ligne et émettant les résultats en sortie suivant un ordre identique. Ce processus nécessite pour ses calculs trois variables de travail pour stocker temporairement les trois pixels en cours du voisinage sur lequel sont appliqués les calculs (cf. figure 2.6).

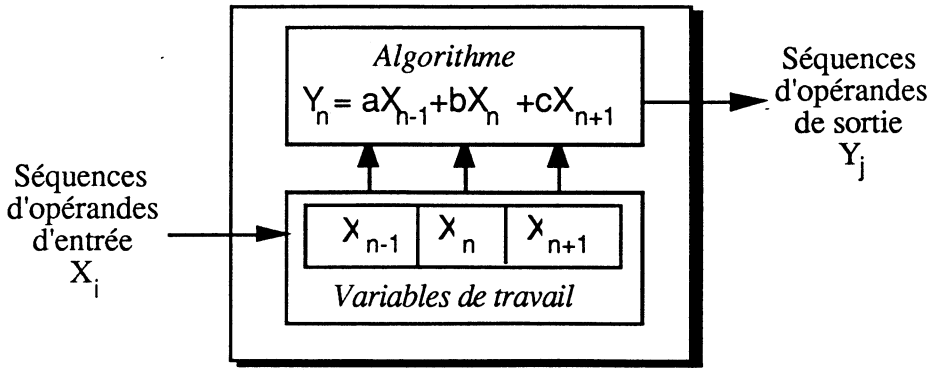


figure 2.6 :
Exemple de processus de bas niveau traitant des séquences

Algorithmes de traitement de moyen niveau :

La structure générale des tâches de traitement d'images de moyen niveau représentée sous la forme de processus traitant des séquences peuvent être décrits de la manière suivante (cf. figure 2.7) :

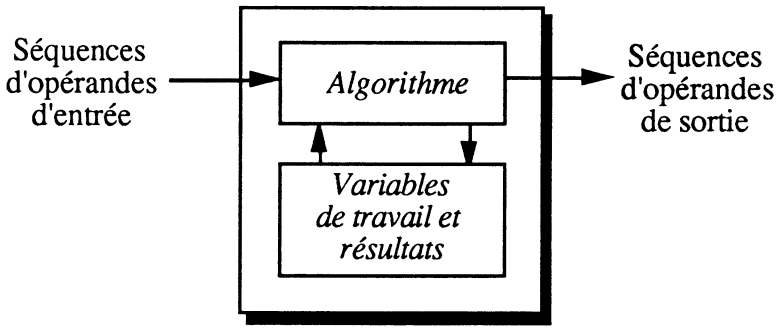


figure 2.7 :
Structure générale des algorithmes réalisant des tâches de moyen niveau

- ces algorithmes de calcul transforment une séquence d'entrée de données à traiter en une séquence de données résultat ou, génèrent une mesure unique à partir de toutes les informations de la séquence d'entrée;
- ils nécessitent pour leurs variables locales et leurs variables résultat, une zone de mémorisation temporaire de taille finie (elle dépend de la dynamique de représentation des données et/ou de la taille de l'image : nombre de pixels par ligne ou nombre de lignes);
- le temps d'exécution pour traiter chaque élément de la séquence d'entrée est borné (le plus souvent un cycle de l'horloge de calcul dans les dispositifs spécialisés de calcul parallèles actuellement réalisés);

- les séquences d'entrée et de sortie sont composées de pixels ou d'informations géométriques (numéros de ligne et/ou de colonne) dans un ordre prédéfini qui ne sera pas modifié au cours de l'exécution;
- les séquences de sortie peuvent avoir une longueur inférieure à la séquence d'entrée;
- la séquence d'entrée peut, dans certains cas, avoir une longueur non bornée.

Exemple :

Une opération de calcul de l'histogramme sur les valeurs d'un ensemble de pixels peut être représenté sous la forme d'un processus traitant une séquence de pixels disponibles en entrée suivant un ordre quelconque dont la mémoire de travail contient un tableau, initialisé préalablement à zéro, qui est utilisé pour le calcul de l'histogramme (cf. figure 2.8) . Lorsque tous les pixels auront été traités, le résultat du calcul (histogramme) sera disponible dans la mémoire de travail.

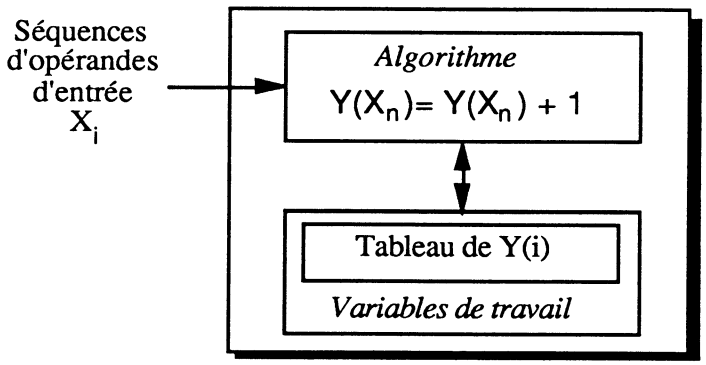


figure 2.8 :
Exemple de processus de moyen niveau traitant des séquences

2.3.3) Structures et dynamiques de représentation des données

Les différentes types d'informations qui sont manipulées par les algorithmes de traitement d'images présentés brièvement dans le paragraphe précédent sont les suivants :

- des images ou des sous-ensembles d'images :
  Ce sont des tableaux de pixels où la valeur de chaque pixel est représentée par un entier 8 ou 16 bits signé ou non. Ces informations sont des structures de données statiques, leur taille est connue à priori et ne dépend pas de résultats de calcul.

- des listes d'informations géométriques :

  Ce sont des vecteurs ou des tableaux stockant la position géométrique dans l'image (numéro de ligne, numéro de colonne codés sur 16 bits) de pixels ayant satisfait à une condition particulière (exemple : pixels appartenant aux contours). Ces informations sont des structures de données dynamiques, leur taille n'est pas connue à priori et dépend des résultats des calculs, cependant, il est possible de borner leur taille par le nombre de pixels de l'image de départ.

- des mesures (primitives)

  Les résultats (caractéristiques ou mesures sur l'image) de la plupart des opérations de moyen niveau sont organisés sous la forme de vecteurs dont chaque élément est un entier codé sur 32 bits (exemple : histogramme). Ces informations sont des structures de données statiques, leurs tailles peuvent être variables, mais sont toujours connues à priori, elles dépendent uniquement de la taille de l'image à traiter ou/et de la dynamique de représentation des pixels qui composent cette dernière.

Les deux premiers types d'informations, du fait de leurs tailles et de leurs modes de calcul, pourront être stockés dans la mémoire locale du module de traitement.

Les mesures, du fait de leurs méthodes de calcul (les variables de travail sont à la fin du calcul les variables résultat), seront uniquement stockées dans une mémoire de travail particulière intégrée au niveau de l'accélérateur de calcul, que l'unité centrale ira lire, à la fin de la phase de traitement.

On peut résumer, sous la forme d'un tableau (cf. figure 2.9), les différents types d'informations à traiter par le coprocesseur et les résultats qui en sont issus ainsi que les algorithmes qui les manipulent.

2.3.4) Informations nécessaires pour réaliser les traitements

Les processus de traitement de séquences des algorithmes cités dans le tableau ci-dessus doivent, pour chaque étape élémentaire de calcul (le traitement d'un pixel), disposer simultanément une ou plusieurs des informations suivantes :

- valeurs du ou des deux pixels des images de départ;
- position du pixel à traiter dans l'image de départ ou position du pixel résultat dans l'image d'arrivée (numéro de ligne, numéro de colonne);
- indicateur déterminant si le pixel répond ou non à une condition particulière.

Informations à traiter	Algorithmes de traitement	Informations résultats
une image ou une zone d'image	**Algorithme de bas niveau** filtrage linéaire filtrage morphologique seuillage transformation par une loi tabulée Opération avec une constante rotation, rééchantillonage	une image ou une zone d'image
deux images ou deux zones de deux images	**Algorithme de bas niveau** Opération entre deux images	une image ou une zone d'image
une ou deux images ou une ou deux zones d'image	**Algorithme de bas niveau** détections de conditions	une image ou une zone d'image où chaque pixel est doté d'un indicateur de conditions
une ou deux images ou une ou deux zones d'image	**Algorithme de moyen niveau (mesures statiques)** histogrammes projections mesures géométriques (moments)	une ou plusieurs mesures sous forme de vecteurs
une image ou une zone d'image où chaque pixel est doté d'un indicateur de conditions	**Algorithme de moyen niveau (mesures dynamiques)** Codage d'image binaire Extraction de liste d'indices	une liste d'informations

figure 2.9 :
Tableau récapitulatif des principaux algorithmes de traiteme d'images utilisés dans des applications industrielles et de structures de données qu'ils manipulent

Il n'est pas techniquement possible, de transmettre simultanément toutes les informations cités ci dessus, dans une architecture parallèle à propagation d'informations, car cela conduirait à utiliser des chemins de données de très grande largeur pour leur transmission en pipe-line à l'intérieur et à l'extérieur des opérateurs.

La plupart des algorithmes de traitement que nous désirons réaliser, traitent séquentiellement les pixels de l'image et les calculs sont ou peuvent être organisés pour que, dans la majorité des cas, le prochain pixel à traiter soit l'un des voisins du pixel en cours de traitement.

Du fait de cette notion de proximité, il est possible de réduire le nombre d'informations nécessaires pour transmettre à tout instant la position du pixel en utilisant des informations codées (par exemple codage de Freeman BAT85). Ce type de méthode est très efficace et a déjà été utilisé dans certains systèmes de traitement d'images Hitachi (HIT88), où il est transmis en pipe-line avec chaque donnée (pixel), un champ de contrôle décrivant l'état d'avancement des calculs qui permet, si nécessaire, à chaque opérateur de recalculer en interne les indices ligne et colonne du pixel (pixel à pixel).

2.3.5) Séquences d'accès aux données

La mémoire locale stockant l'image est à accès aléatoire et de fait, permet de définir librement l'ordre dans lequel les pixels seront traités. Cette possibilité de traiter les informations suivant des ordres complexes apporte une grande souplesse et une grande simplicité au niveau de la mise en oeuvre "pipe-line" de certains algorithmes de traitement d'images (MOL88, TAK88, HIT88).

Les mécanismes de lecture ou d'écriture des données image actuellement utilisés sont les suivants (cf. figure 2.10) :

- balayage vidéo entrelacé (HIT88);
- balayage vidéo désentrelacé dit aussi progressif;
- balayage vidéo horizontaux ou verticaux pour réaliser des traitements séparables;
- balayage dit "Boustrophedon" pour l'étiquettage et l'extraction d'informations sur les composantes connexes (DAV80);
- balayage désentrelacé ligne à ligne sur des régions de forme quelconque (WEI87, TAK88).

Pour les algorithmes réalisant des opérations sur l'image à caractère géométrique (rotations, corrections de déformations géométriques,...), les lois de lecture des informations à traiter

peuvent être très complexes, mais les lois de réécriture sont toujours des balayages vidéo désentrelacés (lois de transformations inverses STR85).

Pour les listes d'indices, les informations sont toujours stockées séquentiellement dans la mémoire locale (deux données consécutives sont stockées à deux adresses consécutives).

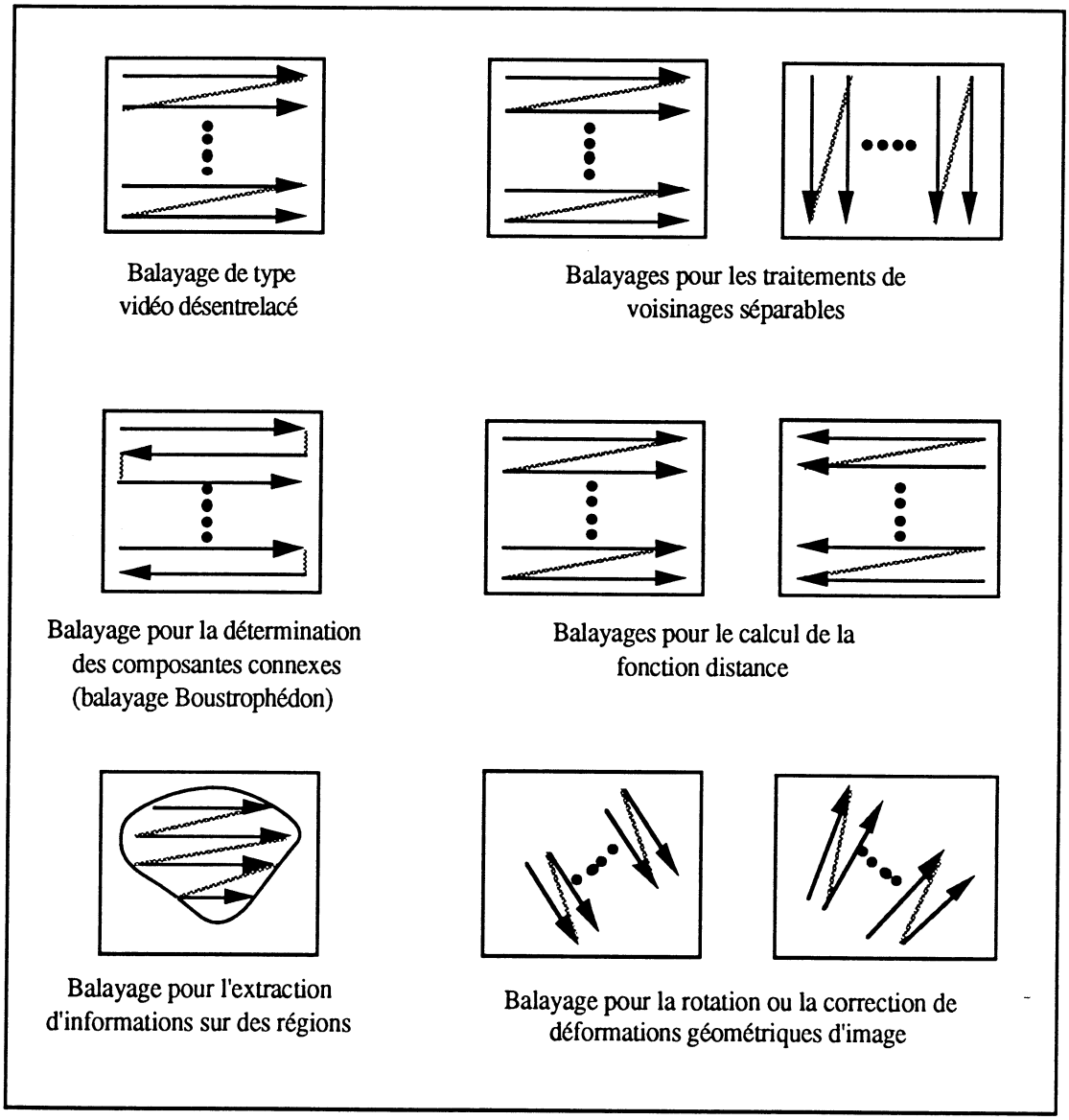


figure 2.10 : Mécanismes de balayage les plus courants

2.4) Choix pour l'architecture du coprocesseur

La conception d'un coprocesseur de traitement d'images capable de réaliser toutes les fonctionnalités décrites dans le paragraphe précédent est dans l'idée un concept très séduisant, mais sa conception apparait comme une tâche plus qu'ardue. Il est impératif de réduire la complexité de l'étude en décomposant ce coprocesseur en sous-systèmes qui pourront être étudiés séparément. Dans un premier temps, nous allons donc nous pencher sur l'architecture générale du coprocesseur et essayer de le subdiviser en dispositifs bien distincts que nous détaillerons dans la suite de cette étude.

2.4.1) Architecture générale

2.4.1.1) Les choix possibles

Nous avons décrit dans le paragraphe 2.2.5. les deux méthodes utilisables pour interfacer un coprocesseur avec le bus système du module (coprocesseur actif, coprocesseur passif). Ce mécanisme d'interfaçage d'une unité de traitement spécialisée est indépendant de l'architecture parallèle interne de celle ci.

Pour obtenir le meilleur compromis (performances/volume d'électronique), nous nous sommes orientés, pour le coprocesseur de traitement d'images, dès le début de sa conception, vers des architectures pipe-lines ou systoliques. Dans ce style de systèmes parallèles :

- les pixels sont traités séquentiellement suivant des lois qui sont définies à priori pour toute la durée de la tâche de traitement;
- les résultats sont souvent disponibles séquentiellement suivant un ordre identique à celui des informations d'entrée;
- les données à traiter et les résultats ne sont lus et écrits qu'une seule fois dans la mémoire locale.

Ces particularités ont incité, pour des raisons de simplicité, la plupart des concepteurs d'accélérateur de calcul parallèle de type pipe-line, à en décomposer l'architecture en deux dispositifs très indépendants (YON88) :

- un dispositif lisant ou écrivant les données en mémoire;
- un dispositif de traitement pipe-line proprement dit.

De telles méthodes dichotomiques appliquées à l'architecture de sous systèmes de traitement spécialisés ont déjà été étudiées pour des systèmes de traitement du signal ou de synthèse et traitement d'images (ANA87, FUJ88, LUQ87, OKL88). Il est possible de subdiviser en trois types d'approches les différentes études réalisées dans ce domaine.

Approche coprocesseur passif :

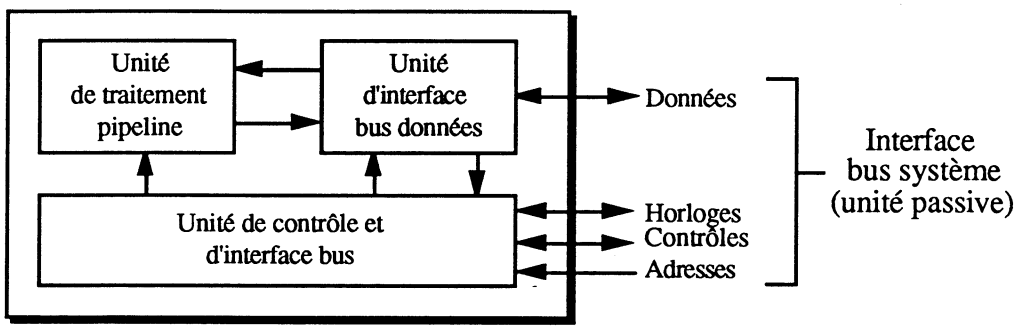


figure 2.11 : Architecture de coprocesseurs passifs

Le coprocesseur est vu sur le bus système comme un ensemble de ports d'entrées-sorties qui sont utilisés pour sa programmation et ses échanges de données. L'unité centrale réalise les transferts d'informations entre le coprocesseur et la mémoire contenant l'image. Les performances de ce type de structure sont souvent limitées par les possibilités de transfert de l'unité centrale. L'unité de contrôle est simple et l'exécution des calculs est séquencée par la disponibilité des données ("data driven processing"). Cette architecture a déjà été utilisée pour des applications de traitement d'images binaires (JEN87).

Approche coprocesseur actif :

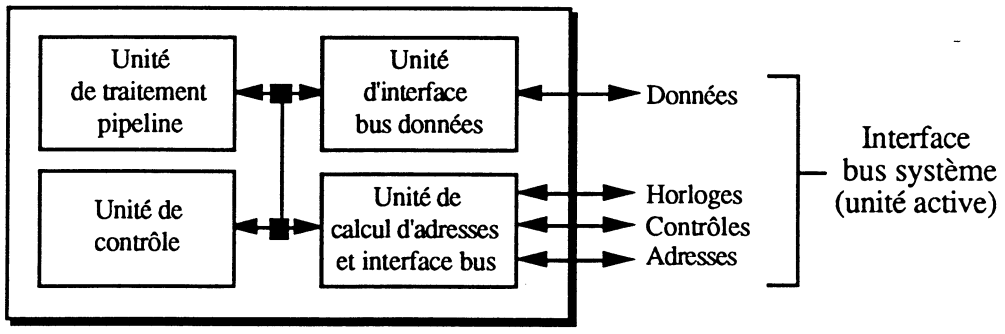


figure 2.12 : Architecture de coprocesseurs actifs

Le coprocesseur est vu sur le bus système comme une unité active, il est capable d'accéder à la mémoire locale pour ses échanges de données. Le microprocesseur central réalise uniquement la configuration de cette unité. Une unité de contrôle centralisée gère le fonctionnement simultané de l'unité de calcul parallèle pipe-line, de l'unité de calcul d'adresses et de l'interface bus. Cette architecture est très proche de celle des microprocesseurs de traitement du signal (DSP). L'architecture de ce type de coprocesseurs est orientée pour la réalisation de ceux-ci sous la forme d'un seul circuit intégré ("monochip") (ZOR87, LUQ87) et leurs performances (nombre d'unités de traitement) dépendent des possibilités de la technologie microélectronique employée.

Approche coprocesseur passif couplé à une unité active :

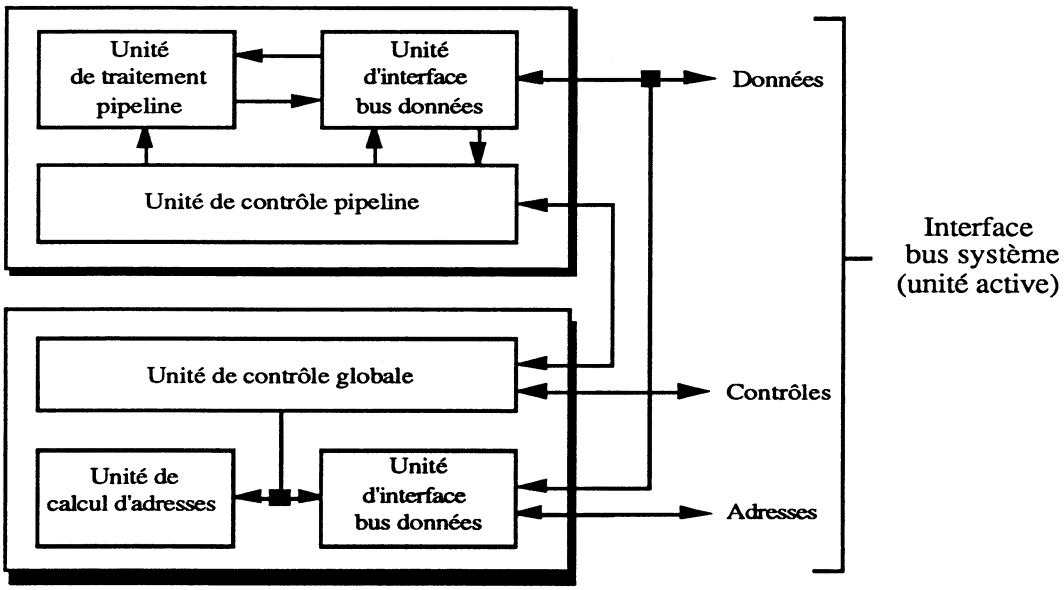


figure 2.13 : Architecture de coprocesseur passif couplé à une unité active

Le coprocesseur est vu sur le bus système comme une unité active, cependant il est décomposé en deux modules : une unité parallèle passive et une unité active capable de réaliser des échanges d'informations avec la mémoire. Chacunes de ces deux unités disposent d'une unité de contrôle. L'unité active gère le contrôle du séquencement global (nombre de données à traiter), l'unité de traitement passive est esclave et est séquencée par la disponibilité des données ("data driven processing"). Les architectures actuellement les plus performantes sont basées sur ce principe (ANA87, FUJ88, LUQ87, OKL88).

Nous avons décidé d'adopter cette dernière approche pour l'architecture de notre coprocesseur, car elle permet de dissocier son étude et sa réalisation en deux parties et elle offre le maximum de latitude pour la définition du dispositif de traitement parallèle pipe-line.

2.4.1.2) Architecture adoptée

Pour disposer d'un maximum de latitude au niveau de la conception de nôtre dispositif de calcul parallèle, nous avons donc décidé de scinder l'architecture de notre coprocesseur en deux unités indépendantes : un "processeur d'interface mémoire" et une "unité parallèle de traitement à propagation d'informations" composée d'un nombre variable de dispositifs de calcul, ces deux unités échangeant entre elles des informations (données + contrôles), par exemple par des files d'attente (LUQ87, OKL88, ROU86). Ce mécanisme de dialogue par files d'attente et de cadencement par disponibilité des données peut s'apparenter aux systèmes parallèles de type "data flow processing" (FUJ88) ou "wavefront processing" (KUN84)

La méthode que nous avons décidé d'adopter est très proche de celle qui est utilisée dans le système TIP3 (FUJ88), que nous considérons comme une référence, pour les systèmes parallèles à propagations d'informations, au niveau de la modularité et de la rigueur des échanges à l'intérieur de la structure (cf. figure 2.14). Tous les échanges d'informations entre les différentes unités qui composent TIP3 ont un format et un protocole identiques, les opérateurs échangent entre eux des jetons ("token") qui sont composés, d'une donnée (donnée à traiter ou résultat) et d'un identificateur (champ de contrôle) permettant de gérer le bon déroulement des calculs.

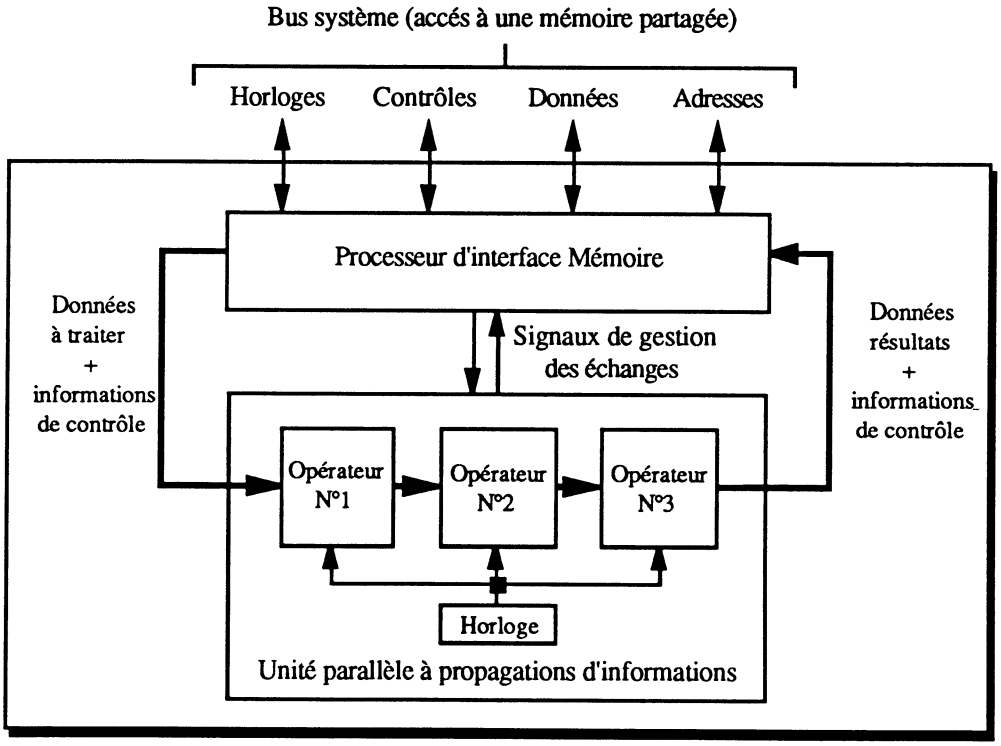


figure 2.14 : Architecture de TIP3

Nous ne nous pencherons pas trop en détail sur les "processeurs d'interface mémoire", car nous avons bon espoir que ceux-ci puissent être réalisés à plus ou moins de frais, avec des circuits du commerce (contrôleurs de DMA programmables, processeurs graphiques) (WSI88, DIE88) ou que les futurs transputers (Inmos série T9000) ou, DSP Texas Instrument TMS320C40 soient capables, via une programmation appropriée, de réaliser les tâches de transferts d'informations entre la mémoire locale du module et l'unité de traitement parallèle passive par l'intermédiaire de leur liens série ou parallèle rapides. L'essentiel de l'étude sur l'architecture du coprocesseur sera donc focalisé sur l'unité de calcul parallèle.

Quelle que soit la nature et le mode de réalisation de l'unité d'interface mémoire, nous avons choisi d'interfacer notre unité de traitement pipe-line de la manière suivante (cf. figure 2.15) :

- sa configuration seront réalisées via des entrées/sorties parallèle 8 bits;
- ses échanges de séquences de données à traiter et de séquences de résultats avec le processeur d'interface mémoire seront réalisés via des files d'attente (FIFO);
- un ensemble de signaux échangés entre le processeur d'interface mémoire et l'unité de calcul parallèle permettront de synchroniser les échanges d'information et d'initialiser cette dernière.

