HAL
open science

Architecture et validation comportementale en VHDL d'un calculateur parallèle dédié à la vision

Thierry Collette

To cite this version:

Thierry Collette. Architecture et validation comportementale en VHDL d'un calculateur parallèle dédié à la vision. Modélisation et simulation. Institut National Polytechnique de Grenoble - INPG, 1992. Français. NNT : . tel-00341876

HAL Id: tel-00341876

https://theses.hal.science/tel-00341876v1

Submitted on 26 Nov 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

TU 18205

THESE

présentée par

Thierry COLLETTE

pour obtenir le titre de

DOCTEUR de l'INSTITUT NATIONAL POLYTECHNIQUE de GRENOBLE

(Arrêté ministériel du 23 novembre 1988)

*Spécialité : **Microélectronique***

ARCHITECTURE ET VALIDATION COMPORTEMENTALE EN VHDL D'UN CALCULATEUR PARALLELE DEDIE A LA VISION

Soutenue le 14 septembre 1992

composition du Jury :

Président:	**J-L. BASILLE**
Rapporteurs:	**J. GALLICE** **C. LANDRAULT**
Examinateurs:	**B. COURTOIS** **A. GUYOT** **D. JUVIN** **J. KAISER**

Thèse préparée au sein du Service Logiciels et Architectures du LETI/DEIN du Commissariat à l'Energie Atomique
(Centre d'Etudes Nucléaires de Saclay)

Remerciements

Ce mémoire est le fruit de trois années de thèse passées dans le Département d'Electronique et d'Instrumentation Nucléaire du LETI. Les idées assemblées dans ce manuscrit n'émanent pas que d'une seule personne, leur élaboration fut possible grâce à la qualité de l'environnement scientifique et humain que j'ai côtoyé pendant ces trois années.

Je remercie monsieur Hamel, chef du DEIN, ainsi que son prédécesseur, monsieur Garderet, de m'avoir offert les moyens techniques d'effectuer ma thèse dans leur département.

J'adresse mes remerciements à monsieur Darnieaud, chef du Service Logiciels et Architectures, ainsi qu'à son prédécesseur, monsieur de Cosnac, de m'avoir accueilli dans leur service.

Je tiens à remercier très vivement monsieur Courtois, directeur de recherches au Centre National de la Recherche Scientifique et directeur du laboratoire TIM3 de l'INPG, d'avoir accepté la direction de cette thèse.

Je remercie monsieur Basille, professeur à l'ENSEEIHT de Toulouse, de m'avoir fait l'honneur de présider mon jury de thèse.

Je remercie également monsieur Gallice, professeur au CUST (Université Blaise Pascal de Clermont Ferrand II), ainsi que monsieur Landrault, directeur de recherches au Centre National de la Recherche Scientifique et directeur du laboratoire LIRM de l'Université de Montpellier II, d'avoir accepté de juger mon travail, ainsi que monsieur Guyot, maître de conférences à l'ENSIMAG, pour m'avoir encadré lors de ma thèse et pour sa participation au jury.

Je tiens à remercier monsieur Juvin, docteur ingénieur et responsable du groupe Architectures et Algorithmes Parallèles (GAAP) du LETI, de m'avoir accueilli dans son équipe, de m'avoir fait confiance lors de cette thèse, et de participer au jury.

Monsieur Kaiser, responsable de ma thèse, m'a toujours prodigué ses conseils avisés et a toujours su mettre mes travaux en avant. Qu'il trouve ici l'expression de ma gratitude.

Un grand merci à Hassane Essafi pour m'avoir soutenu et conseillé lors de cette thèse, ainsi qu'à Pascal Adam, pour avoir corrigé la version britannique du résumé de ce mémoire, de même que pour tous les services qu'il m'a rendu.

Je tiens à remercier également l'ensemble des membres du GAAP pour leur gentillesse et leur disponibilité, notamment mes amis thésards, Marc Viala et Laurent Letellier pour toutes les discussions enrichissantes que nous avons eues ensemble.

Je remercie également Pierre Gallice et Louis Londé, du DEIN, de m'avoir prêté une station de travail, ainsi que Jean Lefèvre et Charles Chaze, ingénieurs système, d'avoir répondu au mieux à mes exigences.

Sans la contribution de messieurs Hantelle, Tissot et Leroux du CEA de Bruyère le Châtel, ce mémoire ne présenterai pas de résultats de synthèse VHDL. Qu'ils soient tous vivement remerciés de leur aimable collaboration.

De même, je tiens à remercier monsieur Airiau, du CNET de Grenoble, de m'avoir offert l'opportunité d'essayer d'autres simulateurs VHDL que celui dont nous disposons.

Enfin, je tiens à signaler la serviabilité de Mireille Denichère et Monique Régnier, en les remerciant également de m'avoir soulagé de certaines tâches administratives.

A mes parents,

Table des Matières.