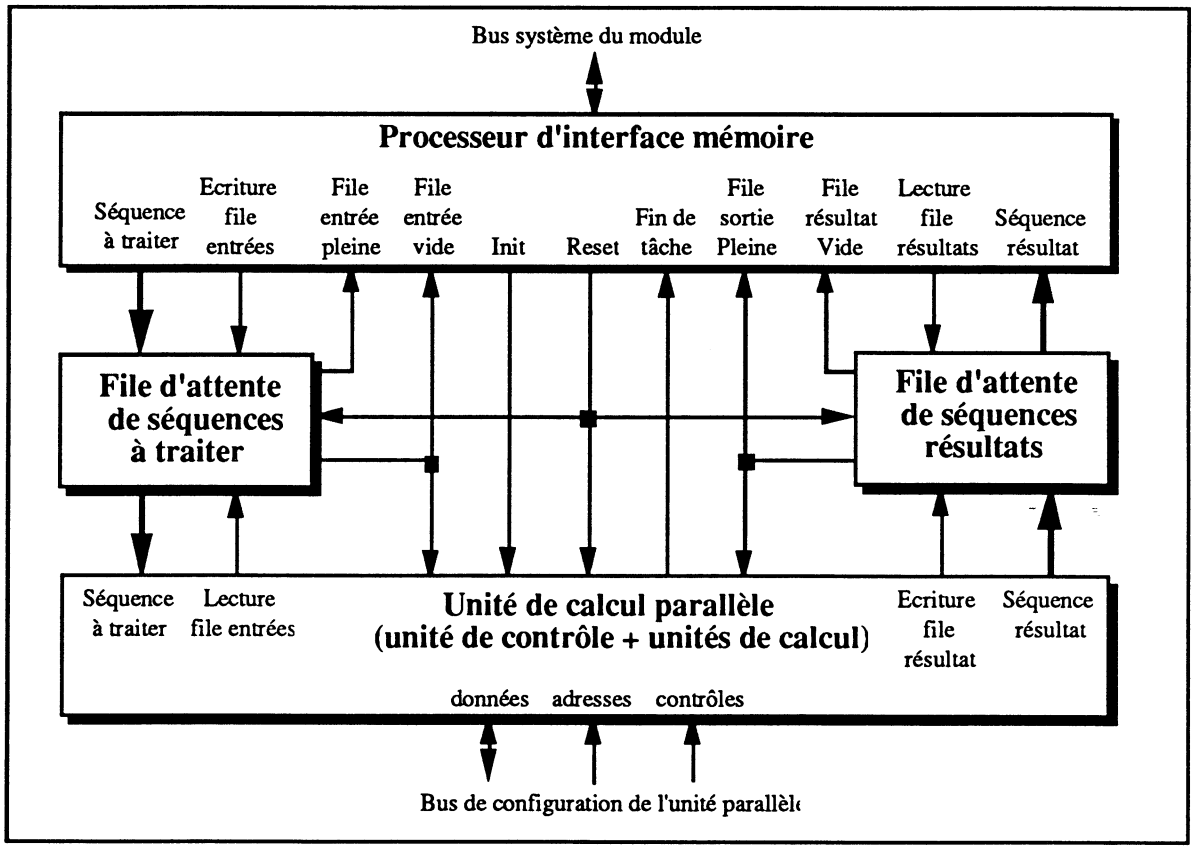


figure 2.15 : Architecture générale du coprocesseur

Tous les calculs réalisés par l'unité de traitement parallèle seront synchrones, cadencés par une horloge de calcul unique. Pour chaque cycle de l'horloge de calcul, le dispositif traitera un pixel. Le déroulement de ses calculs sera conditionné par le taux de remplissage des files d'attente (son horloge de calcul sera suspendue lorsque la file d'entrée sera vide et/ou la file de sortie sera pleine). Les échanges d'informations entre les files d'attente et la mémoire locale seront à la charge du processeur d'interface mémoire, la réalisation de ces tâches sera ici aussi liée aux taux de remplissage des files.

Le mécanisme de dialogue, que nous avons décidé d'adopter pour les échanges entre les deux unités du coprocesseur a, du point de vue réalisation, l'intérêt d'être simple (les files d'attente "monochip" sont disponibles sur le marché des composants électroniques) et du point de vue architectural, l'avantage de pouvoir totalement dissocier, tant sur le plan du séquencement (horloges) que sur celui du fonctionnement, ces deux unités. Dans ces conditions les vitesses de fonctionnement de ces deux unités peuvent être maximisées.

L'utilisation de files d'attente permet de rendre séquentielle la lecture et l'écriture, dans la mémoire locale, des informations disponibles en parallèle sur les entrées et les sorties de l'unité de calcul parallèle (cf. figure 2.16). Les fonctionnements des deux unités du coprocesseur peuvent être totalement concurrents et il est possible dans ces conditions d'utiliser des cycles de transferts multiples ("burst transfer") sur le bus système pour le remplissage ou le vidage des files d'attente.

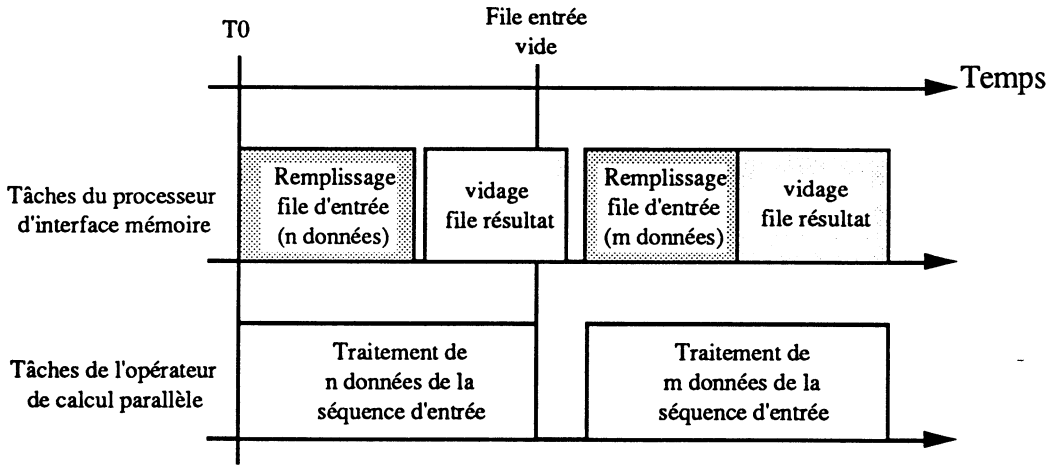


figure 2.16 : Fonctionnement simultané des deux unités du coprocesseur

Dans la suite de ce mémoire, nous n'aborderons plus le processeur d'interface mémoire, la conception d'une version intégrée ou d'une simulation logicielle de celui-ci n'était pas un travail réalisable dans le temps fini alloué à ces travaux de recherche.

2.4.2) Architecture de l'unité de calcul parallèle

L'unité de calcul parallèle doit être modulaire et doit pouvoir intégrer un nombre variable d'unités de calcul suivant le type et le mode de réalisation microélectronique de celles-ci. Dans une première approche (cf paragraphe 2.3), nous avons décidé que celle ci pourrait réaliser des séquences de traitement comprennant au maximum :

Des tâches de bas niveau :
- opération diadique pixel à pixel entre une image et une constante;
- opération diadique pixel à pixel entre deux images;
- opération de voisinage (convolution, morphologie);
- opération de marquage.

Des tâches de moyen niveau :
- opérations d'extraction d'informations;
- opération d'extraction de listes.

Pour réaliser toutes les tâches qui lui sont allouées (calculs et dialogues), l'opérateur de calcul parallèle nécessite pour chacune de ces phases élémentaires de calcul (traitement d'un pixel) un ensemble d'informations (données et contrôles) disponibles simultanément. Le type, la nature et le protocole de transmission de ces informations, c'est-à-dire l'interface externe de cet opérateur parallèle, sont les premiers points à étudier.

2.4.2.1) Interface externe

Nous ne savons pas à priori le nombre d'unités de calcul que comportera notre unité de calcul parallèle, dans ces conditions il n'est pas souhaitable qu'il y ait des communications d'informations (données) globales à l'intérieur sa structure. Nous avons donc décidé que, comme dans l'architecture de TIP3, toutes les informations exceptées les signaux de séquencement des calculs se propageront localement dans la structure d'unité de calcul en unité de calcul. Ce type de transmission des informations de contrôle est utilisée de manière quasi systématique dans la plupart des systèmes pipe-line reconfigurables très complexes (DIE88, CUB85) ou comportant un grand nombre d'étages de calcul (HIT88).

Après des études, que nous espéront exhaustives, sur les informations nécessaires pour réaliser au moyen de systèmes pipe-line, les algorithmes de calculs précédemments cités, nous avons décidés que l'unité de calcul parallèle échangerait avec l'extérieur sur ses entrées et sorties, des séquences de format identique, composées : de données, de contrôles (cf figure 2.17)

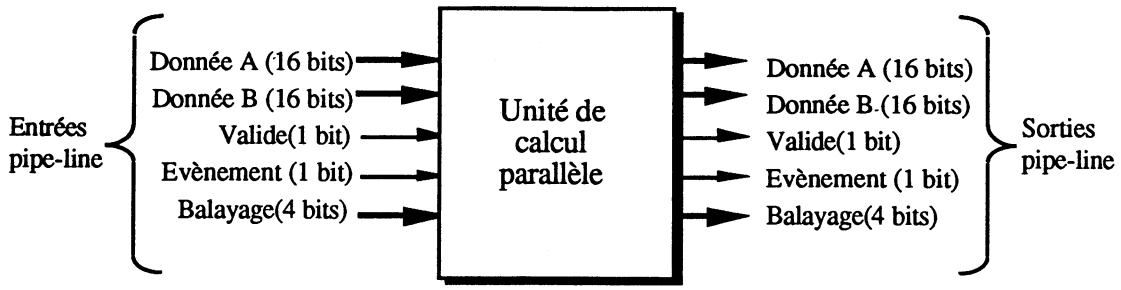


Figure 2.17 : Informations pipeline échangées par l'unité de calcul parallèle

Ces différentes informations (données + contrôle) sont transmises simultanément pour chaque cycle de calcul (traitement d'un pixel) sur les entrées et sorties de l'unité de calcul parallèle. Les significations et fonctions de ces différentes informations sont les suivantes :

Informations de type données

Les deux chemins de données A et B permettent de transmettre simultanément deux entiers 16 bits codés en complément à 2 qui peuvent représenter : deux pixels issus de deux images différentes, la position d'un pixel dans l'image (indices ligne et colonne). Le chemin de donnée A est dédié aux calculs, le chemin de donnée B peut être utilisé pour transmettre des données ou des constantes d'initialisation du réseau de calcul (N° de région, position du premier pixel à traiter, ...).

Informations de type contrôle

L'objectifs des chemins de contrôle est de transmettre des informations permettant de modifier dynamiquement lors du traitement le fonctionnement de certains des dispositifs composants l'unité de calcul parallèle. On peut considérer que ce groupe d'informations agit comme des "instructions" pour gérer le bon déroulement des calculs. Ces informations de contrôle sont de trois types :

- L'information de contrôle "**Valide**" (codée sur un bit, active à 1) permet de sélectionner dans l'ensemble des pixels à traiter de la séquence, ceux qui seront prendre en compte pour les opérations d'extraction de caractéristiques (algorithmes de moyen niveau).

- l'information de contrôle "**Evénement**" (codée sur un bit, active à 1) permet d'identifier certains pixels particuliers (par une opération de marquage) cette information est modifiée par l'unité de calcul parallèle au cours des traitements.

- l'information de contrôle "**Balayage**" (codée sur 4 bits) a deux rôles :

  - elle transmet lors du traitement des indications permettant à chacunes des unités qui composent l'unité de calcul parallèle de recalculer si nécessaire la position dans l'image du pixel en cours de traitement, ceci quelquesoit l'ordre dans lequel les informations sont traitées;
  - elle a aussi pour rôle de transmettre toutes les informations de contrôle qui sont nécessaires pour le traitement d'une séquence par l'accélérateur de calcul parallèle (initialisation, début, réinitialisation, fin). Cette information est codée sur 4 bits, elle n'est jamais modifiée lors du calcul et sa signification et ses actions sont résumée dans le paragraphe suivant.

2.4.2.2) Principe de fonctionnement

Lorsque nous avons défini les tâches de traitement de base devant être réalisées par notre accélérateur de traitement d'images, nous avons étudié tout particulièrement l'ordre dans lequel les pixels étaient traités et la nature des zones d'images sur lesquelles doivent être appliqués les traitements (cf. paragraphe 2.3.5). Nous considérons comme indispensable la possibilité de traiter les pixels suivant un ordre autre que le classique balayage vidéo (ligne à ligne).

Pour pouvoir traiter les données suivant les différentes séquences (ordre) que nous avons préalablement cités, l'unité de calcul parallèle doit pouvoir recalculer à tout instant la position dans l'image du pixel en cours de traitement et, doit pouvoir également se réinitialiser à la volée lors de balayages complexes pour ne pas interrompre les calculs en cours (vidage du pipe-line).

Pour satisfaire à ces contraintes, nous avons décidé que l'unité de calcul parallèle serait en partie ou totalement "reconfigurable" durant certaines phases de traitement. L'information de contrôle "Balayage" doit entre autre permettre de réaliser cette reconfiguration (figure 2.18).

Suivant les valeurs que prend l'information de contrôle "Balayage", des actions spécifiques peuvent être réalisées par certaines des unités composant l'unité de calcul parallèle. Par exemple, lorsque l'indicateur "Balayage" prend les valeurs suivantes, une donnée (constante ou variable d'initialisation) est transmise sur le chemin de données B du pipe-line.

- balayage = 3, donnée B = n° de ligne
- balayage = 4, donnée B = n° de colonne
- balayage = 5, donnée B = n° de région

La possibilité de réinitialiser durant le traitement d'un groupe de pixels les numéros de ligne et numéros de colonne, ainsi que la possibilité de transmettre par la variable "**balayage**" un déplacement décrit par un code de Freeman (balayage = 8 à 15), permettent de minimiser le nombre d'informations qui sont nécessaires pour transmettre à tout instant la position du pixel dans l'image, quelle que soit la loi d'accès (ordre) dans lequel les données sont traitées (cf. paragraphe 2.3.5).

Balayage		Signification
Binaire	Décimal	
0000	0	Aucune action
0001	1	Indicateur de début de séquence
0010	2	Indicateur de fin de séquence
0011	3	Initialisation de l'indice ligne
0100	4	Initialisation de l'indice colonne
0101	5	Initialisation du numéro de région
0110	6	Indicateur de début et de fin de ligne
0111	7	Indicateur de début et de fin de colonne
1000	8	Indice ligne = Indice ligne Indice colonne = Indice colonne + 1
1001	9	Indice ligne = Indice ligne - 1 Indice colonne = Indice colonne + 1
1010	10	Indice ligne = Indice ligne - 1 Indice colonne = Indice colonne
1011	11	Indice ligne = Indice ligne - 1 Indice colonne = Indice colonne - 1
1100	12	Indice ligne = Indice ligne Indice colonne = Indice colonne - 1
1101	13	Indice ligne = Indice ligne + 1 Indice colonne = Indice colonne - 1
1110	14	Indice ligne = Indice ligne + 1 Indice colonne = Indice colonne
1111	15	Indice ligne = Indice ligne + 1 Indice colonne = Indice colonne + 1

figure 2.18 : Signification de l'indicateur de balayage

Les choix qui ont été effectués au niveau de la signification et des actions de l'indicateur de balayage sont liés à des études algorithmiques qui seront présentées dans les paragraphes 3 et 4 de cette thèse. Sans rentrer dans les détails spécifiques à certaines tâches de traitement d'images, nous allons présenter sous la forme d'un exemple l'utilisation d'un tel identificateur associé à chacun des pixels à traiter.

Exemple de balayage :

Si l'on désire traiter une zone d'image de forme non rectangulaire (cf. figure 2.19) suivant un balayage ligne à ligne, par l'unité de calcul parallèle, il faudra par exemple lui transmettre une séquence d'informations ordonnée de la manière suivante (cf. figure 2.20)(on suppose que les pixels à traiter sont transmis sur le chemin de données B).

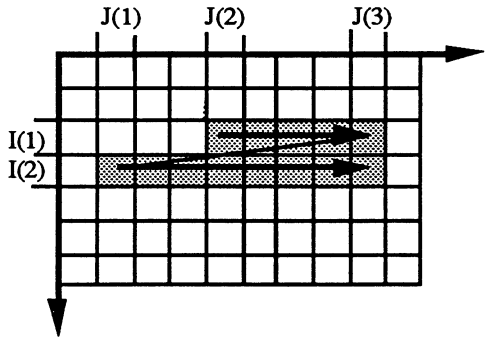


figure 2.19 :
Exemple de balayage d'une zone d'image de forme non rectangulaire

Ordre de traitement

Balayage	Donnée B	Signification	
1	XXX	Début de séquence	
3	I(1)	initialisation de l'indice ligne	
4	J(2)	initialisation de l'indice colonne	
6	XXX	indicateur de début de ligne	
8	Pixel (I(1),J(2))	pixel à traiter	J(2)-J(3) pixels à traiter
...	...	...	
8	Pixel (I(1),J(3))	pixel à traiter	
6	XXX	indicateur de début de ligne	
3	I(2)	initialisation de l'indice ligne	
4	J(1)	initialisation de l'indice colonne	
6	XXX	indicateur de debut de ligne	
8	Pixel(I(2),J(1))	pixel à traiter	J(1)-J(3) pixels à traiter
..	..	..	
8	Pixel(I(2),J(3))	pixel à traiter	
6	XXX	indicateur de fin de ligne	
2	XXX	Fin de séquence	

figure 2.20 : structure de la séquence à traiter

2.4.2.3) Architecture interne

Pour avoir une unité de contrôle centralisée au niveau de l'unité de calcul parallèle, nous avons décidé de diviser l'architecture interne de cette dernière en deux blocs fonctionnels (cf. figure 2.21) :

- une machine à états qui réalise le cadencement du réseau de calcul parallèle;
- un réseau de calcul paallèle composé d'un nombre variable d'unités de calcul fonctionnant simultanément et qui peut être programmé ou configuré par l'unité centrale via des entrées-sorties parallèles dont le fonctionnement est totalement asynchrone des calculs.

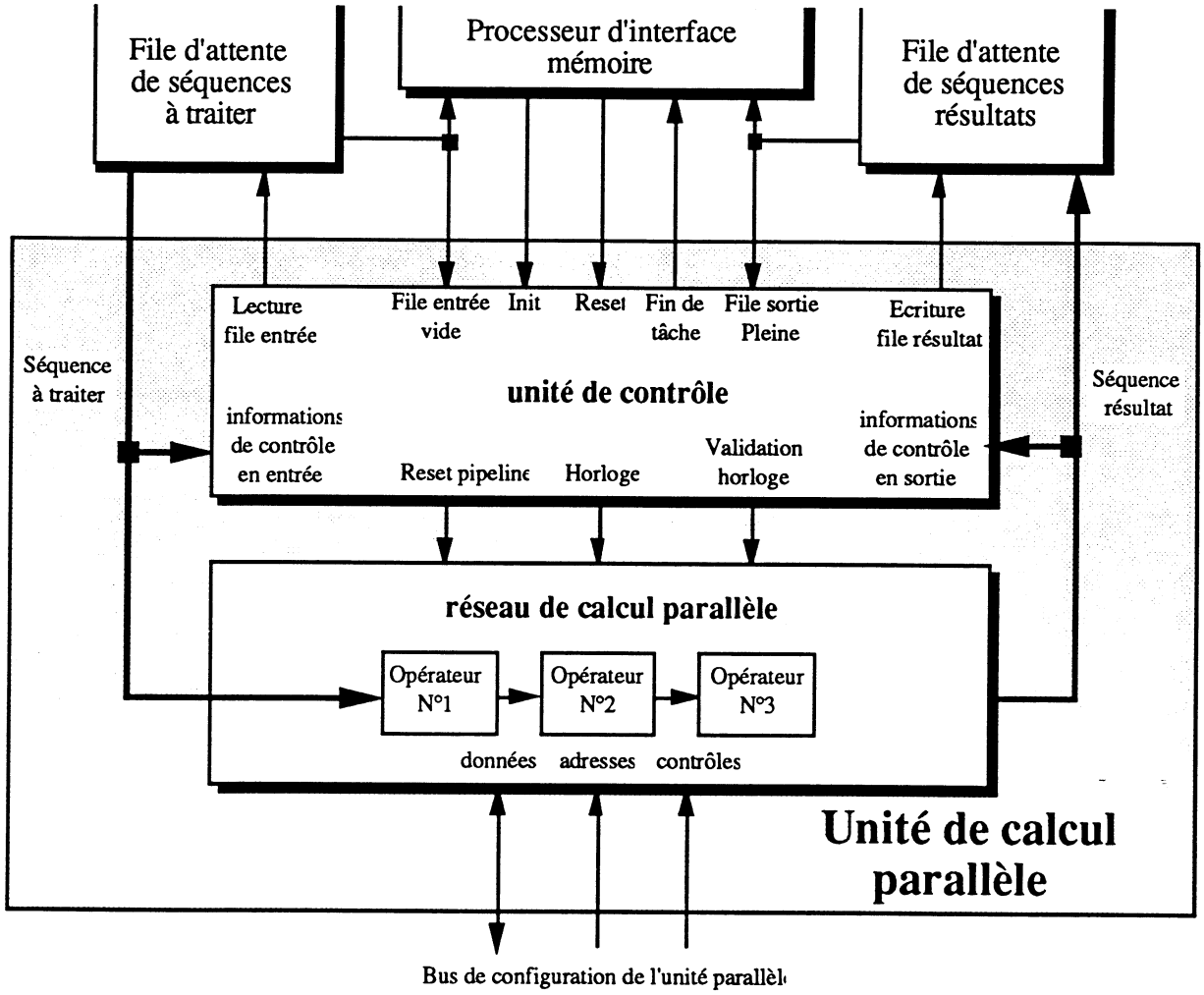


figure 2.21 : Architecture de l'unité de calcul parallèle

L'unité de contrôle est une petite machine à états qui réalise quatre tâches simultanément :

- elle contrôle le séquencement du réseau de calcul parallèle en fonction de l'état de remplissage des files d'attente en entrée et en sortie;
- elle génère les signaux de lecture et d'écriture des files d'attente pour les échanges d'informations avec le réseau de calcul parallèle;
- elle gère le bon déroulement du traitement d'une séquence de données et avertit le processeur d'interface mémoire lorsque le réseau de calcul a terminé le traitement (ceci est réalisé par l'intermédiaire d'une scrutation sur les informations de contrôle disponibles sur les séquences d'entrée et de sortie);
- elle permet, sous le contrôle de l'unité d'interface mémoire, une remise à zéro globale (reset) de tous les chemins de données pipe-line internes du réseau de calcul parallèle et facilite ainsi son test fonctionnel.

2.4.3) Architecture des unités de calcul

Le réseau de calcul parallèle sera composé d'un nombre variable d'unités de calcul, la majorité des possibilités d'évolution de cette structure, que ce soit au niveau des performances ou des fonctionnalités, sera réalisée en utilisant un nombre variable d'unités de calcul qui ne sont pas nécessairement identiques. Quel que soit le nombre d'unités de calcul qui composeront le réseau, les protocoles et formats des échanges d'information avec les files d'attente resteront identiques.

2.4.3.1) Principes généraux

Les principes généraux régissant le fonctionnement des unités de calcul sont identiques à ceux qui ont été précédemment définis au niveau du réseau de calcul global (cf. chapitre 2.4.2). Le traitement est cadencé par une horloge unique et à chaque coup d'horloge une opération et un pixel sont traités et, de même, un pixel est transmis sur les entrées du circuit et un sur les sorties.

2.4.3.2) Interfaces externes

Les interfaces externes d'une des unités de calcul seront identiques aux interfaces externes de l'accélérateur de traitement global (cf. figure 2.16). Cette méthode permet d'avoir un réseau de calcul parallèle qui est vu de la même manière quel que soit le nombre d'opérateurs élémentaires qui le composent (de 1 ou plus à n).

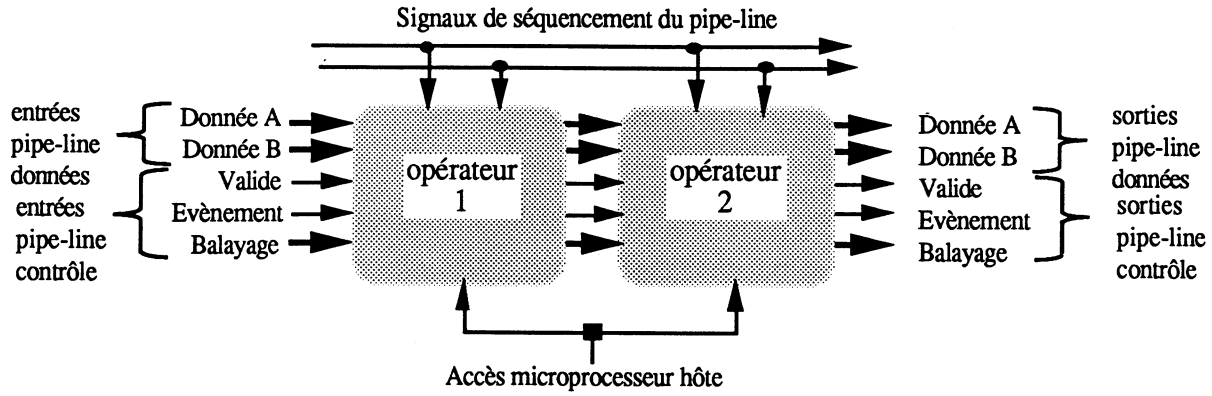


figure 2.22 :
Structure d'un réseau de calcul parallèle composé de deux opérateurs en cascade

2.4.3.3) Architecture interne

Nous avons maintenant défini très précisément l'architecture externe d'un opérateur de traitement et ses mécanismes de séquencement pour qu'il s'interface dans notre structure. Son architecture interne et ses méthodes de traitement n'ont pas d'importance pourvu qu'il réponde à ces spécifications.

Nous avons cependant adopté une démarche commune pour la conception des deux circuits INP (Image Neighbourhood Processor) et IFP (Image Feature Processor) qui sont décrits dans les chapitres 3 et 4 et qui correspondent à deux opérateurs de traitement typiques. Le principal objectif de cette démarche a été de réaliser des dispositifs spécifiques capables néanmoins de pouvoir se reconfigurer pour réaliser différents types d'algorithmes de traitement d'images. La démarche adoptée lors de leur conception s'est articulée en quatre étapes :

- la première étape étudie les similitudes entre une série d'algorithmes classiques afin de les regrouper en familles jouissant de propriétés communes de type mathématique, topologique ou simplement liées à un mode de décomposition. Cette opération cherche à mettre en évidence des caractéristiques exploitables au niveau architectural, comme un mode de parallélisme, un jeu d'opérateurs, une structure élémentaire. Cette étape s'achève par une représentation des algorithmes sous forme de graphes de circulation de données;

- la seconde étape a pour but d'élaborer à partir des représentations fonctionnelles de ces algorithmes une structure de graphe commune minimisant le nombre de noeuds et de liens;

- la troisième étape essaie d'élargir au maximum la famille des algorithmes implantables sous ces formes de graphes. Cette étape aboutit généralement à une évolution des graphes initiaux et est sujette à de nombreux compromis;

- la quatrième étape a pour but l'étude d'une architecture électronique intégrable capable de réaliser ce graphe tout en respectant nos règles d'interfaçage.

Il ne reste plus que l'implantation, le test et la réalisation de ce circuit intégré pour avoir un opérateur adapté à notre architecture.

Les deux circuits que nous décrivons dans la suite de ce mémoire ont été conçus en suivant les mêmes principes, leurs architectures internes n'ont par contre aucun rapport.

2.5) Conclusion

L'objectif de ce chapitre était de spécifier les fonctionnalités et l'architecture générale d'un accélérateur de traitement d'images parallèle polyvalent qui puisse s'intégrer dans notre architecture du module. A partir des contraintes liées à la nature des algorithmes que nous désirons réaliser et de celles liées à la réalisation matérielle d'une telle structure, nous avons décidé de diviser notre "coprocesseur de traitement d'images en deux unités distinctes asynchrones échangeant, via des files d'attente (FIFO), des séquences d'informations (données + contrôles).

La première unité (unité d'interface bus) réalise l'interface avec le bus système du module et permet des échanges d'informations avec la mémoire locale, elle peut être passive ou active (cf. 2.2.2.5). Seules les fonctionnalités et les différentes manières de réaliser cette unité seront décrites dans la suite ce chapitre.

La seconde unité (unité de traitement de séquences) est passive et est une structure parallèle pipe-line et systolique qui intègre dans une structure linéaire, un nombre variable d'opérateurs spécifiques synchrones échangeant également entre eux des séquences. Chacun des opérateurs spécialisés de l'unité de traitement de séquence sera une petite structure parallèle, qui pourra être reconfigurée pour réaliser de multiples algorithmes de traitement d'images. Ces opérateurs spécialisés n'étant pas commercialisés à l'heure actuelle, leurs spécificités et leurs complexités imposent qu'ils soient réalisés en utilisant des techniques d'intégration de la microélectronique.

L'ensemble du travail qui sera présenté dans la suite de ce mémoire sera exclusivement consacré à l'étude approfondie de l'unité de traitement de séquences et des opérateurs qui la constitueront.

Les principes architecturaux de cet accélérateur de traitement d'images ont été validés dans deux réalisations matérielles :

- les échanges de séquence entre les deux unités via des files d'attente ont été mis en oeuvre dans le dispositif de test du circuit INP20 (VAR88) présenté dans le paragraphe 3;
- l'utilisation de lois d'accès complexes pour traiter les pixels suivant des balayages non nécessairement de type vidéo et la transmission d'informations de contrôle associées aux pixels à traiter, a été mise en oeuvre dans le système de traitement d'images monocarte OP2 (THE89) réalisé au LETI. Il est à noter que ce dispositif permet de traiter des pixels à des cadences supérieures à 15 MHz pour des lois de traitement qui peuvent être excessivement complexes (exemple : rotation d'images).

Les spécifications de base que nous avons définies au début de cette étude pour notre accélérateur de traitement ne nous paraissent pas trop restrictives deux ans après. Nous avons cependant, lors de nos parallélisations d'algorithmes, été parfois limités par l'unidirectionnalité des échanges d'informations entre opérateurs. Il serait possible d'accroître les possibilités de développement d'opérateurs de calcul en disposant, dans la structure, d'un chemin de données bidirectionnel pour les échanges d'informations inter-opérateurs (chemin de donnée A).

Troisième partie

Etude d'un processeur de traitement d'image spécialisé dédié aux opérations de voisinage sur des images

INP : "Image Neighbourhood Processor"

La première version intégrée de ce processeur spécialisé à pour nom INP20, il est actuellement commercialisé par le CEA/DLETI et a fait l'objet des publications ci-dessous.

" INP20 : An image Neighbourhood processor for large kernels."
D. David, T. Court, J.L. Jacquot, A. Pirson
IAPR workshop on computer vision, Tokyo, Japon,
october 12-14, 1988 pp (241, 244)

" A highly efficient method for synthesizing some digital filters ."
A. Pirson, T. Court, D. David, J.L. Jacquot
EUSIPCO 88, fourth european signal processing conference,
Grenoble, France, septembre 1988 pp (1481, 1484)

"Une classe d'extraction de contours performants adaptés à une implémentation matérielle fonctionnant à cadence vidéo."
J.L. Jacquot, T. Court, D. David, A. Pirson
TIPI 88, second atelier "du traitement du pixel à l'interprétation",
Aussois, France,avril 1988

"Procédé de traitement de signaux numérisés représentatifs d'une image origine."
D. David, T. Court, CEA/CENG/DLETI
Brevet Francais : n° 88 11594 du 5 septembre 1988

"INP20 Image Neighbourhood Processor."
Application Specific Image Processors
Documentation préliminaire, CEA/IRDI/DLETI , 1988

Sommaire de la Troisième partie

3.1) Introduction

Les premières phases de traitements, qui sont généralement réalisées sur des images, sont dites de "bas niveau" (cf 1.1.2). Elles ont pour but de restaurer les images et/ou de mettre en évidence certaines de leurs caractéristiques. Ces tâches utilisent comme données d'entrée des images et génèrent des résultats qui sont eux-mêmes des images.

Récemment (1987 -> 1990), de nombreux circuits spécialisés adaptés à la réalisation de ce type de tâches (cf. chapitre 1.2.3.2)) ont été échantillonnés sur le marché des composants électroniques. La quasi totalité des circuits actuellement commercialisés, réalise une seule tâche de traitement sur des images dont les pixels sont disponibles séquenciellement suivant des formats vidéo (balayage ligne à ligne). Seuls quelques circuits tels que ISP1 et ISP2 (FUK84, FUK85, FUK86, KOB87, TAK88, HIT88) sont reconfigurables et capables de réaliser différents types de tâches de bas niveau.

Ce chapitre présente une architecture matérielle intégrable capable de réaliser suivant sa programmation différentes tâches de traitement bas niveau. Les structures interne (séquencement des calculs) et externe (communication entre circuits) de cet opérateur, satisfont aux contraintes générales que nous avons définies pour l'architecture de "l'accélérateur de traitement d'image" (cf. 2.3.2).

La présentation de ce circuit est organisée en quatre parties. Dans la première nous décrivons son architecture externe et une première décomposition de sa structure interne en unités fonctionnelles. La seconde partie présente les tâches que nous désirons pouvoir réaliser à l'aide de ce circuit et décrit les ressources matérielles qu'il est nécessaire d'intégrer pour leur mise en oeuvre. La troisième partie est consacrée à l'architecture générale du circuit. La quatrième partie décrit une première intégration de certains blocs fonctionnels de ce circuit et des méthodes qui ont été utilisés pour son test.

La dénomination de ce circuit est I.N.P. "Image Neighbourhood Processor". Sa première version intégrée a eu pour nom INP20, car elle permettait de réaliser des opérations de type morphologie ou convolution pour des voisinages de taille maximum vingt pixels.

3.2) Architecture macroscopique

Ce paragraphe a pour but de présenter brièvement les fonctionnalités que nous attendons de ce circuit. Il décrit sa structure externe et présente macroscopiquement sous forme de graphes de circulation de données, les tâches qu'il doit être capable de réaliser.

3.2.1) Généralités

3.2.1.1) Informations circulant en pipeline

Le circuit traite des séquences d'informations. A chaque cycle de calcul du pipe-line (un cycle d'horloge), il traite un élément (un pixel). Pour chaque pixel traité, deux groupes d'informations (données, contrôle) circulent en parallèle sur les entrées et sorties pipe-line du circuit (fig. 3.1). Pour permettre l'intégration de ce circuit dans notre architecture globale, le format de ses communications externes, pour les informations données et contrôle, respecte l'ensemble des protocoles que nous avons définis pour les dialogues entre les différents circuits de "l'accélérateur de traitement d'image" (cf. 2.3.2).

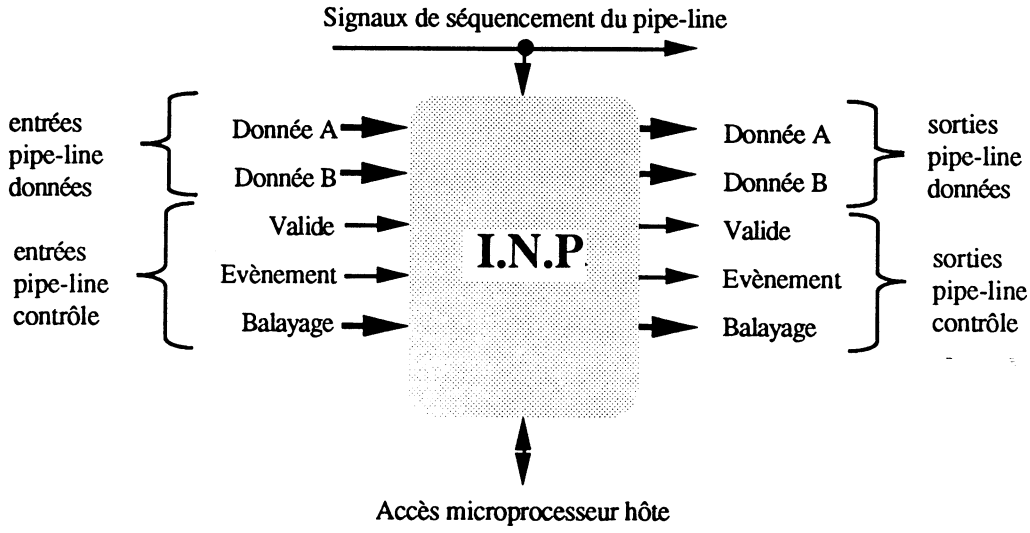


Figure 3.1 : INP, communications externes

La nature et le format des informations lues sur les entrées, leurs influences sur les différentes tâches de traitement réalisées par le circuit sont décrits ci-aprés :

Informations de type données :

Donnée A : (entier 16 bits, codé en complément à 2) valeur du pixel de l'image à traiter (image A), il peut appartenir à l'image d'origine ou être le résultat d'une phase antérieure de traitement pipe-line .

Donnée B : (entier 16 bits codé en complément à 2) valeur du pixel de la seconde image opérande (image B), information particulière pour les traitements d'extraction d'information (numéro de la région à laquelle appartient le pixel), ou, information globale durant les phases de réinitialisation du pipe-line en cours de traitement (cf. 2.3.2).

Informations de contrôle

Valide : (booléen) il indique si le pixel doit ou non être pris en compte dans les opérations de calcul de mesures.

Evènement : (booléen) il indique si le pixel doit ou non être pris en compte lors des opérations d'extraction de listes. Cette information peut être modifiée par le circuit.

Balayage : (codé sur 4 bits) il décrit les modes de balayage de l'image (ordre d'accès aux pixels) et permet de calculer pour chaque pixel sa position dans l'image (Indice ligne, Indice colonne). Le circuit ne recalcule pas de manière interne la position des pixels.

Accès microprocesseur

L'accès microprocesseur ne dépend pas du fonctionnement pipe-line, il permet de configurer le circuit (programmation). Tous les résultats des calculs sont émis en pipe-line sur les sorties du circuit, cet accès ne sert donc pas pour la lecture des résultats.

3.2.1.2) Nature des tâches réalisées

Ce paragraphe décrit la manière dont les tâches de traitement d'image, que nous désirons réaliser à l'aide de ce circuit, s'intègrent dans le pipe-line de calcul de l'accélérateur de traitement. La présentation détaillée de leurs applications et de leurs structures algorithmiques sera faite dans le paragraphe (3.3).

Ce circuit est capable de réaliser un seul type de tâches pipe-line : transformation d'une séquence de pixels d'entrée en une séquence de pixels résultat.

L'algorithme de traitement pour un pixel peut dépendre :

- de pixels qui ont déjà été traités;
- de l'ordre dans lequel les pixels sont traités (mode de balayage de l'image).

Toutes ces tâches de traitement peuvent être décrites sous la forme d'un même processus élémentaire de traitement (cf. fig.3.2) qui, à chaque cycle de calcul pipeline (traitement d'un pixel), réalise séquentiellement les trois sous-tâches suivantes:

- lecture des informations (données, contrôle) disponibles sur les entrées;
- exécution d'un algorithme de traitement utilisant les informations lues sur les entrées et des informations stockées temporairement dans le circuit.
- écriture d'informations (données, contrôle) sur les différentes sorties du circuit.

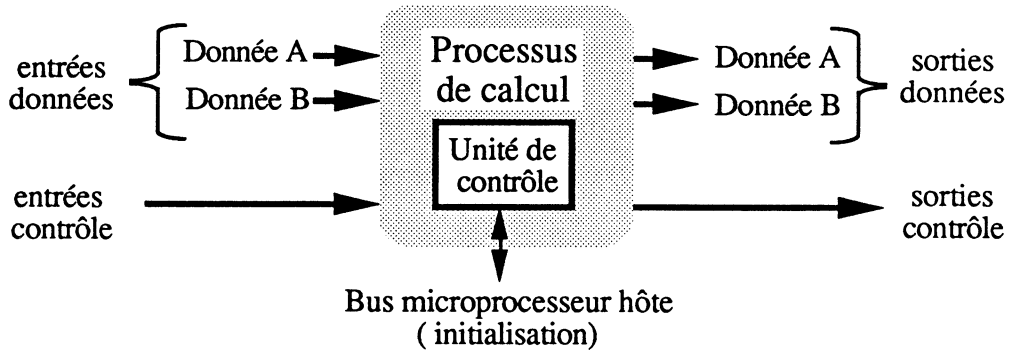


Figure 3.2 : INP, structure générale des processus de traitement

Ces processus utilisent les informations circulant sur les chemins de contrôle et, modifient éventuellement l'une d'entre elles lors d'opérations de marquage (information d'événement). Ils peuvent être classés en quatre groupes suivant leur action sur les informations circulant sur les chemins de données.

* *Opération entre deux images*

Ils remplacent l'une des informations circulant sur les chemins de donnée pipeline par le résultat d'une opération diadique (2 opérandes) sur les données d'entrée (fig. 3.3), ces opérandes peuvent être les valeurs de deux pixels ou un pixel et une constante.

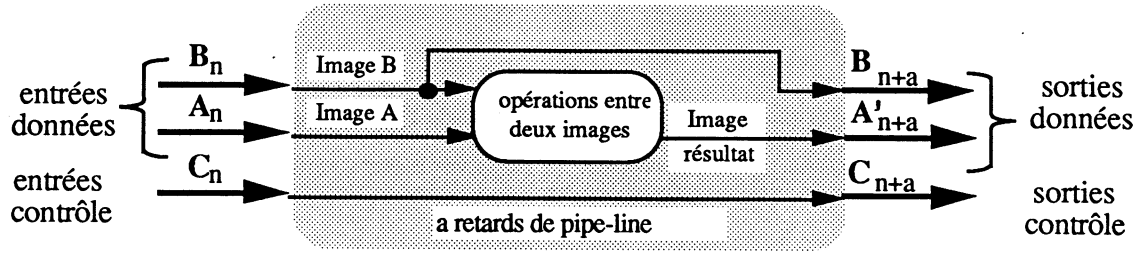


Figure 3.3 : Structure des processus réalisant des opérations entre deux images

* *Processus d'opérations de voisinage (convolution, morphologie)*

Ils modifient l'une des informations circulant sur les chemins de donnée pipeline en la remplaçant par un résultat d'une opération de voisinage sur les pixels en entrée (fig. 3.4);

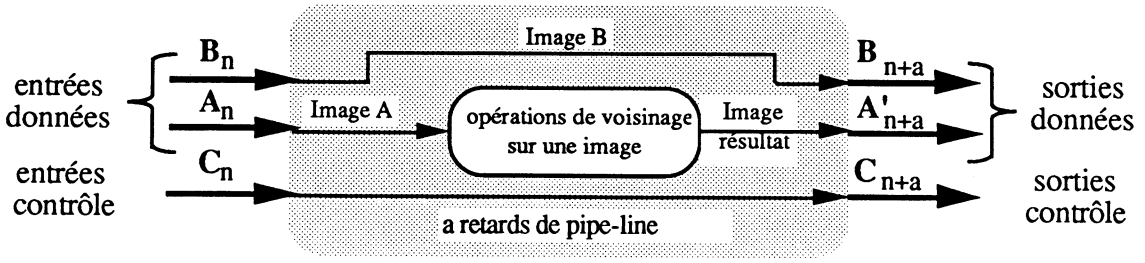


Figure 3.4 : Structure des processus de voisinage

* *Processus de marquage*

Ils modifient l'information "Evénement" circulant sur les chemins de contrôle pipeline en la remplaçant en fonction du résultat d'une opération entre deux images ou entre une image et une constante (fig. 3.5);

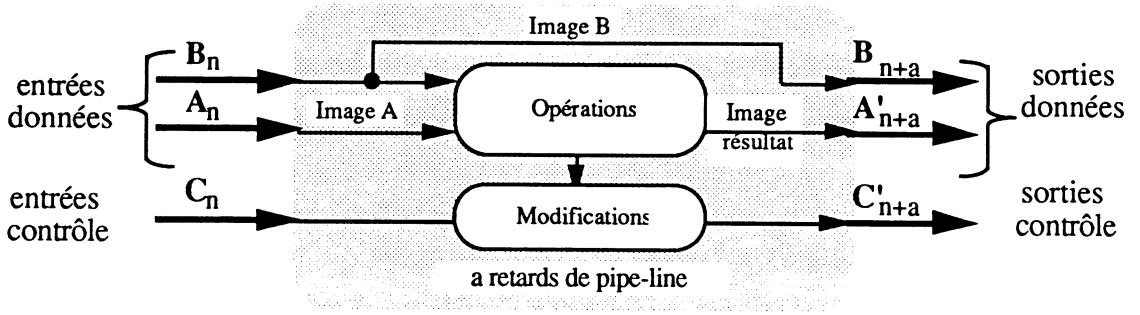


Figure 3.5 : Structure des processus de marquage

** Processus mixtes*

Ils réalisent simultanément des tâches d'opération de voisinage, d'opérations entre deux images et d'opération de marquage, un exemple est donné dans la figure 3.6.

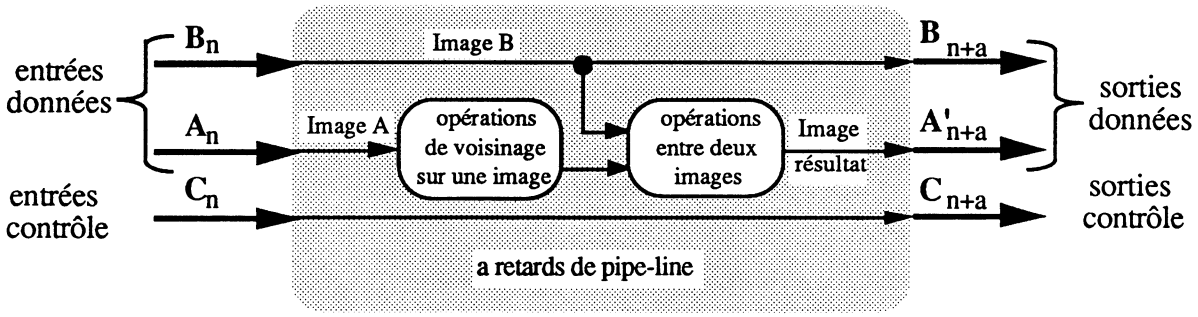


Figure 3.6 : Exemple de processus mixte

3.2.2) Les fonctions de base

Les opérations entre deux images pixel à pixel sont bien connues et il n'est pas utile de décrire leur principe. Par contre, il est nécessaire, pour une bonne compréhension de la suite de ce chapitre, de présenter brièvement les opérations de traitement d'images mettant en jeu des calculs sur des voisinages de pixels.

Toutes les opérations de traitement d'images dites "de voisinage" peuvent être décrites de la manière suivante (cf. figure 3.7) : pour calculer chaque pixel de l'image résultat, on extrait de l'image de départ une fenêtre contenant le pixel de départ et ses voisins (suivant les lignes, les colonnes ou dans deux dimensions) et l'on calcule le pixel résultat en réalisant des opérations combinatoires, prenant en compte la valeur des pixels du voisinage et des constantes.

Dans le cas de la convolution, l'opération combinatoire réalisée est une combinaison linéaire (pondération, sommation) des valeurs des différents pixels de la fenêtre (SAM84, SER82).

Dans le cas de la morphologie mathématique, cette opération combinatoire peut être une comparaison (minimum, maximum) ou une opération booléenne (et, ou) des valeurs des différents pixels de la fenêtre, généralement appelés éléments structurants (SER82, VER88).

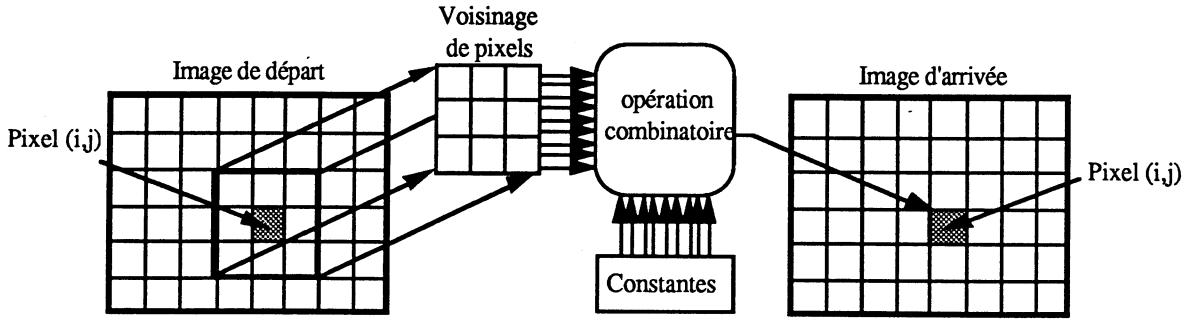


figure 3.7 : Principe de base des opérations de voisinage

De nombreuses études théoriques ont défini des noyaux de convolutions et des éléments structurants (morphologie) pour réaliser plus particulièrement des tâches de restauration d'images, de filtrage ou pour mettre en évidence certaines de leurs caractéristiques.

Les conclusions communes à un grand nombre de ces travaux mettent en évidence que ce type d'opérations de voisinage est d'autant plus efficace que les voisinages utilisés sont de grande taille (de 5 x 5 à 31 x 31 pixels). Il est très difficile de réaliser dans un temps réduit ces tâches, car le nombre d'opérations élémentaires nécessaires pour calculer un seul pixel résultat est très important (de 18 pour un voisinage (3*3) à 1900 opérations pour un voisinage (31*31) pour le calcul de chaque pixel de l'image résultat).

D'autres travaux ont eu pour objet la recherche de méthodes visant à réduire le nombre de calculs nécessaires pour mettre en oeuvre ce type d'algorithme (BUR84, BUR83, ELP87, FER86, LEA88, WEL86, WIE85). La méthode qui est le plus couramment utilisée est, dans le cas de la convolution, l'utilisation de filtres séparables (WIE85), elle permet de réduire d'un facteur dimensionnel, le nombre d'opérations nécessaires (de n x n à 2n). Il est possible, grâce à cette méthode, de réduire notablement le temps ou la quantité d'électronique nécessaire pour réaliser ces tâches, mais ce n'est pas toujours suffisant.

Le concept de séparabilité pour la convolution :

Soit un filtre de convolution bi-dimensionnel de réponse impulsionnelle finie h (i, j). La convolution d'une image I (m, n) par h (i, j) donne une nouvelle image I_R (m, n) de la manière suivante :

$$I_R(m,n) = \sum_i \sum_j h_{i,j} \cdot I(m-i,n-j)$$

Où $h_{i,j}$ sont les coefficients du filtre h (i, j).

Si l'on applique la transformée en Z bi-dimensionnelle à cette équation, on obtient :

$$I_R(z_1,z_2) = \sum_i \sum_j h_{i,j} . z_1^{-i} . z_2^{-j} . I(z_1,z_2)$$

Ce filtre est dit séparable si sa fonction de transfert $H(z_1, z_2)$ peut être écrite sous la forme d'un produit de deux transformées monodimensionnelles $H_1(z_1)$ et $H_2(z_2)$.

$$H(z_1, z_2) = \sum_i a_i z_1^{-i} . \sum_j b_j z_2^{-j} = H_1(z_1) . H_2(z_2)$$

Où a_i et b_j sont respectivement les coefficients des filtres h_1 et h_2.

D'où l'équation suivante :

$$I_R(z_1,z_2) = \sum_i a_i . z_1^{-i} . \sum_j b_j . z_2^{-j} . I(z_1,z_2)$$

Si l'image est balayée ligne à ligne (balayage vidéo désentrelacé ou balayage progressif), elle devient un signal unidimensionnel où la transformée en Z à une dimension peut être appliquée. Si N représente le nombre de pixels dans une ligne, alors $z_1^{-N} = z_2^{-1}$.

L'équation devient dans ce cas :

$$S_R(z) = \sum_i a_i z^{-i.N} . \sum_j b_j z^{-j} S(z)$$

Où S et S_R sont les signaux correspondant à l'image de départ I et d'arrivée I_R.

La séparabilité réduit une opération bi-dimensionnelle sur un voisinage (nxm) à deux opérations unidimensionnelles appliquées successivement sur des voisinages (n) et (m).

Des études de minimisation de la complexité de calcul de ces algorithmes ont été poussées plus loin, et diverses méthodes (WEL86, DUR83) ont été développées, qui nécessitent pour chaque pixel, un nombre d'opérations qui est inférieur au nombre de pixels pris en compte dans le voisinage.

Nous allons, dans la suite de ce rapport, décrire une nouvelle méthode pour réaliser des opérations de voisinage (convolutions, morphologies) qui minimise le nombre d'opérations (PIR90, DAV88, JAC88, PIR88).

3.3) Un peu de théorie...

Pour ne pas imposer aux lecteurs douze pages de théorie pure, nous allons présenter, sous la forme d'exemples, les principes de parallélisation des opérations de voisinage que nous avons développés et cette description évoluera progressivement vers la définition d'une structure parallèle appropriée à leur mise en oeuvre. Si le lecteur désire de plus amples renseignements sur ces principes théoriques, nous lui conseillons de se reporter aux publications suivantes : JAC88, PIR88, DAV88.

3.3.1. Principes théoriques pour la convolution

Premier exemple :

La convolution d'un signal échantillonné x_n par un filtre à réponse impulsionnelle finie et à coefficient uniforme peut s'écrire de la manière suivante :

$$y_n = \sum_{k=0}^{m-1} x_k$$

Si nous appliquons la transformée en z à l'équation ci-dessus, nous obtenons :

$$Y = X \sum_{k=0}^{m-1} z^{-k} = X \frac{1 - z^{-m}}{1 - z^{-1}}$$

Cette expression peut être interprétée comme un système linéaire où x et y sont respectivement les signaux d'entrée et de sortie et où l'expression $(1 - z^{-m})/(1 - z^{-1})$ est la transformée en z de la réponse impulsionnelle du système. Cette expression correspond au schéma fonctionnel suivant (figure 3.8) :

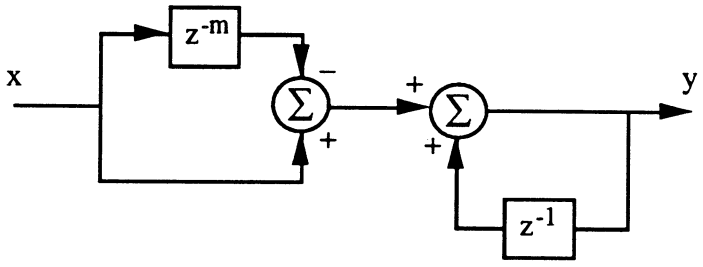


figure 3.8 : Schéma fonctionnel du filtre uniforme

Nous pouvons remarquer que pour réaliser un filtrage uniforme sur un nombre infini d'échantillons, il est seulement nécessaire de réaliser une addition, une soustraction, un retard 1 et un retard m pour chaque échantillon, quelle que soit la taille du noyau du filtre uniforme.

Deuxième exemple :

Le gradient horizontal de Sobel a l'expression suivante :

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

Sa fonction de transfert peut être décrite par l'équation suivante :

$$Y = (1 + 2.z^{-1} + z^{-2}).(1 - z^{-2.N}).X = (1 + z^{-1})^{-2} . (1 - z^{-2.N}).X$$

Ce qui conduit, en utilisant, pour cette équation, le même mécanisme de représentation que dans le premier exemple, au schéma fonctionnel suivant (figure 3.9) :

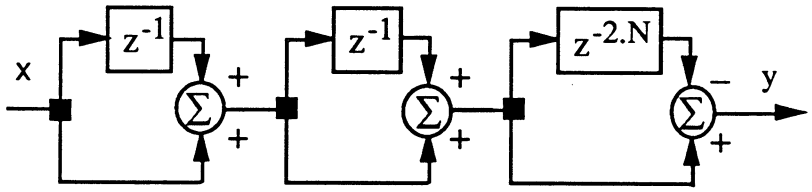


figure 3.9 : Bloc diagramme du gradient de sobel

Nous pouvons remarquer, à partir de ces deux exemples, qu'il est possible de synthétiser ces filtres en cascadant certaines cellules élémentaires de filtrage.

Ces cellules sont de deux types :

- **la cellule directe**
- **la cellule récursive.**

La cellule directe correspond au bloc fonctionnel suivant (figure 3.10) :

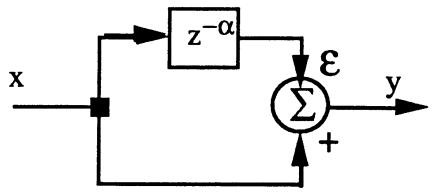


Figure 3.10 : Schéma fonctionnel de la cellule directe

Sa fonction de transfert est : $F(z) = 1 + \varepsilon z^{-\alpha}$, où ε vaut -1 ou +1 et α appartient à N_0. L'opération arithmétique utilisée dans cette cellule est une addition ou une soustraction.

La cellule récursive correspond au bloc fonctionnel suivant (figure 3.11) :

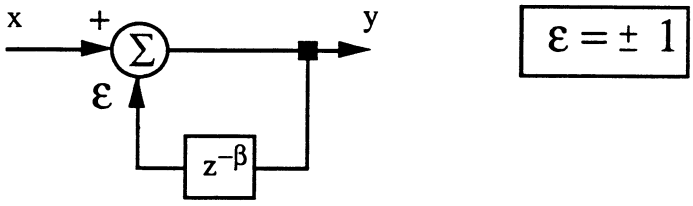


Figure 3.11 : Schéma fonctionnel de la cellule recursive

Sa fonction de transfert est : $R(z) = 1/(1 + \varepsilon z^{-\beta})$, où ε vaut -1 ou +1 et β appartient à N_0. L'opération arithmétique utilisée dans cette cellule est une addition ou une soustraction.

Une cascade quelconque de cellules récursives et directes permet d'obtenir la caractéristique de transfert suivante :

$$H(z) = \frac{\prod_{i=1}^{N_f} (1 - \varepsilon_i z^{-\alpha_i})}{\prod_{j=1}^{N_r} (1 - \varepsilon_j z^{-\beta_j})}$$

Où $\varepsilon_i + \varepsilon_j$ valent -1 ou +1 et α_i, b_j appartiennent à N_0, N_f et N_r sont respectivement les nombres de cellules directes et inverses.

Les principaux intérêts de cette approche sont les suivants :

- il n'y a pas de multiplication, seulement des additions
- le nombre d'opérations nécessaire pour réaliser le calcul est extrèmement réduit, le plus souvent inférieur à la taille du voisinage

Cette méthode très efficace, ne permet pas de réaliser n'importe quelle fonction de transfert, car les filtres de convolutions ne peuvent pas tous être décomposés sous la forme d'un quotient de produits de monomes réalisables par nos cellules élémentaires. Cependant, de très nombreux filtres de convolutions bien connus correspondent à ce modèle.

Nous pouvons citer pour exemple :

- tous les filtres à coefficients uniformes de voisinage rectangulaire;
- les filtres de gradient classiques (Prewitt, Sobel, Roberts);
- certains filtres Laplaciens;
- les extracteurs de texture de Laws;
- certains filtres passe-bas et gradients de grande taille (cf. figure 3.12).

$$\begin{bmatrix} -1 & -5 & -10 & -9 & 0 & 9 & 10 & 5 & 1 \\ -4 & -20 & -40 & -36 & 0 & 36 & 40 & 20 & 4 \\ -8 & -40 & -80 & -72 & 0 & 72 & 80 & 40 & 8 \\ -12 & -60 & -120 & -118 & 0 & 118 & 120 & 60 & 12 \\ -14 & -70 & -140 & -126 & 0 & 126 & 140 & 70 & 14 \\ -12 & -60 & -120 & -118 & 0 & 118 & 120 & 60 & 12 \\ -8 & -40 & -80 & -72 & 0 & 72 & 80 & 40 & 8 \\ -4 & -20 & -40 & -36 & 0 & 36 & 40 & 20 & 4 \\ -1 & -5 & -10 & -9 & 0 & 9 & 10 & 5 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \\ 8 \\ 12 \\ 14 \\ 12 \\ 8 \\ 4 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & -5 & -10 & -9 & 0 & 9 & 10 & 5 & 1 \end{bmatrix}$$

$$H(z_1,z_2) = (1+z_2^{-1})^4 . (1+z_2^{-2})^2 . (1+z_1^{-1})^5 . (1-z_1^{-3})$$

figure 3.12 : Exemple de gradient 9 x 9 factorisable

Afin d'évaluer l'efficacité de cette méthode pour des cas réels, deux études ont été réalisées :

La première a eu pour objet la réalisation d'un programme de factorisation polynomiale qui essaye de décomposer un filtre quelconque dans des suites de monomes telles que nous les avons décrites. Ce logiciel permet de vérifier si un filtre de convolution quelconque est factorisable et réalisable par notre méthode (CEA88).

La seconde a eu pour objet la réalisation d'un logiciel recherchant la meilleure approximation de filtres de convolution quelconque par notre méthode. Cette dernière est réalisée en minimisant les différences, dans le domaine fréquenciel, des caractéristiques des filtres (norme quadratique) (JAC88).

Les résultats de ces deux études ont été suffisamment probants pour que nous considérions que la seule mise en oeuvre, dans notre coprocesseur, de ces filtres de convolution couvre un large éventail de méthodes utilisées en vision industrielle.

3.3.2) Généralisation à la morphologie mathématique

Pour les opérations de morphologie mathématique sur des images binaires ou en niveaux de gris (transformée en chapeau plat), dans le cas de l'érosion monodimensionnelle, on applique l'opération suivante :

$$y_m = \min_{k \in B} (X_{m+k})$$

où B est le support de l'élément structurant.

Dans le cas d'une opération bi-dimensionnelle, l'opération est identique :

$$y_{m,n} = \min_{k \in B} (X_{m+k,\, n+l})$$

où B est le support de l'élément structurant bi-dimensionnel.

Comme pour le filtrage linéaire (convolution), certains filtres de la morphologie peuvent être séparables et être réalisés par deux opérations unidimensionnelles appliquées successivement. Nous avons dans ce cas :

$$y_{m,n} = \min_{k \in B_1} \left(\min_{k \in B} (X_{m+k,\, n+l}) \right)$$

Exemple :

Soit une érosion unidimensionnelle sur un voisinage de quatre échantillons :

$$y_n = \min (x_n, x_{n+1}, x_{n+2}, x_{n+3})$$

Si nous définissons un opérateur de retard R^{-k}, généralisation de la transformée en Z pour le cas non linéaire, tel que $R^{-k}(x_n) = x_{n+k}$ et $R^{-1}(x_n) = x_{n+1}$, cette équation peut être écrite de la manière suivante :

$$y_n = \min (x_n, R^{-1}(x_n), R^{-2}(x_n), R^{-3}(x_n))$$
$$y_n = \min (\min (x_n, R^{-1}(x_n)), \min (R^{-2}(x_n), R^{-3}(x_n)))$$
$$y_n = \min (\min (x_n, R^{-1}(x_n)), R^{-2}(\min (x_n), R^{-1}(x_n)))$$

La dernière équation peut être représentée sous la forme du diagramme fonctionnel suivant (figure 3.13) :

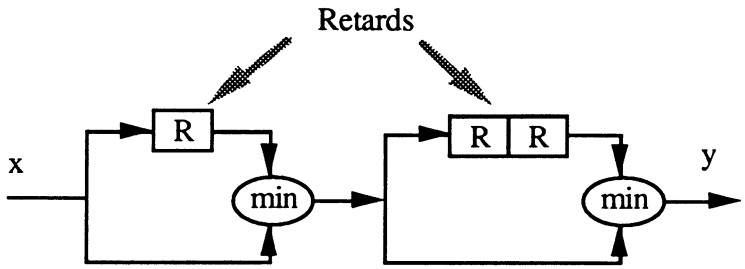


figure 3.13 :
Diagramme fonctionnel des calculs pour l'érosion sur quatre voisins

En généralisant cette méthode, il est possible de réaliser des opérations morphologiques sur des éléments structurants unidimensionnels de taille m en associant M cellules en cascades où M est le premier entier supérieur ou égal à log2 (m). La cellule élémentaire dans ce cas (figure 3.14) est quasi-identique à une cellule directe pour la convolution, mais l'opération arithmétique est remplacée par le calcul du minimum et du maximum.

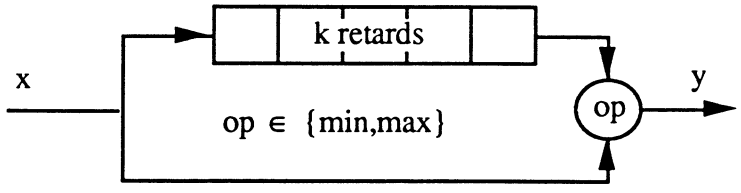


figure 3.14 : schéma fonctionnel de la cellule élémentaire de base

3.3.3) Définition d'une structure de calcul commune à la famille d'algorithmes

Nous avons, dans les deux paragraphes précédents, déterminé une famille de cellules élémentaires (opérateur) qui seraient capables, suivant leur configuration et leur association en cascade, de réaliser indifféremment des opérations de convolution et de morphologie mathématique. Nous allons maintenant, modifier progressivement l'architecture de la cellule de base (figure 3.15) afin d'élargir la gamme d'algorithmes qu'elle sera capable de réaliser.

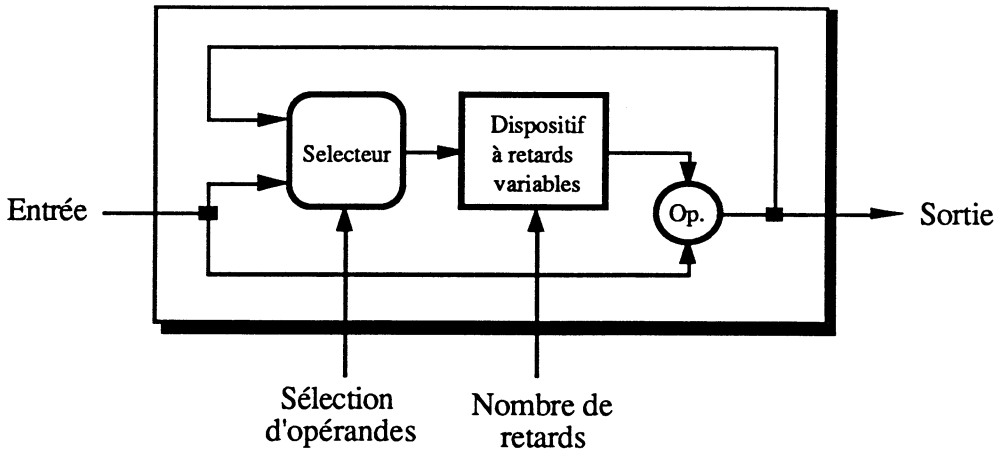


figure 3.15 : Structure de la cellule de calcul de base

Après des études sur la mise en oeuvre d'algorithmes de traitement d'images (convolution, morphologie) classique avec ces cellules, nous avons fait les constatations suivantes :

- dans la majorité des cas, les cellules nécessitent des dispositifs à retards de relativement "petite taille" (4 retards ou 4 retards lignes);

- dans certains cas très particuliers (filtre de grande taille à coefficients uniformes) et uniquement sur des cellules directes, il est nécessaire de disposer de dispositifs à retard de "grande taille" (de 5 à 30 retards ou de 5 à 30 retards lignes).

Pour ne pas surdimensionner la cellule lors de sa future réalisation matérielle, nous avons décidé de remanier la structure de la cellule afin de permettre, en associant deux cellules ou plus en cascade, de réaliser une cellule identique mais disposant de retards plus importants. Cette amélioration nous conduit à la version plus sophistiqué de la cellule décrite en figure 3.16.

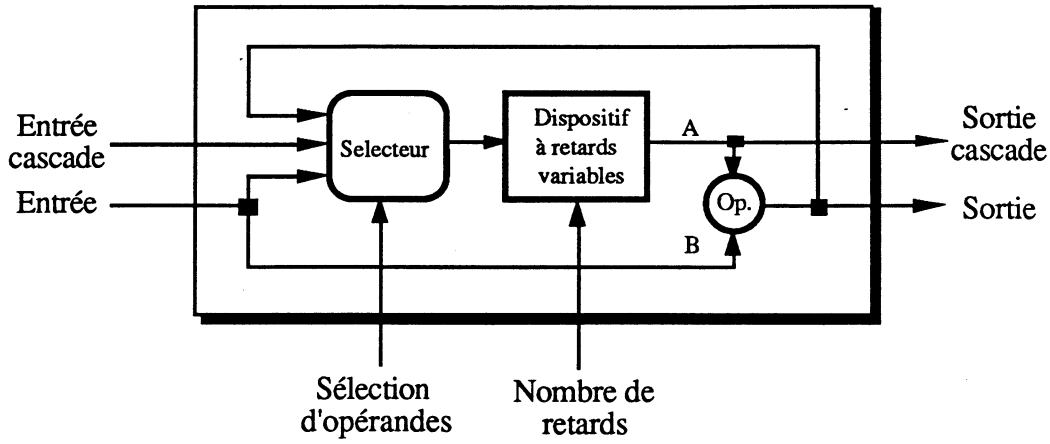


figure 3.16 :
Structure de la cellule améliorée pour cascader les dispositifs de retards

Dans le cas ou deux cellules sont associées pour réaliser une cellule disposant d'un retard plus important, les informations à retarder se propagent de cellule en cellule par les entrées et sorties de "cascade" (cf. figure 3.17).

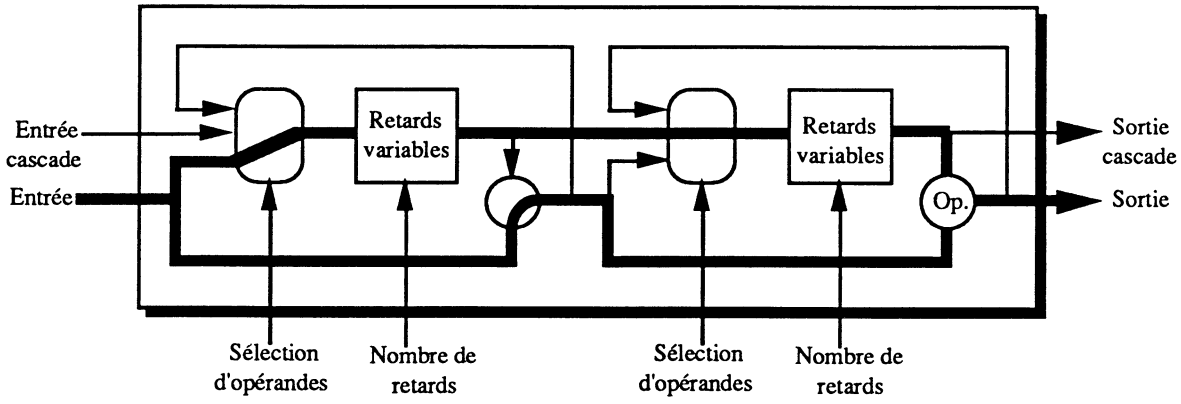


figure 3.17 :
Exemple de deux cellules cascadées pour obtenir un retard plus grand

Pour obtenir la majorité des algorithmes décrits dans le paragraphe 3.2, il faut pouvoir réaliser, en plus des opérations de voisinage, des opérations entre deux images et des opérations entre une image et une constante.

Notre cellule de base comporte déjà une unité arithmétique capable de réaliser certaines de ces opérations de base entre deux opérandes :

- addition, soustraction
- minimum, maximum
- sélection de l'opérande A ou B

Pour réaliser des opérations entre deux images ou entre une image et une constante, il suffit donc de disposer, au niveau de la cellule de calcul, d'un ou deux chemins de données supplémentaires pour pouvoir aiguiller vers l'unité arithmétique le second opérande. Ceci nous conduit à améliorer encore notre cellule générale de la façon suivante (figure 3.18) :

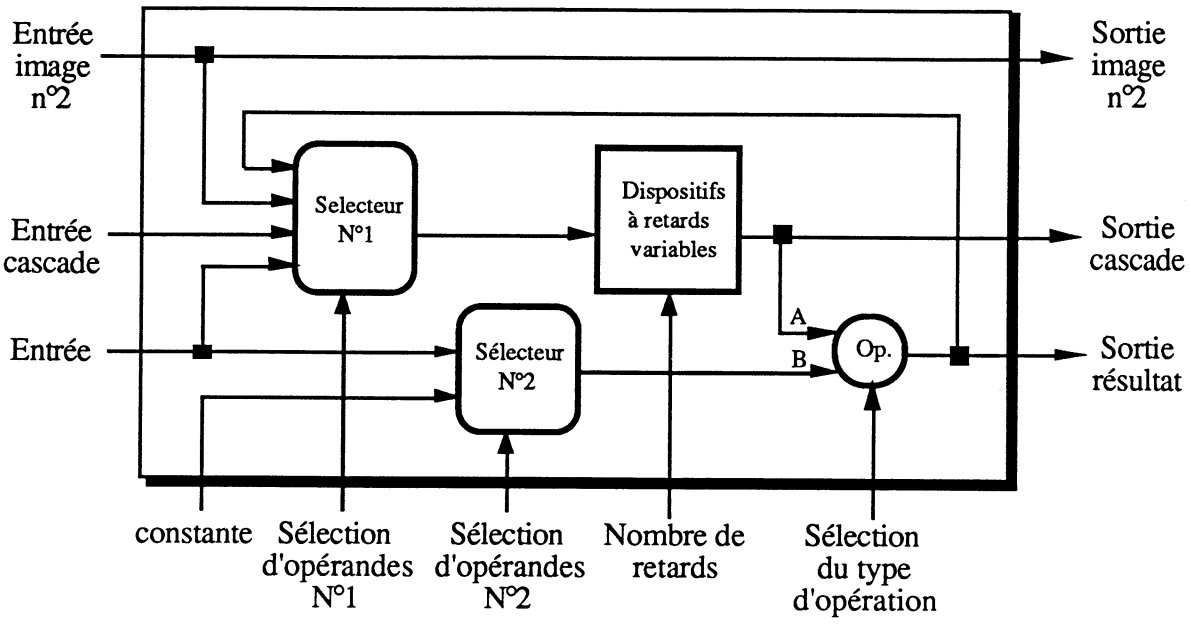


figure 3.18 : Architecture de la cellule, 2ème amélioration

Pour réaliser l'ensemble des algorithmes de traitement d'images bas niveau que nous avons décrit dans le paragraphe 3.2, il nous reste à modifier notre cellule pour qu'elle puisse réaliser des opérations de marquage, c'est à dire modifier l'information d'évènements associés à chaque donnée traitée (pixel).

Dans la grande majorité des cas, cette opération de marquage dépend du résultat d'une opération de type minimum ou maximum entre deux opérandes (deux images ou une image et une constante). Le résultat de la comparaison doit pouvoir, si on le désire, modifier (passage de 0 à 1) l'indicateur d'évènements. La cellule doit donc disposer de deux chemins de données 1 bit supplémentaires pour transmettre cette information et d'un opérateur booléen pour générer cette dernière. Nous obtenons donc la version finale de la cellule de calcul (figure 3.19).

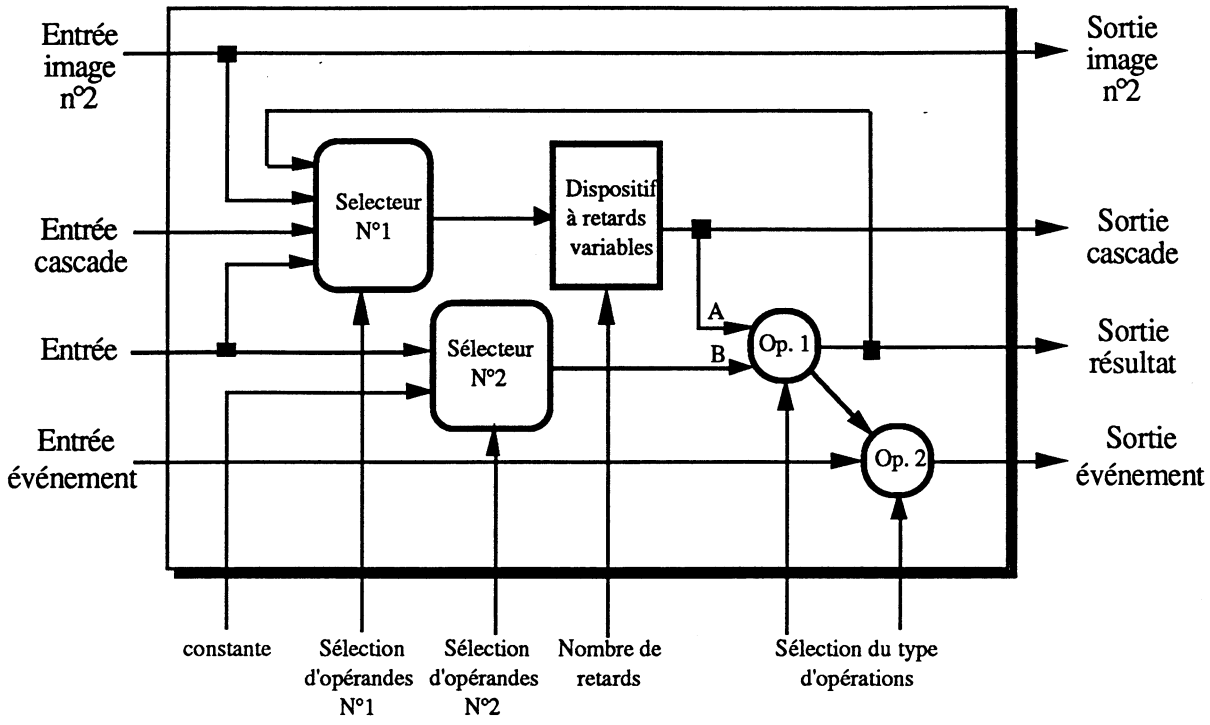


figure 3.19 : Architecture de la cellule finale

Partant de la cellule de base présentée en figure 3.15, nous l'avons fait progressivement évoluer vers cette cellule de calcul finale composée :

- de deux sélecteurs de données;
- d'un dispositif à retard de longueur programmable;
- d'une unité arithmétique sur des données de type pixel entier 16 bits (OP.1);
- d'une unité booléenne sur des données 1 bit (événement) (OP.2).

Il est ainsi possible de réaliser tous les algorithmes précédemment décrits (cf. paragraphe 3.2) à l'aide d'une cascade de ces cellules élémentaires dont tous les blocs fonctionnels sont programmés de manière appropriée.

3.4) Architecture du circuit

Nous avons défini, dans le paragraphe précédent, des méthodes de calcul spécifiques pour réaliser des opérations de voisinage (convolutions, morphologie mathématique) ou des opérations arithmétiques entre deux images ainsi qu'une cellule de traitement élémentaire capable de les mettre en oeuvre. Nous souhaitons maintenant définir un circuit intégré incluant une ou plusieurs cellules décrites dans la figure 3.19 et les circuits annexes associés. Cet opérateur intégré sera adapté à l'architecture de notre coprocesseur de traitement d'images.

3.4.1) Architecture générale

La structure générale de cet opérateur est composée des quatre blocs fonctionnels suivants interconnectés entre eux (cf. figure 3.20) :

- une unité d'interface en entrée;
- une unité de calcul parallèle intégrant plusieurs cellules élémentaires;
- une unité d'interface en sortie;
- une unité de contrôle et de programmation.

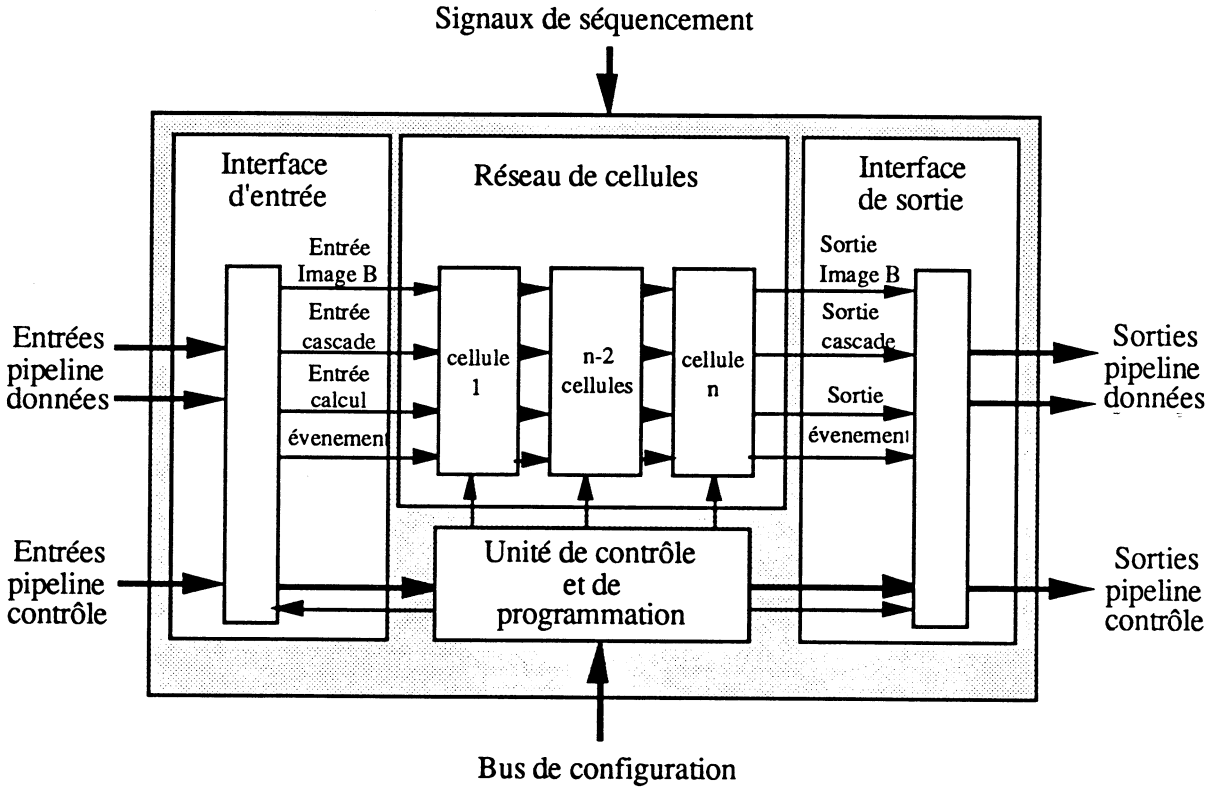


figure 3.20 : Synoptique général de l'opérateur

3.4.2) Interfaces externes

3.4.2.1) Interface d'entrée

L'interface d'entrée a pour rôle d'acquérir tous les signaux circulant en pipe-line et de sélectionner les chemins de données pipe-line qui doivent être connectés sur les entrées du réseau de cellules de calcul.

Les cellules disposent de deux entrées : entrée A, entrée B. Il doit être possible de connecter n'importe quelle entrée externe sur l'entrée calcul du réseau. Cette sélection est statique et est configurée pour toute la durée du traitement d'une séquence.

La structure de l'interface d'entrée se résume à un multiplexeur deux entrées vers une sortie et de verrous de mémorisation sur les chemins de données pipeline (cf. figure 3.21). La configuration de l'interface d'entrée est réalisée via un bit de programmation issu de l'unité de contrôle et de programmation.

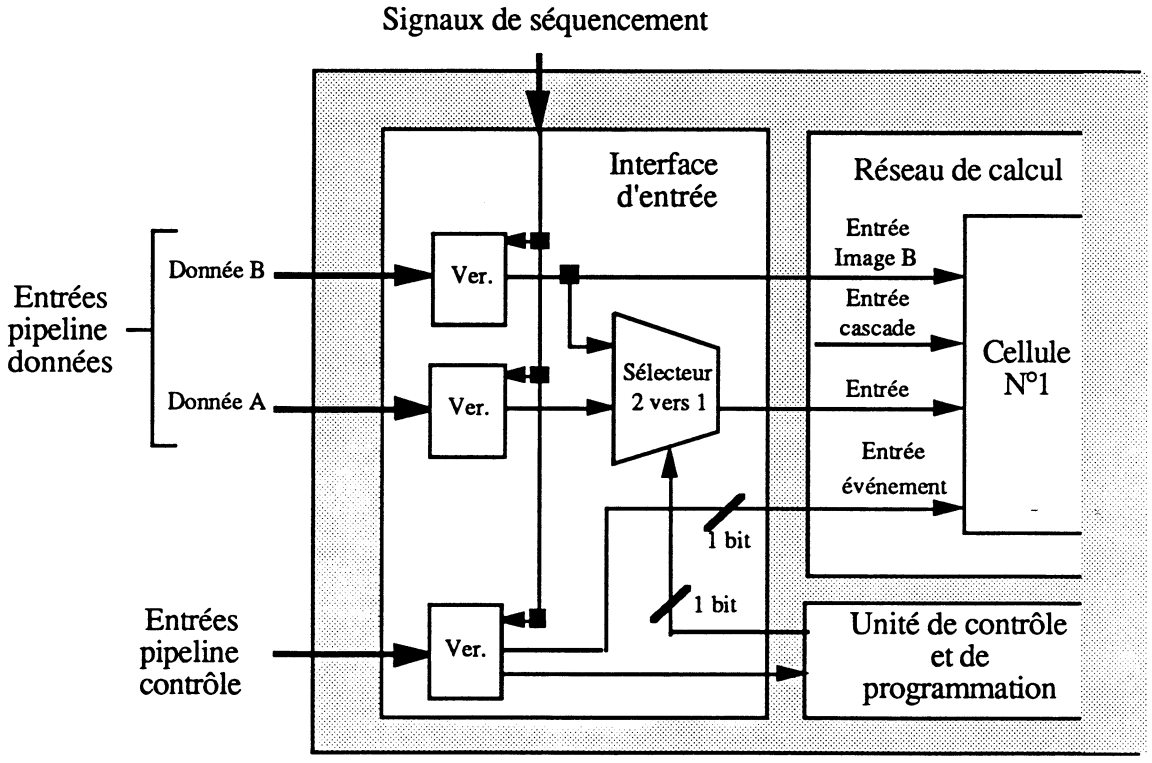


figure 3.21 : Synoptique général de l'interface d'entrée

3.4.2.2) Interface de sortie

L'interface de sortie a pour rôle de sélectionner les informations (données et contrôle) qui doivent être émises sur les chemins de sortie pipe-line de l'opérateur. Les cellules disposent de trois sorties : sortie A, sortie B et sortie cascadibilité, seules les deux premières seront émises sur les sorties de l'opérateur. La structure de l'interface de sortie est très simple et se résume à quatre verrous de mémorisation sur les chemins de données et de contrôle (cf. figure 3.22).

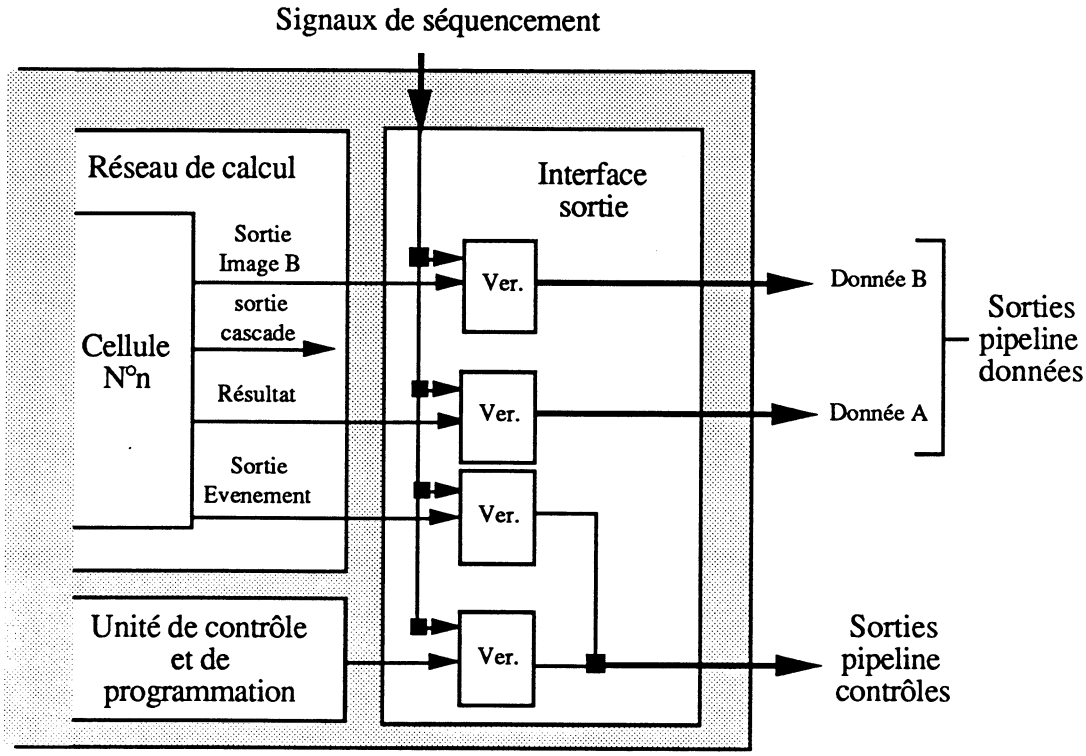


figure 3.22 : Synoptique général de l'interface de sortie

3.4.3) Réseau systolique adopté

Le réseau de cellules de calcul que nous avons choisi pour l'implantation est basée sur une association en cascade d'une version améliorée de la cellule finale décrite en figure 3.19.

Nous avons rajouté au niveau de chaque cellule, la possibilité de réduire (diviser par une puissance de 2) la dynamique de représentation de l'une des données. Nous avons donc intégré une unité de décalage arithmétique à droite ("arithmetic right shifter") sur les chemins de données internes (cf. figure 3.23).

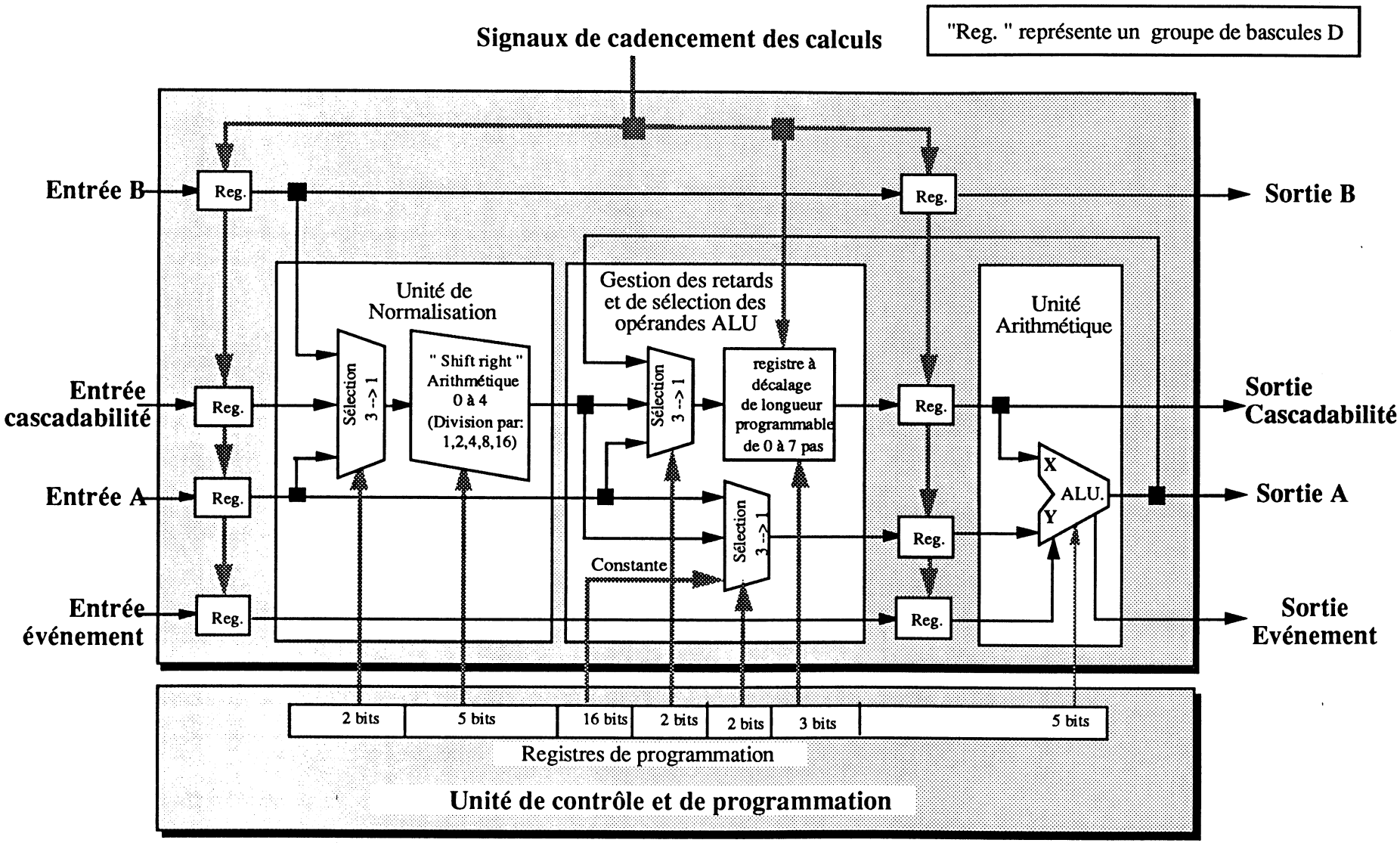


Figure 3.23 : Structure de la cellule de calcul finale

Le réseau de calcul est basé sur une cascade de cellules identiques, interconnectées linéairement et son interface externe (entrées - sorties pipe-line) est identique, quel que soit le nombre de cellules qu'il comporte. Sa programmation est réalisée via le bus d'accès micro-processeur et chaque cellule a ses propres registres de configuration .

3.4.4) Structure d'une cellule du réseau

Chaque cellule se compose des trois blocs fonctionnels suivants, cascadés linéairement, qui permettent de réaliser l'aiguillage des données circulant en pipe-line et les calculs :

- une unité de normalisation;
- une unité de gestion des retards et de sélection des opérandes de l'unité arithmétique;
- une unité arithmétique.

La configuration de tous les blocs fonctionnels d'une cellule est réalisée au moyen d'un registre de programmation de 35 bits de large.

L'unité de normalisation permet d'appliquer à l'une des données circulant sur l'une des trois entrées de la cellule, une division par une puissance de 2 (de 2^0 à 2^4).

L'unité de gestion des retards et de sélection des opérandes ALU permet :

- la sélection de l'opérande d'entrée de l'unité arithmétique qui sera retardé;
- la génération des retards (de 0 à 7 retards);
- la sélection du second opérande de l'unité arithmétique.

L'unité arithmétique travaille à partir de trois opérandes d'entrée X, Y, Entrée événement et génère deux informations résultats :

- un résultat qui sera émis sur la sortie "sortie A" de la cellule;
- une information booléene émise sur la sortie événement.

Cette unité arithmétique est capable, suivant sa programmation, de réaliser l'une des vingt opérations arithmétiques suivantes :

- sortie A = X + Y
- sortie A = X - Y
- sortie A = Y - X

- sortie A = Minimum (X, Y)
- sortie A = Maximum (X, Y)

- sortie A = (0 si $X \geq Y$) et (1 si $X < Y$) (seuillage si $Y > X$)
- sortie A = (1 si $X \geq Y$) et (0 si $X < Y$) (seuillage si $Y < X$)
- sortie A = (X si $X \geq Y$) et (0 si $X < Y$)
- sortie A = (0 si $X \geq Y$) et (X si $X < Y$)
- sortie A = Minimum (X, Y) et Sortie événement = 1 si ($X < Y$)
- sortie A = Minimum (X, Y) et Sortie événement = 1 si ($X \geq Y$)
- sortie A = Maximum (X, Y) et Sortie événement = 1 si ($X < Y$)
- sortie A = Maximum (X, Y) et Sortie événement = 1 si ($X \geq Y$)
- sortie A = X et Sortie événement = 1 si ($X = Y$)
- sortie A = X et Sortie événement = 1 si ($X \neq Y$)
- sortie A = Y et Sortie événement = 1 si ($X = Y$)
- sortie A = Y et Sortie événement = 1 si ($X \neq Y$)

- sortie A = X (opération utilisée pour le test et pour la cascadabilité pour des retards plus grands (cf. figure 3.17))
- sortie A = Y (opération utilisée pour le test et pour la cascadabilité pour des retards plus grands (cf. figure 3.17)))
- sortie A = 0

La cellule engendre deux retards de propagation entre l'entrée des données à traiter au temps t_n dans la cellule et la sortie du résultat associé t_{n+2}.

3.4.5) Unité de contrôle

L'unité de contrôle du circuit a deux fonctions :

- permettre la programmation des différents blocs fonctionnels du circuit par le microprocesseur hôte;
- gérer la propagation des informations de contrôle circulant en pipe-line à l'intérieur du circuit.

Cette unité est extrêmement simple, la propagation des informations de contrôle est réalisée par un dispositif générant (2n) + 2 retards de propagation si le circuit comporte n cellules, et un port d'entrées - sorties 8 bit permettant de transmettre, via un registre à décalage, les informations de configuration aux différentes unités du circuit.

3.4.6) Implantation des algorithmes dans cette architecture

Pour configurer cet opérateur parallèle afin qu'il réalise un algorithme de traitement particulier, il est nécessaire :

- de représenter cet algorithme sous la forme de graphes de circulation de données;
- de déterminer les différents retards qui doivent être appliqués sur les chemins de données pour que, pour chaque opérateur, les opérandes correspondants soient disponibles simultanément sur les entrées de la cellule;
- de déterminer, à partir du nombre de retards et d'opérations élémentaires à réaliser, le nombre de cellules nécessaires pour les calculs;
- de configurer judicieusement les différents blocs fonctionnels de ces cellules afin de réaliser la ou les fonctions souhaitées.

Cette approche est indispensable pour une mise en oeuvre optimale des algorithmes dans cette structure, mais elle nécessite une très bonne connaissance de la structure des graphes de calcul et de la structure de l'opérateur. Elle s'adresse donc à un groupe d'utilisateurs avertis.

Nous avons donc essayé de permettre la programmation de l'opérateur à un niveau supérieur beaucoup plus accessible pour un utilisateur non averti. Celle-ci est basée sur des bibliothèques prédéfinies correspondant à des séquences de traitement souvent usitées, que l'utilisateur peut sélectionner sans se préoccuper des détails de configuration de l'opérateur. Pour cela, un travail important a été réalisé au niveau de la parallélisation d'algorithmes dans cet opérateur et ces résultats sont disponibles sous la forme d'un logiciel (CEA88) qui, à partir de choix dans des menus de traitement, donne directement à l'utilisateur le nombre de cellules de calcul nécessaires et la configuration de celles-ci.

3.5) Implantation actuelle : INP20

Nous avons eu la chance de pouvoir réaliser, sous forme d'un circuit intégré, certaines des idées que nous avons développées au cours de cette étude. Ce circuit dénommé INP20 n'intègre pas toutes les fonctionnalités présentées dans les paragraphes précédents, mais démontre la faisabilité de cet opérateur en utilisant des techniques d'intégration de type compilateur de silicium.

Ce circuit a été réalisé en utilisant la technologie microélectronique CMOS deux microns de la société ES2, son implantation et son test ont été développés à l'aide de l'outil SOLO1000 (ES2). La totalité du circuit a été décrite dans le language de description HDL (Hardware Description Language) et de fait, nous n'avons pas de représentation schématique de celui-ci que nous pourrions intégrer dans ce rapport. L'implantation et la génération des vecteurs de test de ce circuit ont représenté environ 5 mois de travail. La première réalisation intégrée de ce circuit a été la bonne et celui-ci est actuellement commercialisé par le CEA.

3.5.1) Architecture interne

Le circuit intègre six cellules élémentaires identiques dont l'architecture fonctionnelle est une version réduite de la cellule décrite dans la figure 3.23. Chaque cellule, mis à part la dernière cellule du circuit, réalise des opérations sur des données codées sous la forme d'entiers 12 bits signés en complément à deux. La dernière cellule permet de l'arithmétique sur des entiers représentés sur 16 bits.

Le circuit a deux entrées A et B sur 12 bits, une seule sortie sur 16 bits et transmet uniquement en pipe-line des informations de contrôle sur 1 bit. Il ne correspond pas, de fait, à l'ensemble des caractéristiques que nous avons définies pour les opérateurs de notre coprocesseur de traitement d'images.

Le circuit représente 48000 transistors sur une puce de 81 mm^2 encapsulée dans un boîtier céramique 84 broches (cf figure 3.24). Sa fréquence maximale de fonctionnement est de l'ordre de 25 MHz pour l'horloge de calcul, ce qui est bien supérieur à la fréquence à laquelle il est possible de lui fournir les données.

3.5.2) Test du circuit

Ce circuit peut être vu de l'extérieur comme un immense registre à décalage (30 étages de 12 bits de large) transmettant les entrées, via des opérations internes, sur les sorties. Cette structure, qui est typique des systèmes à propagation d'informations, simplifie la testabilité du dispositif. Comme chacun des blocs fonctionnels du circuit peut être rendu transparent, il est possible de tester simplement chacun de ces blocs par une configuration appropriée des autres. Cette particularité nous a permis de tester le circuit en utilisant un nombre réduit de vecteurs de test (850 vecteurs pour tester 95% de fonctionnalité). Il est possible, par cette méthode de test, d'accéder à 99,8% des noeuds internes du circuit.

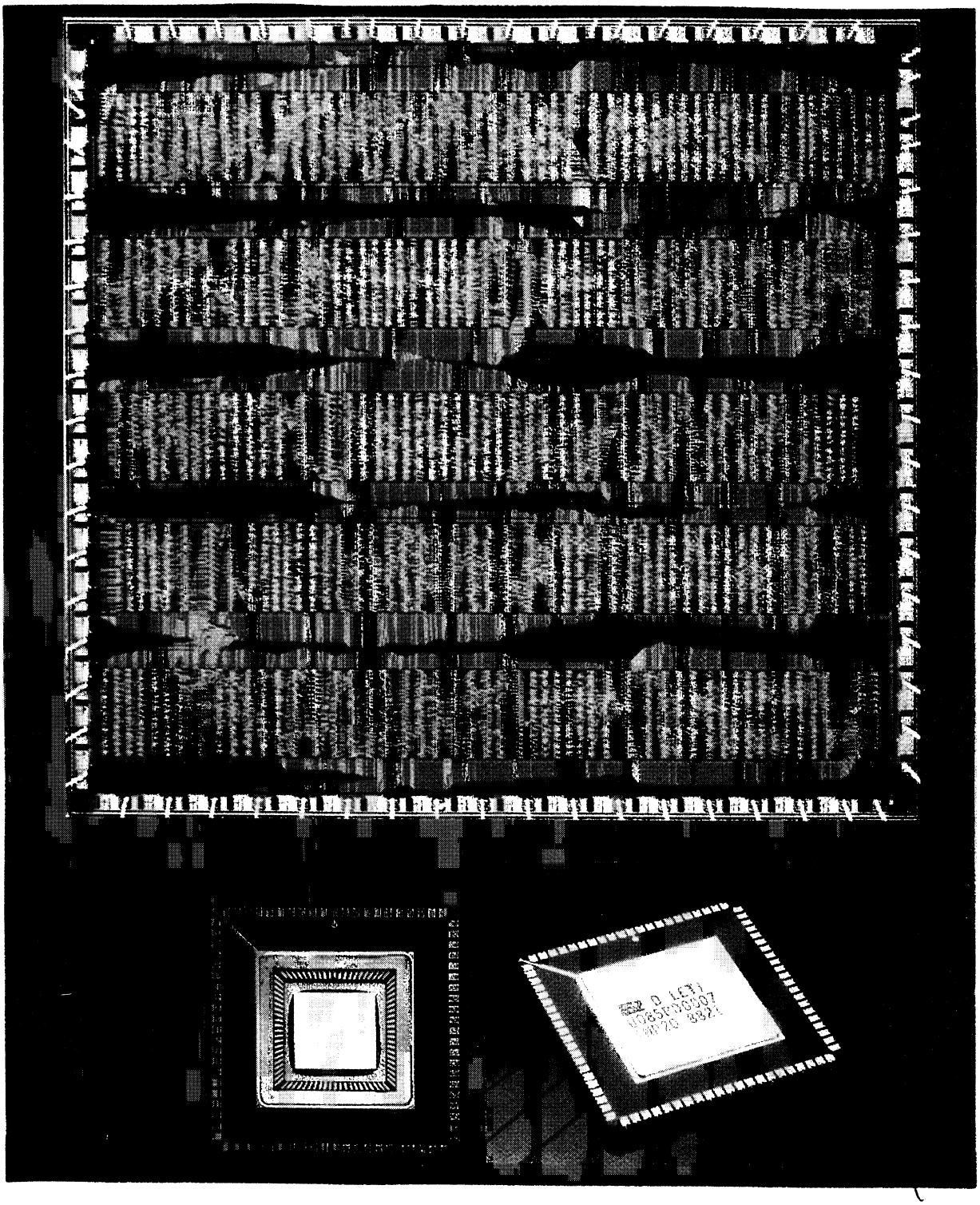


Figure 3.24 : Photographies de la puce et du boîtier de INP20

3.6) Conclusion

Nous avons dans ce chapitre, étudié l'architecture interne et externe d'un processeur spécialisé capable de réaliser des tâches de traitement d'images de bas niveau telles que : opérations entre deux images, convolutions, morphologie mathématique.

Nous regrettons de n'avoir pu intégrer dans un circuit toutes les idées présentées dans ce chapitre. La première intégration de ce circuit qui a eu pour nom INP20 ("Image Neighbourhood Processor") a eu surtout pour but de démontrer la faisabilité et la simplicité de cette approche méthodologique, ainsi que de mettre en valeur les performances de cette approche. La première utilisation de ce circuit dans le système monocarte OP2 (THE88) réalisée à la DLETI nous a permis d'atteindre les temps de traitement suivants pour des algorithmes typiques du traitement d'images industriel appliqués à des images de 512 x 512 pixels :

- convolution 9 x 9 séparable	**42 ms**
- ouverture morphologique en gris de taille 9 x 9	**63 ms**
- opération entre deux images	**21 ms**

Nous ne savons pas encore, si une nouvelle version de ce circuit sera réalisée dans le futur mais celle-ci permet déjà d'atteindre, pour les algorithmes cités ci-dessus, des performances quasi identiques à celles proposées par le système IP200 (TAK88), qui intègre trois circuits ISP2 beaucoup plus complexes qu'INP20.

Si une version améliorée de ce circuit peut être réalisée, nous pensons que la priorité est de simplifier et accélérer encore la mise en oeuvre des algorithmes existants en disposant, au niveau de ce futur circuit, de dispositifs à retard de plus grande taille permettant de stocker une ou plusieurs lignes de l'image et, de fait, de pouvoir réaliser en un seul passage des traitements bidimensionnels séparables.

La technologie dont nous disposions pour notre première réalisation ne permettait pas d'intégrer ces dispositifs à retard de grande taille, car il n'existait pas, au niveau de la bibliothèque de notre compilateur silicium, de blocs fonctionnels de mémoire statique qui représentent la manière optimale de réaliser les dispositifs en question. Cependant, même avec des technologies microélectroniques évoluées, il ne nous semble pas économiquement raisonnable d'intégrer ces dispositifs à retard de grandes tailles au niveau du circuit, il nous semble plus judicieux d'utiliser des mémoires statiques externes pilotées de manière appropriée.

Quatrième partie

Etude d'un processeur de traitement d'images spécialisé dédié aux opérations d'extraction d'informations

IFP "Image Features Processor"

Sommaire de la quatrième partie

4.5.2) Architecture de l'opérateur numéro 6

4.5.3) Stratégie pour le test

4.6) Conclusion

4.1) Introduction

Une image sous forme d'un tableau de pixels, même si elle a déjà subi un traitement pour mettre en évidence certaines de ses caractéristiques (exemples: les contours), représente et contient une quantité d'informations sans commune mesure avec celles que l'on désire réellement extraire. Certaines étapes de traitement (les tâches de moyen niveau) ont donc pour but de synthétiser, à partir de l'image ou de régions de celle ci, des informations (mesures) directement exploitables par des algorithmes réalisés par le microprocesseur local. Si ces dernières années, le marché des composants a vu déboucher de nombreux travaux de recherches sur les circuits dédiés à la réalisation d'opérations de voisinage sur des images, les problèmes d'extraction de mesures ont pour le moment nettement moins de succès : à notre connaissance, seul deux circuits très récents annoncés par LSI Logic réalisent certains d'entre eux (DOL89 : histogramme, transformée de Hough), (LSI89 : calcul de mesures sur des contours).

Ce chapitre présente une architecture matérielle intégrable capable de réaliser suivant sa programmation des tâches de bas niveau (transformation d'image par transcodage) et des tâches d'extraction de mesures de l'image (histogramme, projection, ...). Ses structures externe (communications entre circuits) et interne (séquencement des calculs) satisfont aux contraintes générales définies au niveau de l'architecture de "l'accélérateur de traitement d'image". Cette présentation est organisée en quatre parties. Dans la première nous décrivons l'architecture externe du circuit et une première décomposition de sa structure interne en unités fonctionnelles. La seconde partie présente les tâches que nous désirons réaliser et décrit les ressources de calcul qu'il est nécessaire d'intégrer dans l'unité de traitement pour leur mise en oeuvre. La troisième partie est consacrée à l'étude des autres unités fonctionnelles qui composent le circuit. La dernière partie décrit les choix qui ont été faits pour son intégration.

La dénomination actuelle de ce circuit est I.F.P. "Image Features Processor", certains des blocs fonctionnels qui le composent ont été implantés et testés, mais il n'a pas fait encore l'objet d'une réalisation matérielle intégrée.

4.2) Architecture macroscopique

L'architecture de ce circuit et les tâches qu'il est capable de réaliser sont beaucoup plus hétérogènes que pour le précédent circuit INP. Le but de ce paragraphe est d'introduire brièvement un certain nombre de points qui permettent d'avoir rapidement une vision globale de ses fonctionnalités. Il présente macroscopiquement sa structure externe, la nature des tâches qu'il est capable de réaliser, et introduit une première décomposition de sa structure interne en unités fonctionnelles.

4.2.1) Généralités

4.2.1.1) Informations circulant en pipe-line

Le circuit traite des pixels disponibles séquentiellement. Pour chaque pixel traité (un cycle de calcul du pipe-line) trois groupes d'informations (données, contrôle, cascadabilité) circulent en parallèle et en pipe-line sur les entrées et sorties du circuit (fig. 4.1). Le format de ces communications, pour les informations données et contrôle, respecte les protocoles définis pour les dialogues entre les différents circuits de "l'accélérateur de traitement d'image" (cf. paragraphe 2.4). L'information de cascadabilité est une information supplémentaire qui est dédiée à l'association en pipe-line de plusieurs circuits IFP.

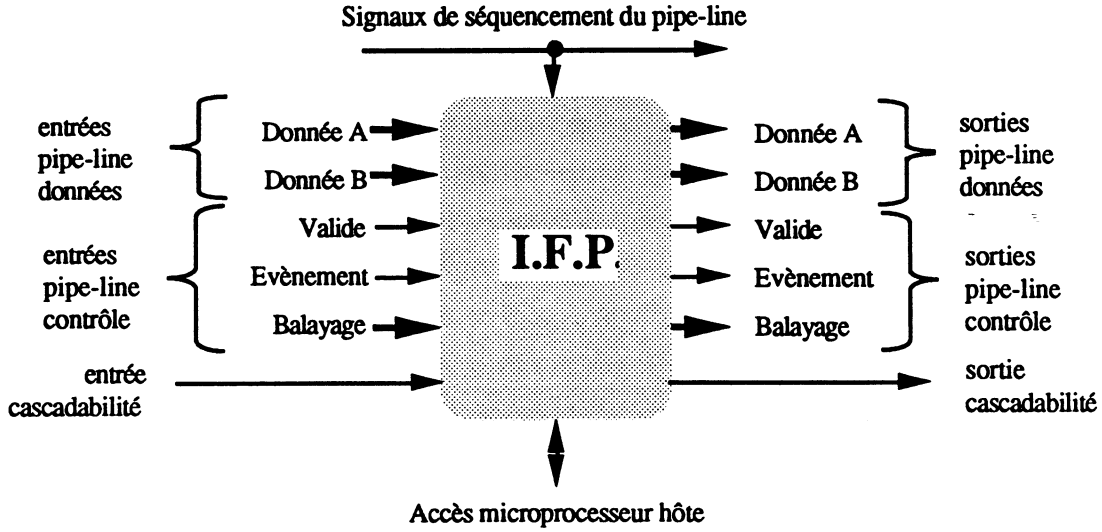


Figure 4.1 : IFP, communications externes

La nature et le format des informations lues sur les entrées, leurs influences sur les différentes tâches de traitement réalisées par le circuit sont décrits ci-dessous :

Informations de type données :

Donnée A : (entier 16 bits, codé en complément à 2) valeur du pixel de l'image à traiter, il peut appartenir à l'image d'origine ou être le résultat d'une phase antérieure de traitement pipe-line . Des dispositifs internes au circuit permettent si on le désire de réduire sa dynamique de représentation avant traitement (Division par 2^n, n = 1, 2 ou 3)

Donnée B : (entier 16 bits codé en complément à 2) cette entrée de donnée transmet lorsqu'elle est utilisée, le numéro de la région à laquelle appartient le pixel. Elle a aussi pour rôle de transmettre certaines informations globales durant les phases de réinitialisation du pipe-line en cours de traitement (cf. paragraphe 2.4).

Informations de contrôle et de cascadabilité :

Cascadabilité : (booléen, report de retenue) permet d'associer deux ou plusieurs circuits pour calculer des mesures résultats avec une dynamique étendue.

Valide : (booléen) il indique si le pixel doit ou non être pris en compte dans les opérations de calcul de mesures.

Evènement : (booléen) il indique si le pixel doit ou non être pris en compte lors des opérations d'extraction de listes.

Balayage : (codé sur 4 bits) il décrit les modes de balayage de l'image (ordre d'accès aux pixels) et permet de calculer pour chaque pixel sa position dans l'image (Indice ligne, Indice colonne).

Accès microprocesseur :

L'accès microprocesseur ne dépend pas du fonctionnement pipe-line, il permet de configurer le circuit (programmation) et d'accéder aux résultats (mesures), lorsque tous les pixels ont été traités.

4.2.1.2) Nature des tâches réalisées

Ce paragraphe décrit la manière dont les tâches de traitement d'image, que nous désirons réaliser par ce circuit, s'intègrent dans le pipeline de calcul de l'accélérateur de traitement. La présentation détaillée de leurs applications et de leurs structures algorithmiques sera faite dans le

paragraphe 4.3. Ce circuit est capable de réaliser trois types de tâches pipe-line :

- transformation d'une séquence de pixels en une séquence de pixels résultat;
- élaboration de mesures à partir d'une séquence de pixels;
- transformation d'une séquence de pixels en une séquence d'informations de position.

Ces différentes tâches de traitements mettent en oeuvre la valeur, la région et/ou la position dans l'image de chacun des pixels de l'image à traiter. L'algorithme de traitement pour un pixel ne fait pas intervenir :

- les pixels qui ont déjà été traités,
- l'ordre dans lequel les pixels sont traités (mode de balayage de l'image).

Toutes ces tâches de traitement peuvent être décrites sous la forme d'un même processus élémentaire de traitement (cf. fig.4.2) qui, à chaque cycle de calcul pipe-line (traitement d'un pixel), réalise séquentiellement les trois sous-tâches suivantes:

- lecture des informations (données, contrôle) disponibles sur les entrées;

- exécution d'un algorithme de traitement utilisant les informations lues sur les entrées et une seule information issue d'une mémoire locale à accés aléatoire qui mémorise des données devant être disponibles en local pour toute la durée de la tâche de traitement (N cycles de calcul pipeline);

- écriture d'informations (données, contrôle) sur les différentes sorties du circuit.

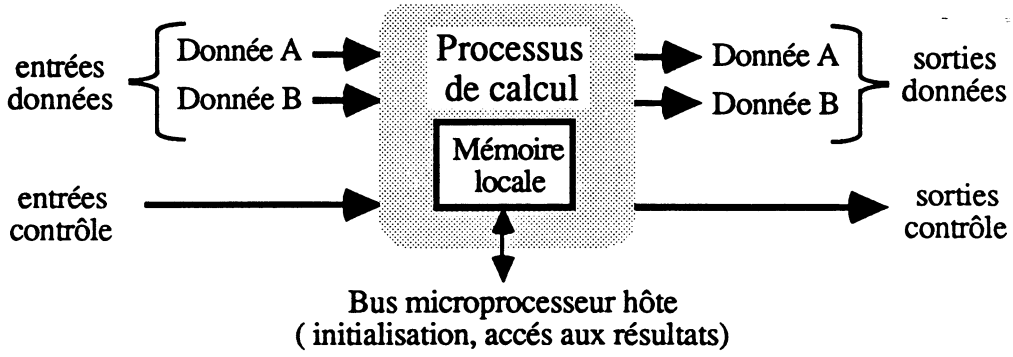


Figure 4.2 : IFP, structure générale des processus de traitement

Ces processus utilisent, mais ne modifient pas, les informations circulant sur les chemins de contrôle. Ils peuvent être classés en trois groupes suivant leur action sur les informations circulant sur les chemins de données.

** Processus de transcodage (LUT, génération dynamique de régions)*

Ils créent ou ils modifient l'une des informations circulant sur les chemins de donnée pipe-line en fonction de tables de constantes mémorisées dans la mémoire locale (fig. 4.3)

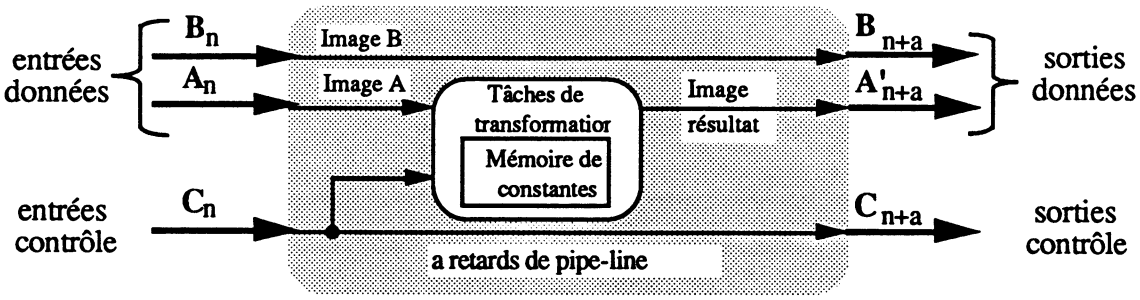


Figure 4.3 : Structure des processus de transcodage (exemple pour donnée B)

** Processus d'extraction de mesures (Histogramme, Projection, ...)*

Ils ne modifient pas les informations circulant sur les chemins de donnée pipe-line; ils calculent, à partir des paramètres lus sur les entrées, des mesures résultats qui sont disponibles à la fin du calcul dans la mémoire locale (fig.4.4) et ils peuvent être associés en cascade pour augmenter la dynamique de calcul des mesures résultats (fig.4.5).

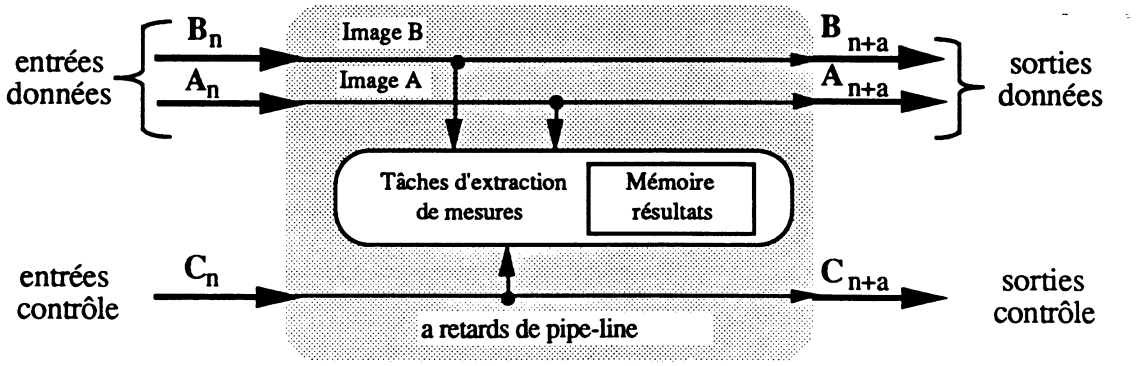


Figure 4.4 : Structure des processus d'extraction de mesures

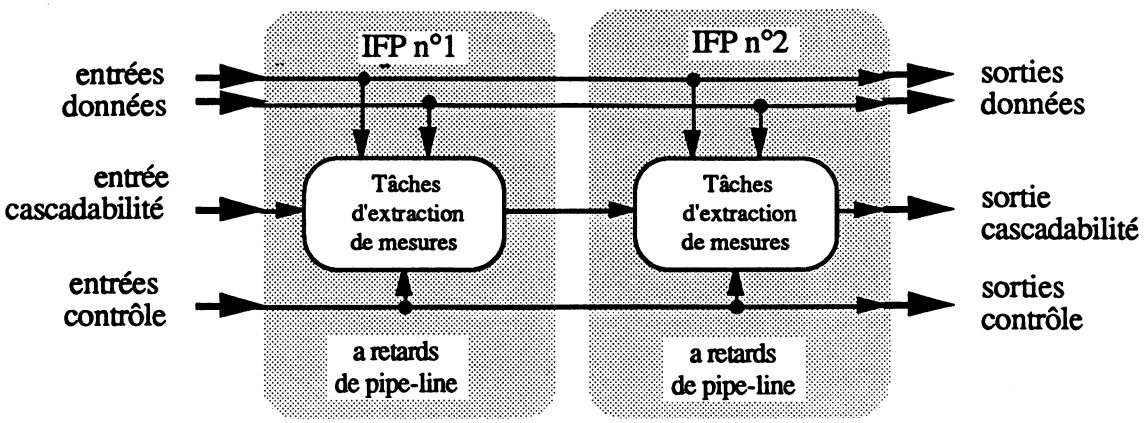


Figure 4.5 : extension de la dynamique de calcul des mesures résultats

* *Processus d'extraction de listes ("Run Code", ...)*

Ils réalisent simultanément deux tâches (fig.4.6) :

- modification des informations circulant sur les chemins de données pipe-line, les données A et B peuvent par exemple être remplacées par la position du pixel dans l'image (indice ligne, indice colonne);

- élaboration d'informations sur la structure de la liste (ex. nombre d'éléments par ligne de pixels) qui sont mémorisées dans la mémoire locale pour stocker les résultats.

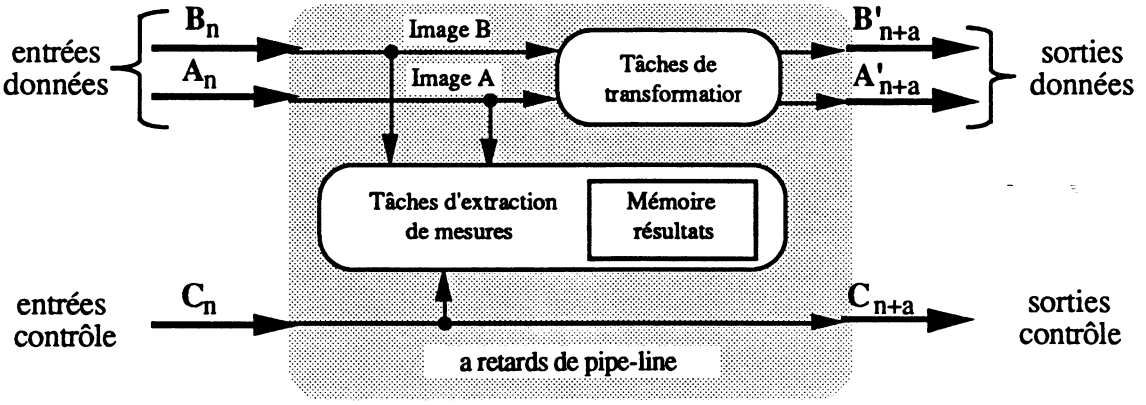


Figure 4.6: Structure des processus d'extraction de listes

4.2.1.3) L'extraction de mesures sur les régions

Certaines procédures d'extraction de mesures ne calculent pas des informations à partir de la totalité des pixels de l'image mais seulement de certains d'entre eux (pixels appartenant à une même région). Il est aussi fréquent d'extraire une même mesure pour chacunes des différentes régions qui composent l'image. Cette notion de régions a en général les provenances suivantes :

- une connaissance à priori de la nature de l'image (régions d'intérêt), de la position d'un objet ou des défauts que l'on désire mettre en évidence (BAT85, SAN86, SAN87);

- une segmentation de l'image en différentes zones pouvant être basée sur des critères statistiques (LOW84);

- un prétraitement appliqué à l'image, tel que l'étiquetage des composantes connexes (components labeling), dans ce cas l'information de région associée à chaque pixel est le numéro du connexe auquel il appartient (DAV80, WEI87, BAT85).

Dans le cas de notre structure, où l'on traite un pixel tous les cycles élémentaires de calcul pipeline, un pixel appartient au plus à une région. L'information région peut être disponible pour les calculs sous les formes suivantes :

- la région est une constante : tous les pixels à traiter appartiennent à une même région;

- la région est une variable interne : il est possible de calculer, de manière interne au circuit, pour chaque pixel, la région qui lui est associée à partir de sa position (SAN86), c'est le cas de la projection suivant des droites parallèles (cf 4.3.2.3);

- la région est l'une des différentes informations disponibles en parallèle sur les entrées pour chaque pixel à traiter (Donnée B) : elle peut être issue d'un traitement réalisé dans un étage de calcul antérieur (exemple une génération dynamique de région cf. 4.3.1.2) ou appartenir à une image des régions qui est balayée en même temps que l'image à traiter.

Tous les algorithmes d'extraction de mesures qui seront détaillés dans la suite de cette étude sont formulés dans ce dernier cas, qui est le plus général.

4.2.2) Présentation générale de la structure interne

4.2.2.1) Décomposition en unités fonctionnelles

A partir de la description macroscopique des tâches de traitement d'images que nous désirons implanter dans ce circuit, nous pouvons définir une première approche de l'architecture interne du circuit, se décomposant en deux unités fonctionnelles cascadées (cf. Fig. 4.7) :

- Une unité de service qui réalise :
  * la sélection et le conditionnement des informations circulant sur les entrées pipe-line;
  * la génération des informations de position (indice ligne, indice colonne) à partir des informations de contrôle pipe-line.

- Une unité de traitement qui réalise :
  * les tâches de traitement d'images, en utilisant comme paramètres de calcul, les informations issues de l'unité de service;
  * la sélection des informations à émettre sur les sorties pipe-line.

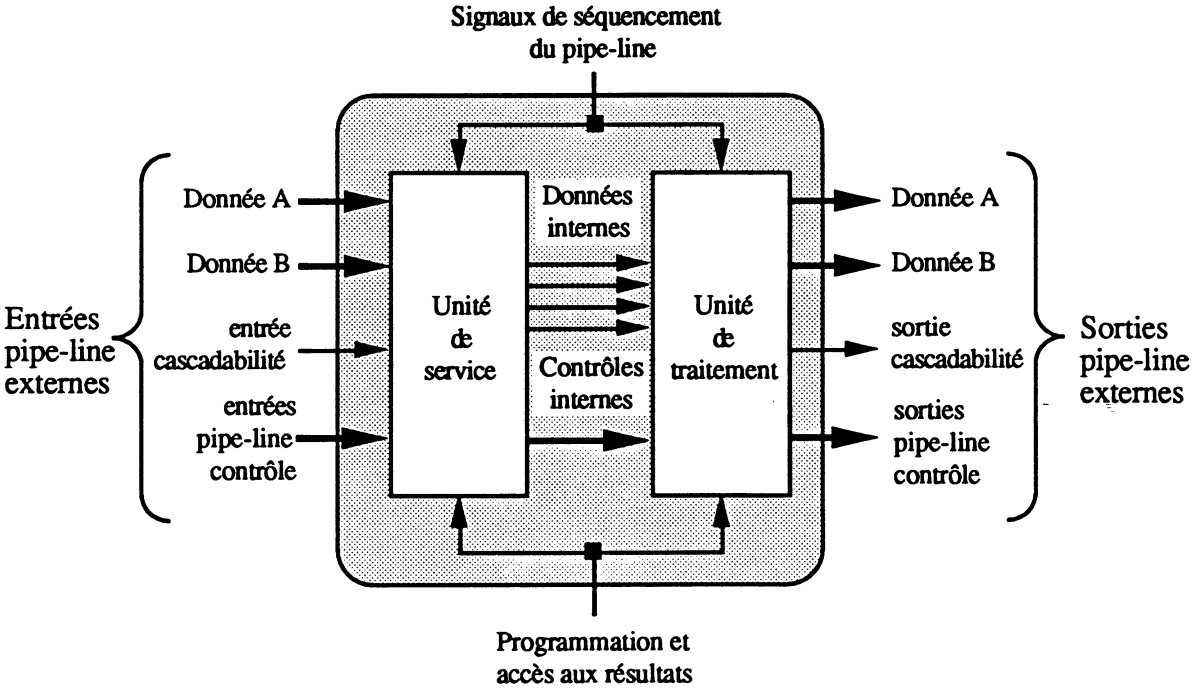


Figure 4.7 : Synoptique général du circuit IFP

4.2.2.2) Architecture des unités fonctionnelles

Il est nécessaire de détailler l'architecture des deux unités fonctionnelles (service, traitement) qui composent ce circuit et les tâches qui leurs sont allouées, afin de définir les échanges d'informations internes entre unités et les variables qui sont disponibles pour les algorithmes de traitement détaillés dans le paragraphe 4.3.

Unité de service

Elle reçoit les données disponibles sur les entrées pipe-line du circuit et génère toutes les variables nécessaires pour les calculs de l'unité de traitement (cf. fig. 4.8).

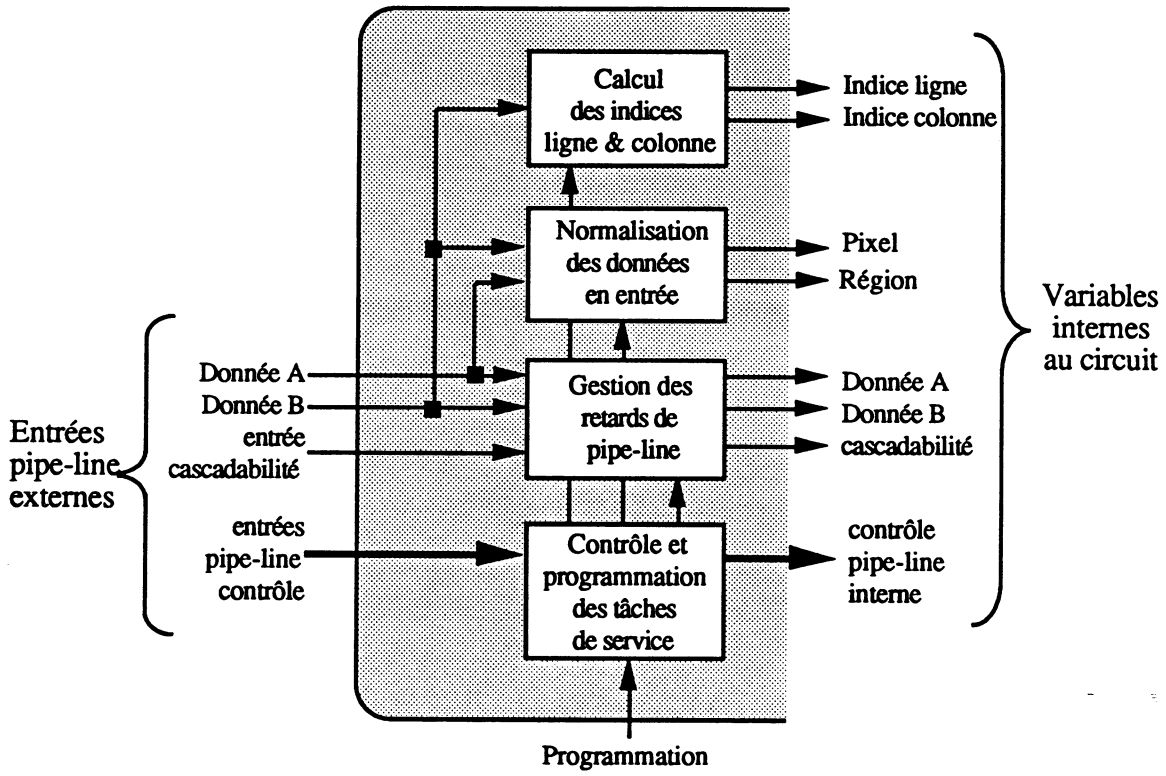


Figure 4.8 : Synoptique général de l'unité de service

Cette unité réalise simultanément trois tâches de service au moyen des trois sous-unités fonctionnelles suivantes, cadencées par une unité de contrôle centralisée :

- Une unité de calcul d'indice :

  Elle génère les informations sur la position du pixel (indice ligne, indice colonne) dans l'image à partir des informations disponibles sur les entrées pipeline, c'est-à-dire informations de contrôle ("Balayage" ordre d'accès aux pixels) et donnée B (pour les phases d'initialisation) (cf. paragraphe 2.3.2)

- Une unité de normalisation :

  Elle génère les variables internes "Pixel" (valeur du pixel) et "Région" (numéro de la région à laquelle appartient le pixel (cf. 4.2.1.3) à partir des données A et B disponibles sur les entrées pipeline du circuit, avec éventuellement une réduction de leur dynamique de représentation.

- Une unité de gestion de retards pipeline :

  Le rôle de cette unité est de conserver une circulation synchrone des données sur les chemins pipeline (DonnéeA, DonnéeB, Cascadabilité) et de compenser les retards engendrés par les unités de calcul d'indice et de normalisation (cf.paragraphe 2.4).

Unité de traitement

Elle reçoit de l'unité de service toutes les informations pipe-line nécessaires à son fonctionnement. Elle réalise simultanément deux tâches au moyen des deux sous-unités fonctionnelles suivantes, cadencées par une unité de contrôle centralisée (cf. fig 4.9) :

- Une unité de calcul :

  Elle réalise toutes les opérations de calcul arithmétique nécessaires à l'exécution de l'algorithme de traitement d'images et gère la mémoire locale de travail.

  Elle utilise comme paramètres d'entrée pour ses calculs, les informations internes suivantes :
  * Pixel (valeur du pixel)
  * Région (numéro de la région de l'image à laquelle appartient le pixel)
  * Indice ligne, Indice colonne (position du pixel dans l'image)
  * Valide, Evènement (informations de contrôle, cf. 4.2.1.1)
  * Cascadabilité (pour l'association en cascade de circuits, cf. 4.2.1.1)

Deux informations sont disponibles en sortie de cette unité de calcul :

* Cascadabilité (pour l'association en cascade de circuits)
* Résultat (utilisé lorqu'on réalise des processus de transcodage, cf. 4.2.1.2)

- Une unité de sélection des données en sortie :

Elle compense tous les retards pipe-line engendrés par l'unité de calcul, afin de conserver une circulation synchrone des données, et sélectionne les informations à émettre sur les sorties de données pipe-line externes du circuit.

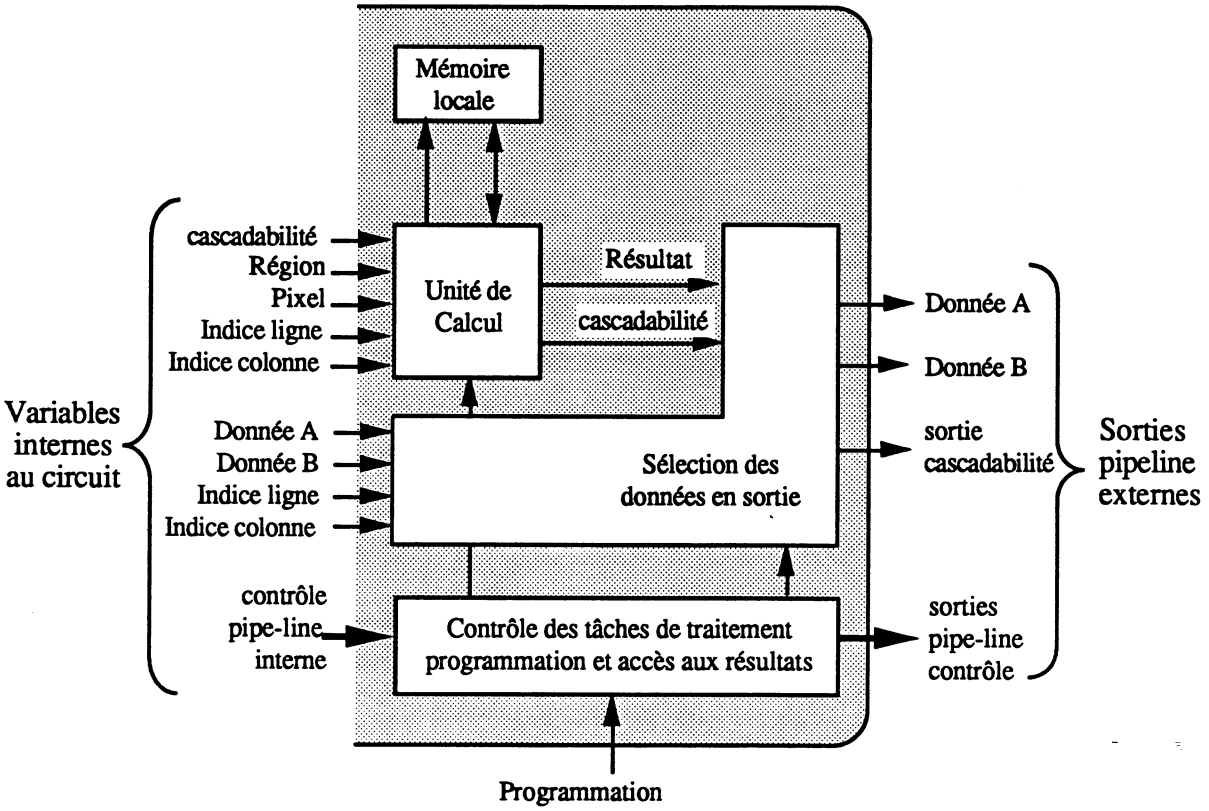


Figure 4.9 : Synoptique général de l'unité de traitement

4.3) Etude des processus de traitement

Ce paragraphe présente les différentes tâches de traitement d'images réalisées par le circuit, la structure de leurs algorithmes de calcul pour le traitement d'un pixel et l'étude des ressources matérielles (calcul, chemins de donnée, mémorisation) qu'il est nécessaire d'intégrer dans l'unité de traitement pour leur mise en oeuvre. La formulation algorithmique adoptée pour la majeure partie de ce paragraphe est apparentée au langage Pascal parce qu'elle est, dans ce cas, plus lisible qu'une description sous forme de graphe de flots de données.

4.3.1) Les transcodages

Ces traitements modifient (transformation par loi tabulée) ou créent (génération dynamique de régions) les informations circulants sur les chemins de donnée pipeline (A,B), les résultats sont émis sur l'un des chemins de donnée de sortie du circuit à destination d'un étage de traitement pipeline ultérieur.

4.3.1.1) Transformation par loi tabulée

Ces traitements correspondent aux transformations par loi tabulée (L.U.T . "Look Up Table" fréquemment utilisées en traitement d'image). La mémoire locale contient une loi de transformation chargée par le microprocesseur hôte durant la phase d'initialisation. Cette transformation peut être appliquée sur la valeur du pixel, sa région et dans certains cas sur sa position (indice ligne et colonne) (SAN86). Il est possible, en outre, d'appliquer différentes lois de transformation à une donnée, en fonction d'un second critère (exemple : une région). Le processus général de calcul pour chaque pixel traité est le suivant :

```
Résultat = Transcodage (Pixel, Région, Indice ligne, Indice colonne)

/* Informations stockées dans la mémoire locale */
Codage(k) : entier 16 bits /* table de transcodage */

/* Variables temporaires */
Indice      : entier 16 bits /* indice pour pointer dans la table de transcodage */
Résultat    : entier 16 bits /* paramètre résultat */

/* Constantes, chargées lors de la programmation */
a : entier 16 bits   /* bornes de représentation de X, - ((a/2)+1) < X < (a/2)+1 */
b : entier 16 bits   /* offset pour que Indice soit toujours positif (b = a/2) */

/*  la variable X peut être : Pixel ou Indice ligne ou  Indice colonne */

Début
Indice = a * Région + X + b
Résultat = Codage (Indice)
Retour    (Résultat)
```

4.3.1.2) Génération dynamique de régions

Dans certains cas, où l'on désire réaliser des opérations d'extraction de mesures sur des régions dont les formes et les positions sont connues à priori, il est possible en utilisant des transformations par lois tabulées, de calculer la région auquel appartient chacun des pixels à traiter, à partir de leurs positions dans l'image. La méthode que nous avons décidée d'implanter dans ce circuit s'applique uniquement pour des ***régions rectangulaires***, elle est dérivée d'une méthode beaucoup plus générale utilisée dans l'architecture PPPE "Parallel Pipeline Projection Engine" (SAN86,SAN87) qui permet de générer dynamiquement des régions de formes polygonales ou circulaires. La méthode utilisée pour des images de n lignes de m colonnes est la suivante :

Principe de base :

Les différentes régions que l'on désire définir dans l'image sont des bandes verticales de largeurs variables. L'information numéro de région peut être générée en utilisant une loi tabulée (fig.4.10) de m éléments qui est appliquée sur l'indice colonne (la loi contient pour chaque colonne, le numéro de la région qui lui est associé).

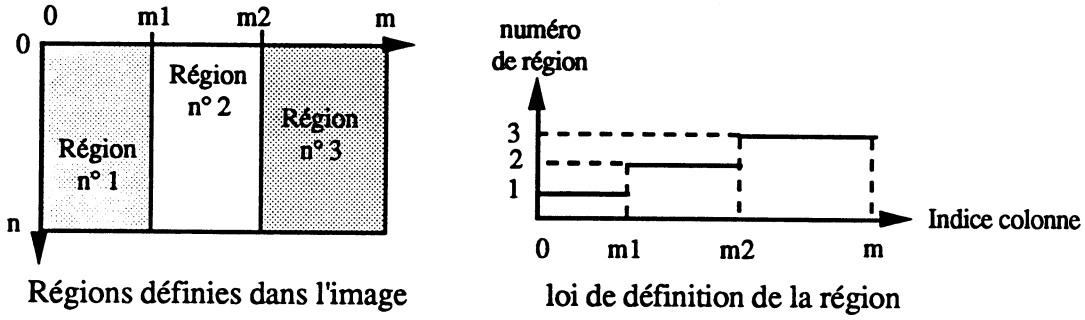


Figure 4.10 : définition des régions sous forme de bandes verticales

Généralisation aux régions rectangulaires quelconques

Cette méthode basée sur l'utilisation de lois tabulées peut être généralisée aux régions décomposables en rectangles de tailles quelconques, la génération de l'information numéro de région est dans ce cas réalisée en deux étapes en utilisant deux lois tabulées (fig.4.11) (deux circuits IFP) en cascade (fig.4.12).

1ère étape: le premier circuit définit en utilisant une loi tabulée de m éléments, les zones de l'image pour lesquelles la structure des régions suivant les lignes est identique;

2ème étape: le second circuit calcule le numéro de région à partir du numéro de zone et de l'indice colonne, sa mémoire locale contient pour chacune de ces différentes zones une loi tabulée définissant le numéro de région pour chacun des points de la ligne. Le nombre de régions est limité par la taille de la mémoire locale de ce second circuit.

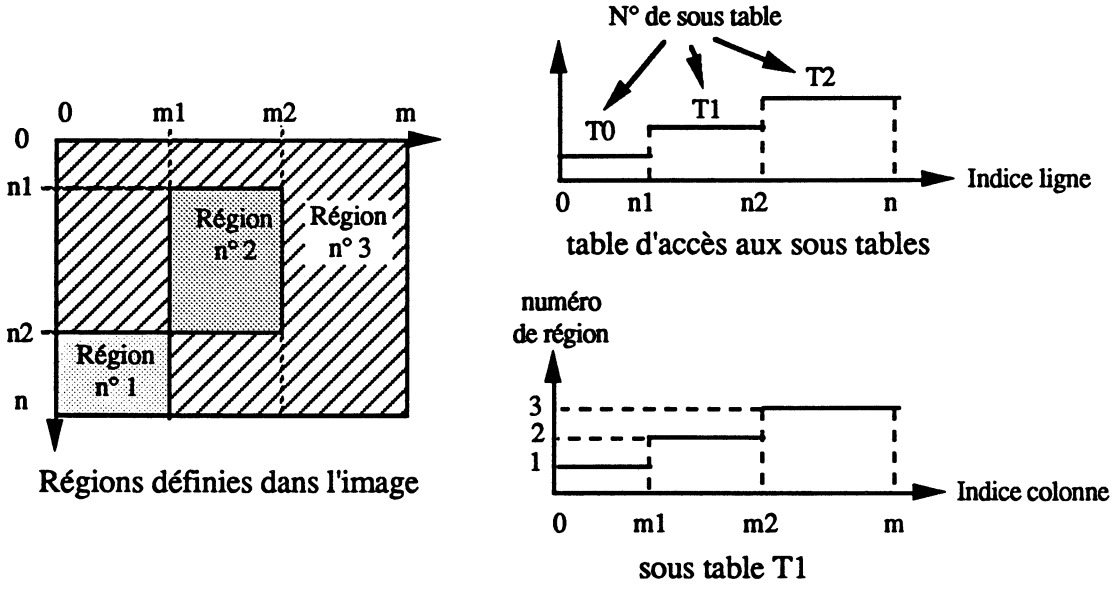


figure 4.11 : définition des régions de formes rectangulaires

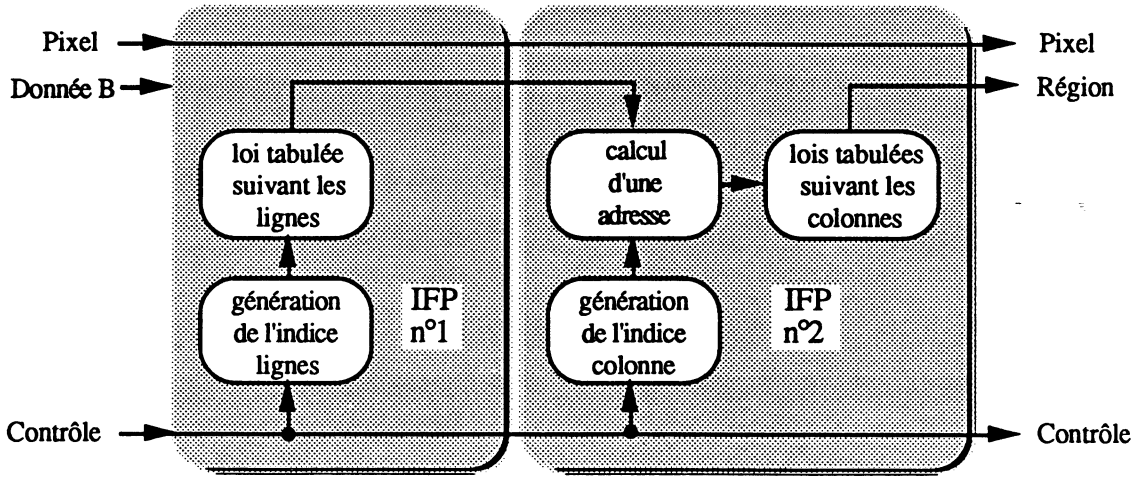


Figure 4.12 : génération des régions par deux circuits en cascade

4.3.2) L'extraction de mesures sur l'image

Les mesures résultats issues de ces processus de traitement sont disponibles dans la mémoire locale lorsque tous les pixels ont été traités (N cycles de calcul pipeline). Elles peuvent être organisées sous forme de scalaires, de vecteurs (ex: histogramme, projection) ou de tableaux (ex: Histogramme sur des régions).

4.3.2.1) Caractéristiques scalaires

Nous pouvons désigner sous ce terme une grande variété d'informations ponctuelles que l'on peut extraire de l'image ou des régions qui la composent. Nous pouvons citer les exemples classiques suivants qui utilisent pour le calcul de ces primitives la radiométrie du pixel et /ou sa position dans l'image:

- Mesures de nature statistique (BAT85, CAS84) :
  moyenne, variance, bornes supérieure et inférieure, moments, éléments de la matrice de covariance;
- Mesures sur des objets (régions) issus d'un étiquetage des composantes connexes :
  aire, périmètre, barycentre, rectangle englobant, matrice d'inertie, coordonnées du dernier point appartenant à l'objet. (DAV80, WEI87, PAU88).

La phase d'initiation préalable aux calculs de ces différentes mesures est en général l'initialisation à 0 des données résultats. Le processus élémentaire de calcul pour chaque pixel traité est le suivant :

```
Caractéristique (Pixel, Région, Valide, Indice ligne, Indice colonne)

/* Informations stockées dans la mémoire locale */
Résultat (k)        : entier 32 bits /* vecteur résultat */

/* Variables temporaires */
Indice              : entier 16 bits /* indice pour pointer dans le vecteur résultat  */
Contribution    : entier 32 bits /* information pour mettre à jour le vecteur résultat */

/* X = Pixel ou Indice ligne ou  Indice colonne */
/* Y = Pixel ou Indice ligne ou  Indice colonne */

/* la fonction F(a,b) peut être l'une des opérations suivantes (a*b, b) */
/* la fonction G(a,b) peut être l'une des opérations suivantes (a+b, min(a,b), max(a,b) ) */

Début
Indice = Région
Contribution = F(X,Y)
Si Valide = vrai alors Résultat (Indice) = G (Contribution, Résultat (Indice) )
Fin
```

4.3.2.2) Histogramme

C'est la description probabiliste au premier ordre (fonction de répartition) des occurences d'un certain attribut (ex: niveau de gris du pixel). L'étude de l'histogramme procure des indications sur la nature statistique de l'image, qui peuvent être utilisées pour des tâches de segmentation ou de classification (LOW83, LOW84). Les résultats sont structurées sous la forme d'un tableau à une ou plusieurs dimensions (ex: histogramme d'images multispectrales, histogramme pour chaque région) dont la taille dépend de la dynamique de représentation des attributs étudiés. La phase d'initialisation correspond à la mise à 0 du vecteur résultat, le processus de calcul pour chaque pixel traité est le suivant :

```
Histogramme (Pixel, Région, Valide)

/* Informations stockées dans la mémoire locale */
Résultat (k)        : entier 32 bits /* vecteur résultat */

/* Variables temporaires */
Indice              : entier 16 bits /* indice pour pointer dans la table de transcodage */
Contribution        : entier 32 bits /* information pour mettre à jour le vecteur résultat */

/* Constantes, chargées lors de la programmation */
a : entier 16 bits   /* dynamique de représentation de Pixel , -(a/2)-1 < Pixel < (a/2)+1 */
b : entier 16 bits   /* offset pour que le résultat du calcul soit toujours positif (b = a/2) */

Début
Indice = a * Région + Pixel + b
Si valide = vrai alors Résultat (Indice) = Résultat (Indice) + 1
Fin
```

4.3.2.3) Projections

La projection est apparentée aux opérations d'extraction de caractéristiques scalaires sur des régions. On extrait comme mesures, la somme des valeurs des pixels appartenant à chacunes des régions. Ces régions d'accumulation (Projection) correspondent, dans l'espace bidimentionnel discrétisé de l'image, aux équipotentielles définies par une équation analytique mettant en oeuvre les indices ligne et colonne (position du pixel).

Il est en théorie possible, pour une équation analytique quelconque, de calculer pour chaque pixel la région de projection, mais dans certains cas ce calcul devient excessivement complexe et semble difficilement réalisable de manière matérielle, on se limitera aux projections suivant des droites parallèles (fig.4.13). Dans ce cas, on peut approximer l'expression permettant de calculer la région de projection pour chaque pixel (SAN86, DOL89) de la manière suivante :

Région = (Indice ligne + Indice colonne * Tan (Θ) + b) pour (0° ≤ Θ < 45°)
Région = (Indice ligne * Cot (Θ) + Indice colonne + b) pour (45° ≤ Θ < 90°)
Région = (-Indice ligne * Cot (Θ) + Indice colonne + b) pour (90° ≤ Θ < 135°)
Région = (Indice ligne - Indice colonne * Tan (Θ) + b) pour (135° ≤ Θ < 180°)

Cette expression peut se généraliser sous la forme : ***Région = a * X + Y + b***

où les variables X et Y sont sélectionnées parmi [indice ligne, indice colonne], a est un réel compris dans l'intervalle (-1 < a < 1), b est une constante entière dont la valeur est déterminée pour que le résultat de ces équations soit toujours un ***entier positif*** (ce qui sous-entend que lors du calcul, on arrondit à l'entier le plus proche, le résultat d'un calcul réel).

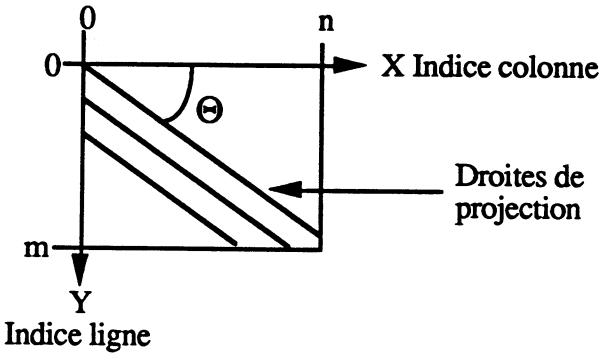


Figure 4.13 : orientation des droites de projection

La phase d'initialisation préalable au calcul est la mise à zéro du vecteur résultat, le processus de calcul pour chaque cycle pipeline est le suivant :

```
Projection (Pixel, Indice ligne, Indice colonne, Valide)

/* Informations stockées dans la mémoire locale */
Résultat (k)        : entier 32 bits /* vecteur résultat */

/* Variables temporaires */
Indice              : entier 16 bits /* indice pour pointer dans la table de transcodage */
Contribution        : entier 32 bits /* information pour mettre à jour le vecteur résultat */

/* Constantes, chargées lors de la programmation */
a : entier 16 bits /* a/n représente le terme Tan ou Cot */
n : entier 16 bits /* dynamique de représentation de a, -2^n < a < +2^n
b : entier 16 bits /* offset pour que le résultat du calcul soit toujours positif */

/* ( X=Indice ligne et Y=Indice colonne ) ou ( X=Indice ligne et Y=Indice colonne) */

Début
Indice       = ( (a*X) / 2^n ) + Y + b
Contribution =   Pixel
Si Valide = vrai alors Résultat (Indice) = Résultat (Indice) + Contribution
Fin
```

4.3.3) L'extraction de listes d'indices

4.3.3.1) Principe

Nous regroupons sous ce terme les tâches de traitement qui mémorisent des informations (résultats) pour certains pixels particuliers (dans notre cas les pixels marqués par l'information de contrôle "Evènement"). Les informations résultat issues de ces traitements sont organisées sous forme de liste dans laquelle les informations sont rangées en fonction de leur ordre d'apparition (ordre de balayage des pixels). Les cas les plus fréquemment rencontrés sont les suivants :

- liste des indices lignes et colonnes des points de contours (MOH87);
- liste des points de contours appartenant à une mire de positionnement (DER84);
- liste des indices lignes et colonnes des pixels présentant une configuration de voisinage particulière. L'exemple le plus classique est le codage "Run Code" (BAT85) : on extrait la position et la valeur des pixels correspondant à une transition (variation d'intensité) suivant les lignes horizontales; ce qui s'apparente à un codage de l'image sous la forme d'une suite de segments horizontaux d'intensité uniforme.

La taille de cette liste résultat n'est pas connue à priori, elle peut seulement être bornée par le nombre de pixels à traiter, pour cette raison nous avons décidé que la liste résultat sera stockée dans la mémoire de données image du module de traitement. La tâche réalisée par le circuit dans ce cas est décomposée en deux parties :

- calcul des données qui seront stockées dans la liste;
- calcul et mémorisation du nombre d'éléments de la liste.

4.3.3.2) Comptage d'évènements

Cet algorithme a pour but de compter pour chacune des lignes de l'image à traiter, le nombre de pixels marqués comme évènement . Ces informations peuvent être utilisées pour :

- réaliser des tests de convergence sur un traitement appliqué de manière itérative, l'exemple classique étant les opérations de squelétisation en morphologie mathématique (L'évènement marque les pixels dont la valeur différe entre deux étapes consécutives de traitement) (SER82);

- élaborer des informations qui permettrons de calculer des pointeurs définissant le début de chaque ligne dans des listes(MOH87).

Les données résultats doivent être préalablement initialisées à 0, le processus de calcul pour chaque pixel à traiter est le suivant :

```
Cnt.Evénement (Indice ligne, Evènement, Valide)

/* Informations stockées dans la mémoire locale */
Résultat (k)     : entier 32 bits /* vecteur résultat */

/* Variables temporaires */
Indice           : entier 16 bits /* indice pour pointer dans la table de transcodage */
Contribution     : entier 32 bits /* information pour mettre à jour le vecteur résultat */


Début
Indice = Indice ligne
Si Evènement = vrai alors Contribution = 1
Sinon Contribution = 0
Si valide = vrai alors Résultat (Indice) = Résultat(Indice) + Contribution
Fin
```

4.3.4) Caractéristiques communes aux différents processus de traitements

4.3.4.1) Structure algorithmique des calculs

Les différents tâches de traitement d'image que nous venons de décrire présentent toutes une structure algorithmique commune pour la réalisation de leurs calculs, celle ci peut se décrire sous la forme de quatre processus coopérants (fig. 4.14)

- un processus qui sélectionne, à partir des différents paramètres disponibles en entrée, deux couples de variables (X1, X2) (Y1,Y2) :

  * X1 : sélectionné parmi (Pixel, Région, Indice ligne, Indice Colonne)
  * X2 : sélectionné parmi (Pixel, Région, Indice ligne, Indice Colonne)
  * Y1 : sélectionné parmi (Pixel, Evènement, Indice ligne, Indice Colonne)
  * Y2 : sélectionné parmi (Pixel, Indice ligne, Indice Colonne)

- deux processus de calcul qui déterminent une adresse dans la mémoire locale (Indice) et une donnée (Contribution) à partir des couples de variables (X1, X2) et (Y1, Y2);

- un processus de calcul et de gestion de la mémoire locale qui réalise des opérations de type "lecture" ou "lecture modification écriture" à l'adresse "Indice" de la mémoire locale. Sa tâche de traitement est réalisée conditionnellement à la valeur de l'information de contrôle "Valide".

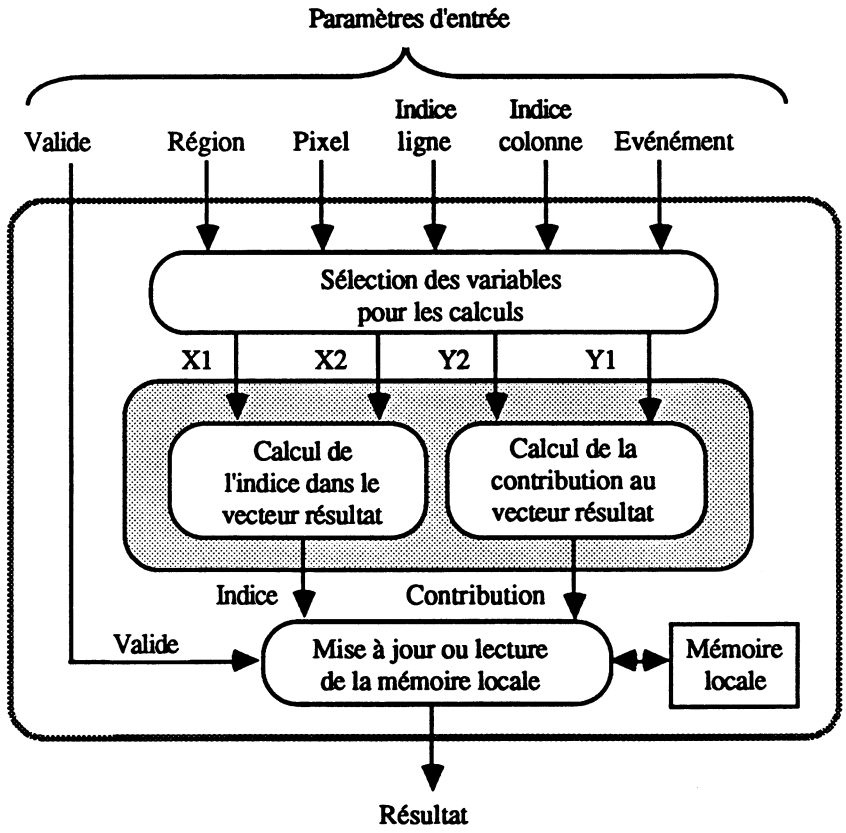


Fig 4.14 : Structure générale des processus de calcul

Pour tous les algorithmes précédemment cités, les deux sous tâches qui calculent les variables "Indice" et "Contribution" nécessitent au plus pour leur réalisation :

- 4 opérations arithmétiques (2 additions, 1 multiplication, 1 division par 2^n)
- 3 constantes entières (a,b,2^n) et les variables suivantes (Valide, Région, Pixel, Indice ligne, Indice colonne, Evénement)

Une étude détaillée, sur la minimisation du nombre d'opérations arithmétiques nécessaires à la réalisation des tâches de tous les algorithmes précédemment cités, conduit au graphe général de calcul de la figure 4.15, qui comporte cinq opérateurs arithmétiques dont la fonction est programmable, et deux opérateurs de sélection de données.

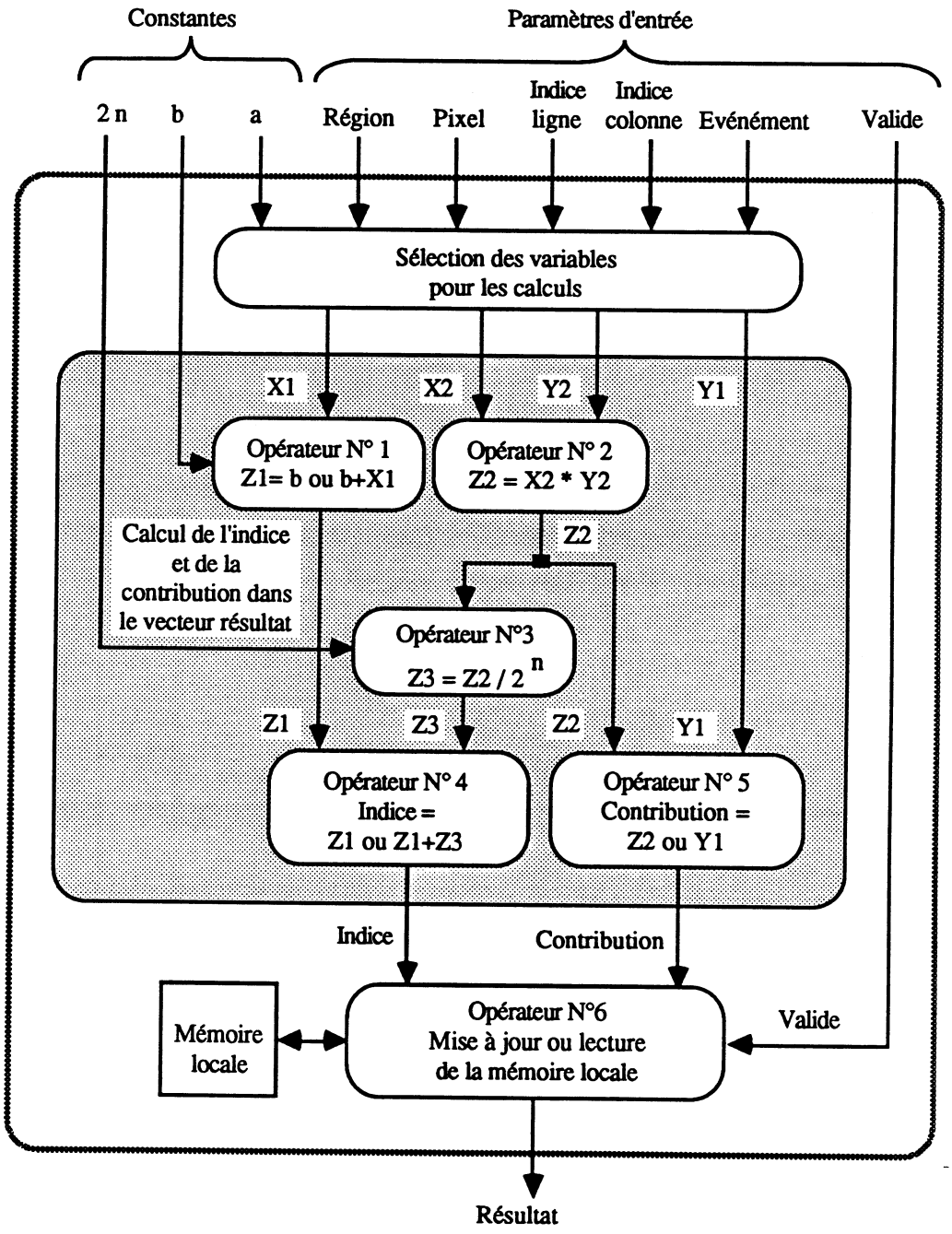


Figure 4.15 :
Structure de calcul commune aux algorithmes de traitement

4.3.4.2) Dynamique des calculs

Données de départ

Les informations externes (valeur du pixel, numéro de région, indice ligne, indice colonne), utilisées par les différentes tâches de traitement que réalise le circuit, sont des entiers 16 bits signés codés en complément à 2. Des dispositifs internes au circuit permettent, si on le désire de réduire avant traitement la dynamique de représentation de certaines de ces informations.

- La valeur du pixel peut être divisé par (4, 16, 64, 256) pour réduire par exemple le nombre de classes d'une opération de type histogramme.

- Le numéro de région peut être divisé par 2, cette possibilité est utilisée dans le cas où les régions représentent des objets, le bit le moins significatif de cette donnée différencie les pixels de l'objet et les pixels de son contour.

Données résultat

Les dynamiques de calcul des résultats des algorithmes de traitement bas niveau présentés dans le paragraphe 4.3. peuvent être les suivantes :

- Pour l'histogramme et les projections, les résultats sont codés au maximum sous la forme d'entiers 32 bits si la dynamique de représentation des pixels ne dépasse pas 16 bits et la taille de l'image n'excède pas 32 767 x 32 767 pixels (entiers 16 bits positif).

- Pour les caractéristiques scalaires, la dynamique des résultats ne dépasse pas 48 bits

- Pour tous les autres algorithmes, les résultats sont codés sous la forme d'entiers 16 bits.

4.3.4.3) Mémoire locale

Cette mémoire contient des informations qui doivent être disponibles en local durant toute la durée de la tâche de traitement (X cycles pipeline). Ces informations peuvent être des tables de constantes (transcodage) ou des variables et des informations résultats (extraction de mesures, listes). les principales caractéristiques de cette mémoire sont les suivantes :

- Pour un cycle de calcul du pipeline il y a **au plus une lecture ou une modification d'une seule case** de cette mémoire.

- Le microprocesseur hôte est capable d'accéder à cette mémoire en lecture pour l'accés aux résultats, et en écriture pour l'initialisation du contenu.

- L'accés à cette mémoire locale est aléatoire et ininterruptible par le microprocesseur hôte pour toute la durée du traitement (N cycles de calcul pipeline).

La mémoire est organisée en mots de 32 bits, elle peut contenir des informations sur 32 bits (mesures résultat) ou sur 16 bits (Constantes), dans ce dernier cas pour maximiser l'utilisation de cette mémoire un mot contient deux constantes.

Cette mémoire doit permettre de stocker des données structurées (vecteurs) comprenant jusqu'à 2^{16} éléments, c'est à dire 64 Kmots de 32 bits.

La technologie microélectronique à notre disposition (ES2, Cmos 2 et 1,5 micromètres) et la plupart des autres technologies existantes ne nous permettent pas d'envisager, dans un premier temps, l'intégration de cette mémoire dans le circuit, nous utiliserons donc des mémoires RAM statiques rapides du commerce.

4.4) Etude des processus de services

4.4.1) Processus de calcul des indices ligne et colonne

Le processus de calcul des indices ligne et colonne génère, à partir des informations disponibles sur l'entrée donnée pipe-line B et les informations de contrôle, la position géométrique dans l'image discrétisée du pixel en cours de traitement (cf. paragraphe 2.4.2.1).

Ce processus de calcul réalise l'initialisation, l'incrémentation ou la décrémentation de deux compteurs en fonction des informations de contrôle. Il n'est pas nécessaire de détailler sa structure, car celle-ci est extrêmement simple, ses différentes actions sont régies par le tableau présenté dans le paragraphe 2.4.2.

4.4.2) Processus de sélection des données en sortie

Ce processus aiguille sur les sorties du circuit les résultats de calcul et transmet les informations d'initialisation des calculateurs d'indice et de colonne. Suivant la nature de la tâche réalisée par le circuit, ce processus peut être organisé de trois manières :

- Le circuit extrait des mesures sur l'image, dans ce cas ce processus transmet sur les sorties du circuit les données disponibles sur les entrées de ce dernier.
- Le circuit réalise une opération de type transformation par loi tabulée, ce processus transmet dans ce cas le résultat de cette transformation sur le chemin de données A ou B, excepté pendant les phases d'initialisation.
- Le circuit réalise une opération d'extraction de liste, ce processus transmet dans ce cas les indices ligne et colonne des pixels dont le signal d'évènement est actif excepté pendant les phases d'initialisation.

4.5) Etude d'une architecture intégrable

4.5.1) Architecture de base

Nous n'avons pu, pour des raisons de temps, pousser jusqu'au bout, c'est à dire l'intégration, l'étude de ce circuit. Seuls certains des blocs fonctionnels qui le composent ont pu être testés et simulés. La technologie microélectronique dont nous disposions à l'époque (ES2, CMOS 2 microns) ne permettait pas sa réalisation car, en première estimation, l'intégration de ce circuit nécessite 83000 transistors, ce qui dépassait les possibilités de notre compilateur de silicium (SOLO1000, ES2) et de la technologie.

Nous avons seulement étudié la dernière unité de calcul du circuit, c'est à dire l'opérateur n° 6 de la figure 4.15. La tâche réalisée par cette unité est la lecture/modification/écriture des données disponibles dans la mémoire locale. Nous avons privilégié l'étude de cette unité car celle ci détermine la cadence de calcul maximum du circuit.

4.5.2) Architecture de l'opérateur n° 6

Pour réduire au maximum la durée du cycle de calcul de cette unité nous avons développé une méthode, qui permet de faire fonctionner simultanément le dispositif de calcul et le dispositif

réalisant l'interface avec la mémoire locale. Le principe que nous avons décidé d'utiliser s'apparente d'une part aux mécanismes de gestion des mémoires caches, et d'autre part aux méthodes dites de "scoreboarding" utilisées dans les dernières générations de microprocesseurs de type superscalaire. Il permet de réduire la durée du cycle de calcul de l'opérateur n° 6 pour réaliser la mise à jour des variables stockées dans la mémoire locale du circuit IFP. On passe d'un cycle ***lecture/modification/écriture*** à des cycles entrelacés de type ***lecture/écriture.***

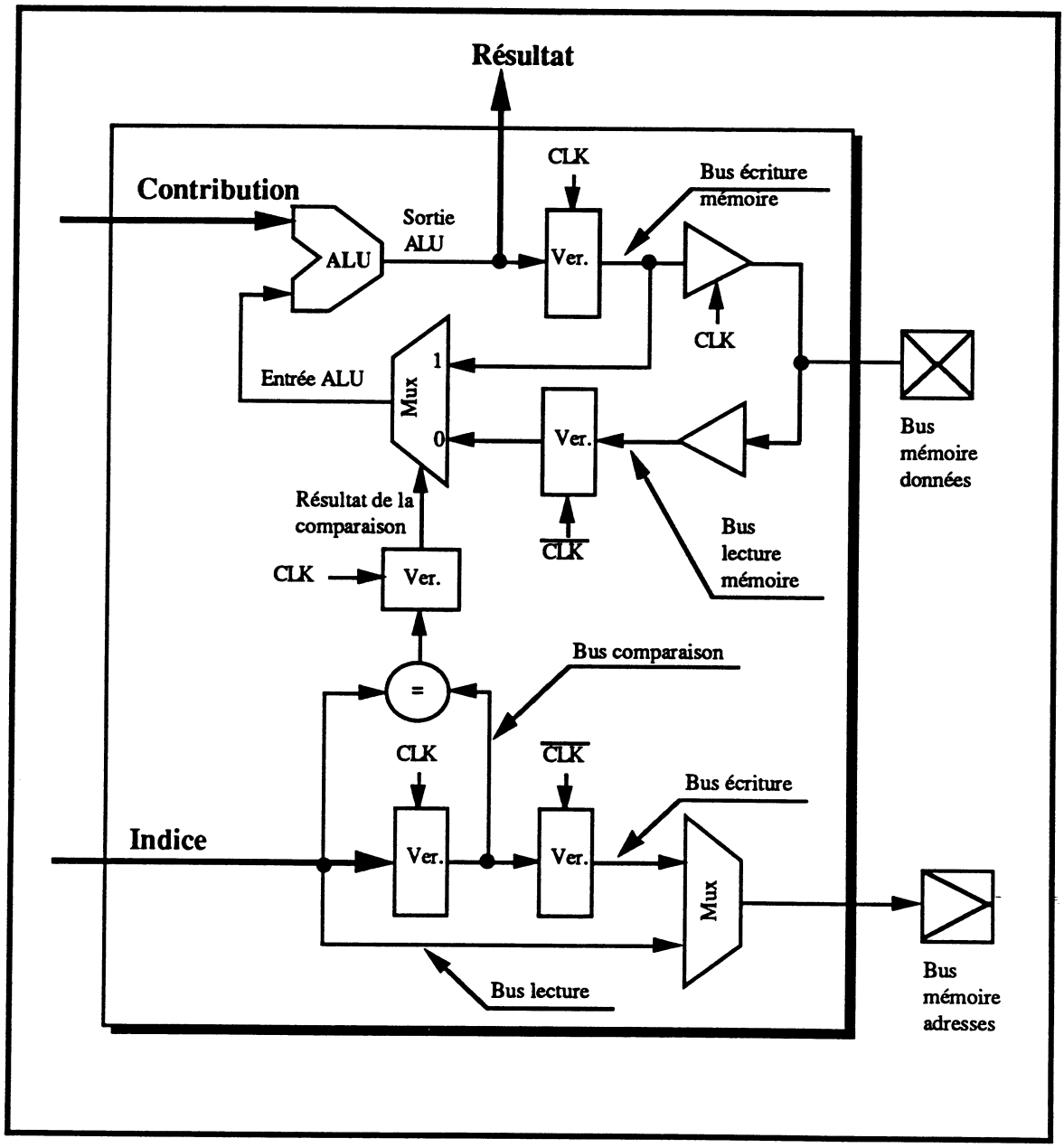


figure 4.16 : Structure de l'opérateur n° 6

La majorité des algorithmes de calcul de mesures sur des images sont basés sur des cycles de calcul de type lecture/modification/écriture pour leur interface avec la mémoire locale et, les références mémoire qui seront modifiées ne sont pas connues a priori. Dans ces conditions, il n'est pas possible de lire une donnée en mémoire pour une phase de calcul ultérieure tant que la phase en cours n'a pas réécrit la donnée, sinon il pourrait y avoir une incohérence des données en mémoire (modifications consécutives d'une même case mémoire).

Afin de réduire la durée du cycle lecture/modification/écriture, il est possible de lire la prochaine donnée à traiter avant la fin du cycle en cours si on peut détecter que deux phases de calcul consécutives font référence à la même case mémoire ou à des cases mémoire différentes. Si le second cas ne pose aucun problème, il faut dans le premier remplacer, à l'intérieur du circuit, la donnée lue par le résultat du calcul en cours pour conserver la cohérence du résultat.

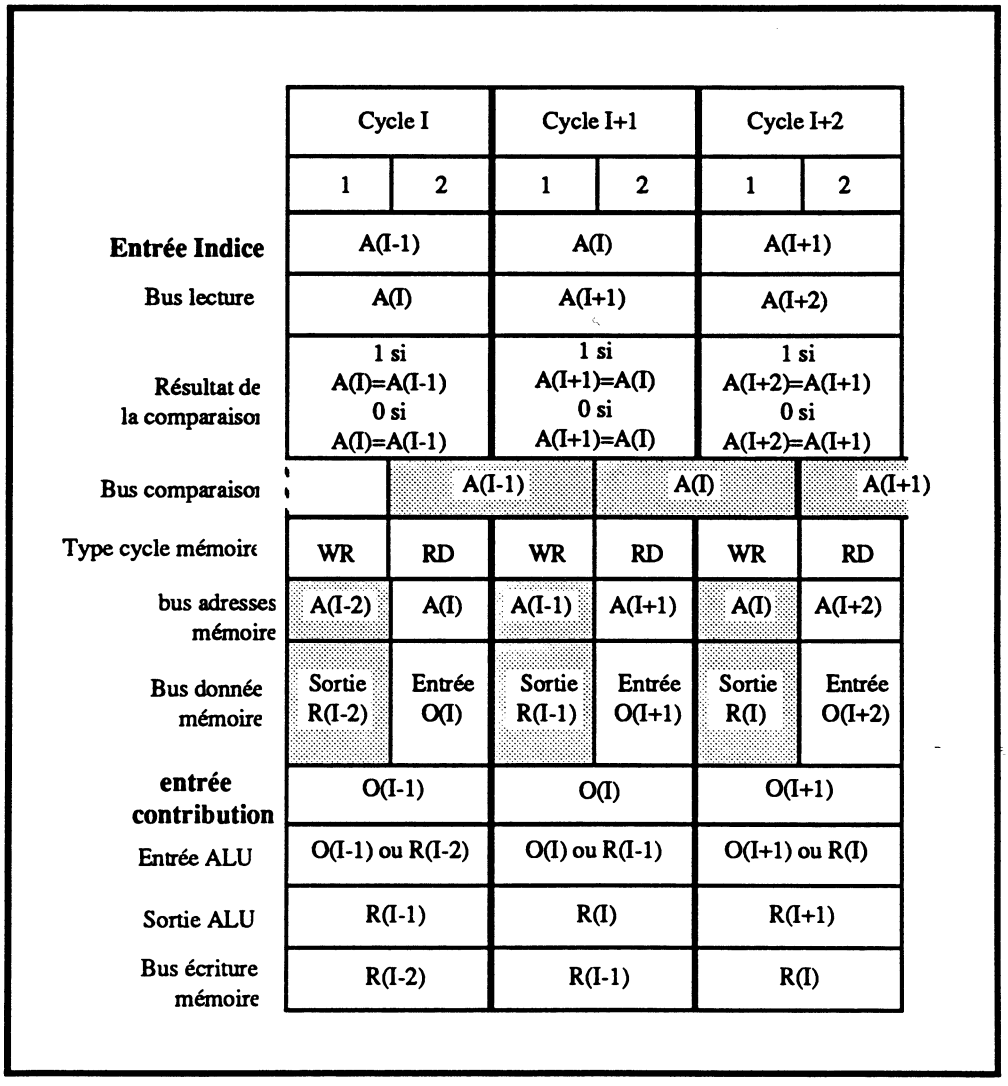


figure 4.17 : Séquencement des différents dispositifs de l'opérateur n° 6

Il est assez simple de réaliser ce dispositif de détection en utilisant un comparateur sur les adresses consécutives (cf. figure 4.16) et un dispositif de séquencement approprié (cf. figure 4.17).

4.5.3) Stratégie pour le test

La méthode que nous avons décidé d'adopter pour tester la version intégrée de ce circuit est identique à celle qui a été mise en oeuvre pour le test du circuit INP20 (cf. paragraphe 3.5.2). Si les structures et les possibilités de configuration des différentes unités qui composent ce circuit sont bien adaptées, il peut être vu de l'extérieur comme un immense registre à décalage transmettant les stimulis d'entrée, via des opérations internes, sur les sorties. Par une programmation appropriée, les différents opérateurs internes du circuit peuvent être rendus transparents et, il est possible de les tester séparément en utilisant une séquence de test par opérateur. Cette méthode simplifie considérablement le test du circuit et nous permet d'espérer un nombre très réduit de vecteurs de test.

4.6) Conclusion

L'objectif de ce chapitre était de définir un dispositif de calcul parallèle dédié aux tâches de traitement d'images de moyen niveau qui soit capable de s'intégrer dans notre architecture de coprocesseur. Dans ce but, nous avons déterminé un ensemble de mécanismes de calcul communs à différents types d'algorithmes : histogramme, projection, calcul de mesure scalaire sur des images, ... et défini un réseau de calcul parallèle capable de les mettre en oeuvre.

Nous regrettons fortement de n'avoir pu terminer l'étude de ce circuit et réalisé celui-ci sous forme d'un circuit intégré car nous ne disposions ni du temps, ni des outils de conception VLSI nécessaires à son intégration (en première approximation sa réalisation intégrée nécessiterait 80000 transistors).

Ce dispositif de traitement est tout à fait complémentaire du circuit INP20 et à tous deux, ils pourraient constituer un dispositif de calcul capable de réaliser rapidement un grand nombre d'algorithmes de traitement d'images utilisés en vision industrielle. Nous considérons, même, qu'il est la clé de cette architecture et qu'il aurait dû être réalisé en premier, car pour certaines applications simples, il peut être à lui seul, un coprocesseur viable.

Conclusion

La conception de ce type d'opérateurs spécialisés est étroitement liée à l'état de l'art de la technologie, dont l'évolution permanente entraîne une rapide obsolescence des structures.Les besoins des systèmes de vision commencent à être précisés et la vision architecturale de ceux-ci est beaucoup plus prévisible. Dans cette optique, il est nécessaire de concevoir des structures spécialisées qui puissent bénéficier progressivement de l'évolution de la micro-électronique, c'est à dire des opérateurs dont l'architecture est toujours viable et efficace s'ils sont réalisés avec des technologies permettant de doubler le nombre de transistors qui les composent ou la fréquence de leur horloge de calcul.

L'objectif de cette étude était de spécifier et de démontrer la faisabilité technique d'un dispositif de calcul parallèle reconfigurable (coprocesseur de traitement d'images) interfacé avec un bus système, et non des bus vidéo, qui soit capable de réaliser des tâches de traitement d'images bas et moyen niveau sur des images complètes ou des zones d'images de forme arbitraire. Les études sur la nature des algorithmes à réaliser nous a conduit à architecturer ce dispositif en deux blocs fonctionnels distincts : un dispositif non nécessairement parallèle assurant la lecture ou l'écriture de données en mémoire et un réseau de calculs parallèle cadencé par la disponibilité des données qui peut comporter un nombre variable de dispositifs de calcul tous interfacés de la même manière. L'essentiel de ce travail de recherche a eu pour but d'étudier des dispositifs de calcul parallèles reconfigurables, adaptés à cette architecture, l'un d'entre eux a même été réalisé sous forme d'un monocircuit intégré : INP20.

L'architecture que nous avons définie dans le cadre de cette étude s'approche de ces critères par quatre aspects principaux :

- l'architecture externe des opérateurs est indépendante de leur mécanisme de calcul interne
- les circuits composant le coprocesseur peuvent être conçus séparément
- les circuits dans leur état actuel, peuvent déjà bénéficier d'une évolution de l'intégration, si celle-ci permet d'intégrer localement de la mémoire.

- ses performances sont tout à fait comparables, voire supérieures sur certains algorithmes spécifiques aux meilleurs systèmes de traitement d'images actuels

Nous n'avons, par contre, pas prévu que les versions intégrées de ces opérateurs permettraient des cadences de calcul aussi importantes (25 mégapixels à la seconde pour INP20). Nous nous sommes basés sur une architecture où, à chaque cycle de calcul, on traitait une information. Pour fournir ces informations aux fréquences maximales de calcul du circuit INP20, il est nécessaire de réaliser un processeur d'interface mémoire très complexe et incompatible avec la taille limitée que nous nous sommes fixés pour le module élémentaire de traitement. Si on considère que les futures versions de ces circuits seront réalisées avec une technologie micro-électronique inférieure à 2 microns, cette fréquence de calcul ne peut que croître.

Il peut donc être judicieux, pour réduire la complexité du processeur d'interface mémoire et augmenter encore les performances du coprocesseur, de réduire le rapport fréquence des dialogues externes / fréquence de calcul, c'est à dire utiliser des opérateurs parallèles spécialisés "microprogrammés" où chaque unité de traitement réalise une petite séquence de calcul durant un ou plusieurs cycles d'horloge pour chaque pixel à traiter. Cette approche d'opérateurs pipe-line "microprogrammés" est plus complexe que l'approche classique basée sur le traitement d'un pixel à chaque cycle d'horloge. Elle est, à notre avis, l'avenir de ce genre de structure et ses premiers prémices apparaissent déjà dans des circuits tels que RISP (AON87) et VSP (YAM87).

Nous n'avons pas abordé en détail les différentes manières de réaliser l'unité d'interface mémoire. Des informations récentes à propos de la dernière génération de transputers (INMOS série T9000) nous laissent espérer dans un futur proche, qu'une réalisation logicielle des tâches dévolues à cette unité sera réalisable par des microprocesseurs à des vitesses :

- dépassant les 10 mégaoctets/seconde en utilisant par exemple les liens série haute vitesse de cette nouvelle famille de transputer pour le dialogue avec l'unité pipe-line de traitement
- ou plus rapides encore en utilisant les instructions de transferts de blocs d'octets vers des périphériques disponibles sur le bus système.

Il nous est difficile de juger la validité et l'intérêt technique de cette architecture d'un point de vue global, par contre nous considérons que cette étude nous a permis de démontrer qu'il était possible de réaliser des dispositifs de calcul pipe-line et systolique multi-fonctions pour le traitement d'image sans augmenter notablement la complexité microélectronique de ces derniers.

BIBLIOGRAPHIE

ABE87 M. ABE, H. KOKUBUN, T. AIDA, K. GOTO, K. KOBAYACHI
"A high speed digital filter LSI for video signal Processing"
IEEE solid-state circuits, Vol sc-22, N° 3, June 1987, pp (396, 402)

ANA87 M. ANNARATONE, E. ARNOULD, T. GROSS, H.T. KUNG and all
"The Warp Computer : Architecture, Implementation, and Performance"
IEEE transaction on computer, Vol C-36, N° 12, December 1987 pp (1523, 1538)

AON87 K. AONO, M. MARUYAMA, T. MORI, H. YAMADA, K. HATOYA
"Implementation of a Bipolar Real Time Image Signal processor - Risp II"
IEEE solid-state circuits, Vol. sc-22, N° 3, June 1987, pp (403, 408)

BAS86 J.L. BASILLE
"Structures parallèles pour le traitement d'images"
Journée SEE, architecture et processeur de traitement de signal, Octobre 86,
Gif sur Yvette, France, pp (5-1, 5-14)

BAT85 B.G. BATCHELOR, D.A. HILL, D.C. HODGSON
"Automated visual inspection"
IFS Publication Ltd, North Holland 1985, UK

BAJ88 T. BAJI, H. KOJIMA and all
" A 20 nS CMOS Micro DSP core for Video Signal Processing"
IEEE Solid State Circuits, Vol 23, N° 5, October 88, pp (1203,1211)

BON86 P. BONTON, J.P. DERUTIN, J. GALLICE, L. GROUCHE
"Les éléments structurants plans et volumiques sur des images à niveau de gris
appliqués sur un opérateur temps réel vidéo"
Traitement du signal, vol. n°3, n°6, 1986, pp (313, 319)

BRE79 B.Y. BRETAGNOLLE, C. RUBAT DU MERAC
"Romuald, un multiprocesseur interactif pour la saisie et le traitement d'image"
2ème Congrés Afcet-Iria, Reconnaissance des formes et Intelligence Artificielle,
Toulouse, France, Septembre 1979

BUR84 D. J. BURR
"A fast filtering operator for robot stereo vision"
7th International Conference on Pattern Recognition, Montréal 1984, Canada
pp (669-672), ISBN 08186 0545-6

BUR83 P. J. BURT
"Fast algorithms for estimating local image properties"
Computer vision, graphics, and image processing, N° 21, 1983, pp (368, 382)

CAN86 J. CANNY
"A computational approach to edge detection"
IEEE transactions on pattern analysis and machine intelligence
Vol. PAMI 8, N° 6, November 1986, pp (679, 698)

CAS84 S. CASTAN
"Extraction d'informations élémentaires dans les images"
4ème Congrés reconnaissance des formes et intelligence artificielle, Rocquencourt,
France, janvier 1984, pp (147, 164)

CAS85 S. CASTAN
"Architectures adaptées au traitement d'images"
Technique et science informatique, 1985, N° 5, pp (431, 445)

CEA88 CEA / IRDI / DLETI / SETIA
"INP20 : " Image Neighborhood Processor " processeur de traitement d'image" présentation préliminaire
Rapport CEA / IRDI / DLETI / SETIA, 1988

CHA87 Y. CHAN, A. YEN
"Use structured arrays for high-performance data processing"
Electronic Design News, April 1987, pp (177, 184)

CUB85 D.L. McCUBBREY, R.L. LOUGHEED
"Morphological image analysis using a raster pipeline processor"
IEEE 6th conference on computer architecture, 1985, pp (444, 452)

DAV80 D. DAVID
"Traitement d'image par analyse de connexité et paramétrisation en un passage"
Thèse de Docteur Ingénieur présentée à l'INPG le 28 Janvier 1980

DAV88 D. DAVID, T.COURT, J.L. JACQUOT, A. PIRSON
"INP20 : An image Neighborhood processor for large kernels"
IAPR workshop on computer vision, TOKYO october 12-14, 1988 pp (241, 244)

DAW88 B.M. DAWSON
"Developments in image processing hardware"
Beeldverwerking Vision System seminar, May 1986,
MIKROCENTRUM, Netherland, pp (a_1, a_{20})

DER84 J.P. DERUTIN, J. ALIZON, J. GALLICE
"Détermination de la position d'un objet manufacturé en temps réel vidéo par processeurs câblés"
1er Colloque image Traitement Synthèse Technologie et Application, 1984, Biarritz, France, pp (455, 459)

DES81 J.D. DESSIMOZ, J. BIRK, R. KELLEY, J. HALL
"A vision system with splitting bus"
IEEE 2nd conference on computer architecture, 1981, pp (62, 66)

DEW86 P. DEW, L. MANNING
"Comparison of systolic and SIMD architectures for computer vision computations"
1st international workshop on systolic array, July 1986, pp (273, 282),
Adam Hilger Publication, Oxford, England

DIE88 T. DIEDE, C.F. HAGENMAIER, G.S. MIRARKER and all
"The Titan Graphics Supercomputer Architecture"
IEEE Computer, September 1988, pp (13, 30)

DOL89 S. DOLAN, E. SASENA
" Hough in hardware processors get algorithm specific "
Electronic System Design Magazine, February 1989, pp (51, 58)

DUF81 B.DUFF, S. LEVIALDI
" Langage and architecture for image processing "
Academic Press, 1981, London, UK

DUF83 M.J.B.DUFF
" Computing structures for image processing "
Academic Press, 1983, London, UK

DUF88 M.J.B. DUFF
"Some considerations on the limitations of image processing computer architecture"
IAPR Workshop on Computer Vision, October 88, Tokyo, Japan, pp (1-5)

ELD88 J. ELDON, R WEGNER
" Using the TMC2301 Image Resampling Sequencer "
TRW LSI products, Applications Notes, 1988

ELP87 A.C. ELPHINSTONE, A.P. HERON, G.S. HOBSON and All
"RAPAC : a high speed image processing system"
IEEE Proceeding, Vol.134, n° 1, January 1987, pp (39, 46)

ETC89 Etablissement Technique Central de l'Armement (ETCA)
Compte rendu d'activité 1987/1988 "systèmes de perception"
ETCA, 1989, Arcueil, France

EYC85 L. VAN EYCKEN, P. WANBACQ, A. OOSTERLENCK
"The time-shared bus, a viable way to image multiprocessing"
SPIE, Vol 534, Architecture and algorithms for Digital image processing II, 1985, pp (2,7)

FER86 L.A. FERRARI, P.V. SANKAR, J. SKLANSKY, S. LEEMAN
"Efficient two-dimensional filters using B-Spline Function"
Computer vision, graphics, and image processing, N° 35, 1986, pp (152,169)

FUJ88 Y. FUJITA, M. IWASHITA, T. TEMMA
"A large scale image processing system TIP.4 Prototype"
IAPR Worksop on Computer Vision, October 88, Tokyo, Japan, pp (365, 368)

FUK84 T. FUKUSHIMA, Y. KOBAYASHI and all
"ISP, a dedicated LSI for image local operation"
IEEE 7th International Conference on Pattern Recognition, August 1984, Montréal, Canada, pp (581, 584)

FUK85 T. FUKUSHIMA, Y. KOBAYASHI and all
"Construction methods and performance evaluation for image processing system with ISP LSI"
ICASSP 86, April 86, Tokyo, Japan, pp (789, 782)

FUK86 T. FUKUSHIMA, Y. KOBAYASHI and all
"Architecture of an Image Signal Processor 2, ISP2"
IEEE 8th International conference on Pattern Recognition, Octobre 86, Paris, France, pp (38, 41)

GAS87 W.S. GASS, R.T. TARRANT and all
"Multiple digital signal processor environment for intelligent signal processing"
Proceeding of IEEE, Vol 75, N° 9, September 1987, pp (1246, 1259)

GAI84 G. GAILLAT
" Le calculateur parallèle CAPITAN, 600 mips pour l'imagerie temps réel"
1er Colloque Image Traitement Synthèse Technologie et Application, Mai 1984
Biarritz, France, pp (473, 482)

GOT85 T. GOTOH, S. SOSAKI, M. YOSHIDE
"Two image processing systems challenging the limits of local parallel architecture"
IEEE 6th conference on computer architecture, 1985, pp (272, 279)

GRO87 R. GRONDALSKY
"A VLSI chip set for massively parallel architecture"
ISSCC 87, 1987, pp (198, 199)

GUE86 C. GUERRA
"Systolic algorithms for local operations on images"
IEEE transaction on computer, Vol. C35, N° 1, January 86

HAN88 P. HANNINEN, J. VATANEN, J. SALO, J. TATALE
"Tagips, and adaptable parallel processor for imaging applications"
IAPR Workshop on Computer Vision, October 88, Tokyo, Japan pp (365, 368)

HAS81 M. HASEGGAWA, T. NAKAMURA, Y. SHIGEI
"Distributed communicating media, a multitrack bus, capable of concurrent data exchanging"
IEEE 2nd conference on computer architecture, 1981, pp (367-372)

HAT86 M. HATAMIAN
" A real-time two dimensionnal moment generating algorithm and its single chip implementation "
IEEE ASSP, Vol. 34, N°3, June 1986, pp (546, 553)

HIT88 HITACHI Ltd
" Hitachi IP/200 Image Processor "
Hitachi Industrial Products, Application Note

HOT86 T. HOTTA, I. MASUDA, H. MAYIMA and all
"CMOS/Bipolar circuits for 60 MHz Digital Processing"
IEEE solid-state circuits, Vol sc-21, N° 5, October 1986, pp (808, 813)

HOR84 H. HORGEN, B. ZAVIDOVIQUE
"Correspondance entre algorithmes et architectures de machines"
4ème Congrés reconnaissance des formes et intelligence artificielle, Rocquencourt, France, janvier 1984, pp (103, 116)

HUE86 A. HUERTA, G. MEDIANI
"Detection of intensity changes with subpixel accuracy using laplacian gaussian mask"
IEEE transaction on pattern analysis and machine intelligence
Vol. PAMI 8, N° 5, September 1986, pp (651, 664)

INO88 A. INOUE, A. MAEDA
"The architecture of a multi-vector processor system, VPP"
Parallel Computing, N°8,1988, pp(195,193)

JAC88 J.L. JACQUOT, T.COURT, D. DAVID, A. PIRSON
"Une classe d'extraction de contours performants adaptés à une implémentation matérielle fonctionnant à cadence vidéo"
TIPI88, second atelier "du traitement du pixel à l'interprétation", Aussois, France, avril 1988

JEN87 R.E. JENKINS, D.G. LU
"An application - specific coprocessor for high speed Cellular Logic Operations"
IEEE micro, December 1987, pp (63, 70)

JOH86 T. JOHNSON, T DURHAM
" Parallel Processing "
Ovum Publications, London, Great Britain

KAN87 K. KANEDO, T. NAKAGAWA, A. KUICHI and all
"A 50 ns DSP with Parallel Processing Architecture"
ISSCC 1987, February 1987, pp (158, 159)

KAN88 K. KANAYAMA, T. FUJI, N. OHTA, S. ONO
"Architecture and performances of a multicomputer type digital signal processing system"
ICASSP 1988, April 1988, New York, USA, pp (1977,1980)

KAW86 T. KAWADA, Y. TAKAHASHI, N. TSUDA and all
"A pattern matching processor array with defect tolerance"
ISSCC86, February 1986, pp (90, 91)

KID83 M. KIDODE
"Image processing machines in Japan"
IEEE Computer, January 83, pp (68, 79)

KIT87 J. KITTLER, J. EGGLETAN, J. ILLINGWORTH, K. PALER
"An averaging edge detector"
Pattern Recognition letters N° 6, June 1987, pp (27, 32)

KOB87 Y. KOBAYASHI, T. FUKUSHIMA, S. MUIRA and all
" A BICMOS Image Signal Processor with line memories"
IEEE ISSCC87, March 1987, pp (182,183)

KOB87 Y. KOBAYASHI, E. OSAKI, K. HORIGUCHI, H. YARO
"High speed 32 bits Image Processor System DSPT 9506"
IAPR workshop on Computer Vision, October 88, Tokyo, Japon, pp (375, 379)

KUN79 H.T. KUNG
"Let's design algorithms for VLSI systems"
Conference on VLSI, architecture, design, fabrication
California institute of Technology, January 1979, pp (65, 90)

KUN80 H.T. KUNG
"Special purpose devices for signal an image processing : an opportunity for VLSI"
Proceeding of SPIE, Vol 241, Real time signal Processing III, July 1980, pp (76, 84)

KUN82 H.T. KUNG
"Why systolic architectures"
Computer magazine, Vol 15, January 82, pp (37, 46)

KUN84 S.Y.KUNG
"On supercomputing with systolic/wavefront array processors"
Proceeding of the IEEE, Vol.78, N°7, July 1984, pp(867,884)

KUS81 T. KUSHNER, A.Y. VU, A. ROSENFELD
"Image processing on ZMOB"
IEEE 2nd conference on computer architecture, 1981, pp (88-95)

LOU85 R.M. LOUGHEED, C.W. SWONGER
"An analysis of computer architectural factors contributing to image processor capacity"
SPIE Vol 596 Architecture and algorithms for Digital Image Processing, 1985,pp (3,13)

LEA88 D. O'LEARY
"Some algorithms for approximating convolutions"
Computer vision, graphics, and image processing, N° 41, 1988, pp (333, 345)

LOP85 J. LOPEZ KRAME
"Détection de structures symétriques et radiales dans les images"
5ème Congrés Reconnaissance des formes et intelligence artificielle, Grenoble, France 1985, pp (803, 813)

LOW83 G. E. LOWITZ
"Can a local histogram really map texture information?"
Pattern Recognition Vol. 16, N° 22, 1983 , pp (141, 147)

LOW84 G. E. LOWITZ
"Méthodes et procédés non paramétriques en traitement d'images"
Cours ESA, DIV008, Edition 84

LSI89 LSI Logic
" L64290 Object Contour tracer preliminary"
LSI Logic, October 1989

LUQ87 E. LUQUE, J. SORRIBES, A. RIPOLI
"Coprocessor for real-time dynamic migration"
Microprocessing and microprogramming, n° 20, 1987, pp (197, 208)

MAL89 C. MALACHOWSKY
"Designing a low cost scalable graphics accelerator"
High performance systems, May 1989, pp (68, 78)

MAR88 M. MARESCA, M.A. LAVIN, H. LI
"Parallel architectures for Vision"
Proceeding of the IEEE, Vol.76, N°8, August 1988, pp(970,979)

MOH87 P. L. MOHAN, D. Mc CUBBREY
" Approaches to real-time feature extraction "
Vision 1987, Conference proceeding, Detroit, USA, pp (2-27, 2-39)

MOL88 G. MOLLAH, B. SCHNEIDER, T. LEHNERT
"A special hardware for detection of attaching or overlapping objects"
IAPR workshop on Computer Vision, October 88, Tokyo, Japon, pp (116, 119)

MOR85 S.G. MORTON, E. ABREN, F. TSE
"ITT CAP - Toward a personal supercomputer"
IEEE micro, December 85, pp (37, 49)

OKL88 V.G. OKLOBDZIJA
"Issue in CPU-Coprocessor communication and synchronisation"
Microprocessing and Microprogramming, Vol. 24, 1988

PAU88 D. PAUL, W. HATTICH, W. NILL, S. TATARI, G. WINKLER
"Vista : Visual Interpretation system for technical applications - Architecture and Use"
IEEE Transaction PAMI, Vol. 10, N° 3, May 1988, pp (399, 407)

PIR88 A. PIRSON, T.COURT, D. DAVID, J.L. JACQUOT
"A highly efficient method for synthesizing some digital filters"
EUSIPCO 88, fourth european signal processing conference,
Grenoble, France, septembre 1988, pp (1481, 1484)

PIR90 A. PIRSON, T.COURT, D. DAVID, J.L. JACQUOT
"Formalisation en OCCAM et simulation sur un réseau de Transputers d'un processeur systolique de traitement d'images"
à paraitre dans la revue " Traitement du Signal "

PIR90 A.PIRSON
"Conception et simulation d'architectures parallèles et distribuées pour le traitement d'images "
Thèse de l'institut national polytechnique de grenoble, option microélectronique, soutenue le 4 mai 1990

PEL87 M.J.M. PELGROM, H.E.J. WULMS, P. VAN DER STOCKER and all
"Febris : a chip for Pattern Recognition"
IEEE solid-state circuits, Vol sc-22, N° 3, June 1987, pp (423, 429)

PFE87 D.M. PFEIFFER
" High-powered desktop image processing gets closer "
Electronics, February 1987, pp (62, 64)

PLE86 PLESSEY Semiconductors
"PDSP 16401, a 2 dimensional edge detector "
Plessey Data Sheet

POT83 J.L. POTTER
"Image processing on the Massively Parallel Processor"
IEEE computer, January 83, pp (62, 67)

PRA82 W.K. PRATT, T.J COOPER
"Recursive pipeline architecture aids image processing"
Computer technologie Review, Winter 1982, pp(117,118)

PRA85 W.K. PRATT
"A pipeline architecture for image processing and analysis"
IEEE conference on computer architecture, 1985, pp (516,520)

REA85 C. READER, S. SEARING, J. SKUBIC, M. VASQUEZ, L. WEINER, G. BOSE
"A new architecture for real time image processing"
SPIE, Vol 534, Architectures and algorithms for Digital Image Processing II, 1985, pp (72, 85)

ROE78 R.P. ROESSER
"Two. dimensional microprocessors pipelines for image processing"
IEEE transaction on computer, N° 2, Vol C. 27, February 1978, pp (144, 156)

ROS83 A. ROSENFELD
"Parallel image processing using cellular arrays"
IEEE computer, January 83, pp (14, 20)

ROU86 F. ROUSEE
"Un processeur modulaire pour la synthèse d'images"
2éme colloque image, traitement, synthèse et applications
Nice, France, Avril 1986, pp (368, 373)

RUB82 C. RUBAT DU MERAC
"Romuald"
Journées de travail Afcet sur les architectures spécialisées en traitement d'images
ETCA 8, Décembre 1982

RUE87 P.A. RUETZ, R.M. BRODERSEN
"Architectures and design techniques for Real-Time Image processing IC's"
IEEE solid-state circuits, Vd sc-22, N°2, Avril 87, pp (233, 250)

RUE87 P.A. RUETZ, P.H. ANG
"A chip set for real time 20 MHz DSP"
ICASSP 1988, April 1988, New York, USA, pp (1977,1980)

SAM84 R. SAMY
"Traitement de séquences d'images télévision"
Thése de 3ème cycle, Université d'Aix Marseille, faculté des sciences et techniques de St Jérome, soutenu le 9 mai 1984

SAN86 J.L.C. SANZ, E.B. HINKLE
"Computing projections of digital images in image processing pipeline architectures"
IEEE Transaction ASSP Vol. 35, N° 2, February 1987, pp (198, 207)

SAN87 J. L.C. SANZ, I. DINSTEIN, D. PETKOVIC
"Computing multi-colored polygonal masks in pipeline architectures and its applications to automated visual inspection"
Communication of the ACM, Vol. 30, N° 4, April 1987, pp (318, 329)

SCH89 D. SCHIDMORE
"Fast times for Flexbus"
ESD Electronic System Design, September 1989, pp (63, 67)

SER82 J.SERRA
" Image analysis and mathematical morphology ", Academic Press, London 1982, UK

SHO85 R.L. SHOEMAKER, P.M. BARTELS and all
"Image-data-driven dynamically-reconfigurable multiprocessor system in automated histopathology"
SPIE, Vol 596, Architecture and algorithms for Digital image processing, 1985, pp (190,198)

STR85 Z. Z. STROLL, E. E. SWARTZLANDER, J. ELDON
" VLSI for Image Rotation "
VLSI Signal Processing, IEEE Press, 1984

TAK83 N. TAKAHASHI, N. AMAMIYAMA
" A data flow processor array system : Design and Analysis"
IEEE 4th conference on Computer Architecture, 1985, pp (243, 250)

TAK88 H.TAKENAGA, Y. KOBAYASHI and all
"Architecture of an high performance image processing system"
IAPR workshop on Computer Vision, October 88, Tokyo, Japon, pp (6, 9)

TAN80 S.L. TANIMOTO, A. KLINGER
"Structured computer vision"
Academic Press, 1980, LONDON, UK

TAN88 S.L. TANIMOTO
"Use of a pyramid processor in intermediate-level vision"
IAPR Worshop on Computer Vision, October 88, Tokyo, Japan, pp (427,430)

THE88 B. THEVENIN, D. VARREAU
"Spécification et architecture de l'opérateur N°2 pour le système VX300"
Rapport interne CEA/DLETI/DSYS/SETIA (1988)

THO87 C. THOMPSON, S.M. NICHOLS
"Comparison of advanced VLSI architectures for Machine Vision"
Proceeding of Vision 87, Detroit, USA, pp (2-4, 2-59)

THR87 W. THREATT, N ACCROMANDO
" Image and array processors, a marriage made on VME"
Electronic System Design, October 1987, pp (105, 109)

TOR86 V. TORRE and T.A. POGGIO
"On Edge detection"
IEEE transaction on pattern analysis and machine intelligence, Vol. PAMI 8, N°2, March 1986, pp (,)

TRW87 TRW LSI Products Inc.
" TMC 2243, Cmos FIR Filter "
TRW data Sheet, 1987

TUR87 S. TURNER
"Image processing with the IMS A100"
Inmos Applications Note

TUR88 S. TURNER
"Image and Signal Processing Sub-system IMS A110"
Inmos Data Sheet

VER88 P.W VERBEEK, H.A. VROOMAN, L.J. VAN VLIET
"Low-level image processing by max-min filters"
Signal Processing, vol.15, N°3, October 1988, pp (249, 258)

VAR88 D. VARREAU
"Spécification et architecture de la carte de test pour le circuit INP20"
Rapport interne CEA/DLETI/DSYS/SETIA (1988)

WAL88 I. WALLACE
"An intermediate level vision system for populated printed circuit board inspection"
IAPR workshop on Computer Vision, October 88, Tokyo, Japon, pp (427, 430)

WAL89 R.S. WALLACE, M.D. HOWARD
"HBA Vision Architecture : Built and Benchmarked"
IEEE transaction on pattern analysis and machine intelligence
Vol 11, N° 3, March 1989, pp (227, 232)

WEI87 T. WEIR
"Imaging boards put algorithms in hardware to up throughput"
Digital Design, Mars 1987

WEL86 W. M. WELLS
"Efficient synthesis of gaussian filter by cascaded uniform filter"
IEEE transaction on pattern analysis and machine intelligence
Vol. PAMI 8, N° 2, March 1986, pp (234, 239)

WIE85 J.S. WIYAK, H. BUXTON, B.F. BUXTON
"Convolution with separable masks for Early Image Processing"
Computer vision, graphics and image processing, N° 32, 1985, pp (279, 290)

WIL86 A. WILSON
"Application specific architectures target DSP"
Digital Design, July 1986, pp (69, 73)

WIL87 A. WILSON
"Chip set performs fast Imaging"
Electronic System Design Magazine, August 87, pp (17, 18)

WSI88 WAFERSCALE INTEGRATION Inc.
"High performance programmable standalone microcontroller (PAC)"
Waferscale Integration, PAC 1000, preliminary data sheet

YAG88 N.YAGI, R.YAJIMA, K.ENAMI and All
"Chip set for real-time video signal processing"
IAPR workshop on Computer Vision, October 88, Tokyo, Japon, pp (237, 240)

YAL85 S. YALAMANCHILI, J.K. AGGARWAL
"Analysis of the model for parallel image processing"
Pattern Recognition, Vol 18, N° 1, 1985, pp (1, 16)

YAM87 M. YAMASHINA, T. ENOMOTO, T. KUNIO and all
"A Microprogrammable real time Video Signal Processor (VSP) LSI"
IEEE solid-state circuits, Vol 22, N° 6, December 87, pp (1117, 1122)

YAM88 M.YAMASHINA,T.ENOMOTO and all
"A microprogrammable real time video signal processor, for motion compensation"
IEEE Solid State Circuit, Vol 23, N°4, August 88, pp (907, 915)

YON88 H. C. YOUNG
"Code scheduling methods for some architectural features in PIPE"
Microprocessing and Microprogramming, vol. N°22,1988,pp (39,63)

ZAV84 B. ZAVIDOVIQUE, A.MERIGOT
"Parallélisme massif en traitement d'images"
4 ème congrés reconnaissance de formes et intelligence artificielle, janvier 1984,
Roquencourt, France, pp (117, 140)

ZOR87 ZORAN Corporation
"Digital signal processors"
ZORAN Data Book 1987, pp (21,63)

Liste des figures

Grenoble, le 2 Décembre 1991

ECOLE DOCTORALE

Affaire suivie par Michèle SIMEON
Tél : 76.57. 4525

N/Réf. : JPV/MS

Objet :

AUTORISATION de SOUTENANCE

Vu les dispositions de l'arrêté du 23 Novembre 1988 relatif aux Etudes Doctorales
Vu les rapports de présentation de :

- Mr GALLICE Jean Professeur Université CLERMONT II
  Mr JOURLIN Guy chef de service Electronique ICPI LYON

Monsieur COURT Thierry

est autorisé(e) à présenter une thèse en soutenance en vue de l'obtention du diplôme de DOCTEUR de l'INSTITUT NATIONAL POLYTECHNIQUE de GRENOBLE, spécialité : "Microélectronique"

Pour le Président de l'I.N.P.-G.
et par délégation,
le Vice-Président
M. GARNIER

CENG / SERVICES TECHNIQUES / REPROGRAPHIE / 1992