	Pages
Introduction.	**1**
I : Traitement d'Images et Architectures Parallèles.	**3**
I 1 : Les différentes phases du traitement d'images.	4
I.1.1 : Les opérations de traitement d'images bas niveau.	4
I.1.2 Les opérations de traitement d'images moyen niveau.	5
I.1.3 : Les opérations de traitement d'images de haut niveau.	6
I.1.4 : Conclusion.	6
I.2 : Les Architectures Parallèles.	7
I.2.1 : Les architectures synchrones.	8
I.2.2 : Les architectures asynchrones.	13
I.2.3 : Les architectures mixtes.	18
I.2.4 : Conclusion.	19
I.3 : Parallélisation des opérations de traitement d'images.	20
I.3.1 : Nécessité de la parallélisation.	21
I.3.2 : Les différentes approches.	26
I.3.3 : Conclusion.	30
I.4 : Conclusion.	31
II : Evolution de l'Architecture de SYMPATI2.	**32**
II.1 : SYMPATI2.	33
II 1.1 : L'architecture de SYMPATI2.	34
II 1.2 : Les Performances de SYMPATI2.	40
II.1.3 : Choix de l'approche et conclusion.	41
II.2 : SYMPATI2 et le Traitement Moyen Niveau.	43
II.2.1 : Les Algorithmes Considérés.	43
II.2.2 : Les problèmes Rencontrés.	45
II.2.3 : Conclusion.	47
II.3 : SYMPATIX, l'architecture proposée.	48
II.3.1 : Du Besoin vers l'Idée.	49
II.3.2 : Le Réseau Intelligent Ouvert.	50
II.3.3 : SYMPATIX.	53
II.3.4 : Conclusion.	55
II.4 : Conclusion.	57

III : Méthodologie de Conception et de Simulation. **58**

III.1 : La simulation est-elle une nécessité? 59

III.2 : Environnement d'évaluation. 63
III.2.1 : La conception descendante. 64
III.2.2 : Le langage de description de matériel retenu. 66
III.2.3 : La plate forme de simulation d'architectures. 72

III.3 : La modélisation de SYMPATIX. 75
III.3.1 : La méthodologie de conception du modèle de SYMPATIX. 75
III.3.2 : Elaboration du modèle de SYMPATIX. 79
III.3.3 : Le modèle retenu. 82

III.4 : Conclusion. 88

IV : Résultats. **90**

IV.1 : Le domaine d'application de SYMPATIX. 91
IV.1.1 : SYMPATIX et les opérations de traitement d'images. 92
IV.1.2 : Les opérations graphiques supportées par SYMPATIX. 108
IV.1.3 : Conclusion. 113

IV.2 : VHDL : Simulations et Synthèse. 116
IV.2.1 : La simulation VHDL de haut niveau : une réalité? 116
IV.2.2 : La synthèse VHDL : les contraintes. 118
IV.2.3 : Les résultats de synthèse. 122
IV.2.3 : Conclusion. 123

IV.3 : Evolutions. 124
IV.3.1 : La vitesse de décalage : ses conséquences. 125
IV.3.2 : Améliorations du processeur. 126
IV.3.3 : Travaux à envisager. 127
IV.3.4 : "... et si c'était à refaire". 129

IV.4 : Conclusion. 130

Conclusion. **132**

Références. **134**

Références du Chapitre 1. 135
Références du Chapitre 2. 137
Références du Chapitre 3. 140
Références du Chapitre 4. 141

Annexes. **142**

Opérations de Traitement d'images Moyen Niveau : Présentation des Opérations et de leur Parallélisation. 143
A1.1 : Extraction de Points Caractéristiques. 144
A1.2 : Codage de Freeman. 144
A1.3 : Les "Psi-curves". 146
A1.4 : Approximation polynomiale par les moindres carrés. 147
A1.5 : Le Transformée de Hough. 149
A1.6 : L'étiquetage de régions. 150
A1.7 : Division/Fusion. 156

Détail des Entrées/Sorties de la Cellule. 158

Introduction.

L'ouïe et la vue sont les deux sens que l'homme essaie de reproduire artificiellement. L'électronique est le moyen le plus fréquemment utilisé pour exploiter les informations contenues dans les images et dans les sons. Les images ont commencé à être traitées par les premiers ordinateurs dès les années 1960, et depuis, les applications du *traitement des images* sont nombreuses, elles touchent les domaines industriels tels que, le contrôle non destructif de produits manufacturés, l'aide à la navigation de robots mobiles ou de missiles, la transmission d'images, le graphisme...

Ces traitements s'effectuent grâce à des calculateurs plus ou moins complexes dont le choix dépend de l'application : certaines exigent des temps de calcul rapides et connus, ce sont les applications *temps réel*, d'autres n'ayant pas de contraintes de rapidité rigides, sont supportées par n'importe quel type de calculateur. En revanche, en ce qui concerne les applications temps réel, le calculateur choisi doit effectuer des opérations, sur un nombre de données figé en un temps déterminé. Les dimensions des images augmentent d'année en année, les traitements s'avèrent de plus en plus complexes, et ainsi, pour les applications temps réel, la mise en oeuvre de calculateurs très performants devient une nécessité.

Les évolutions de la technologie des microprocesseurs ont permis aux ordinateurs d'augmenter leurs capacités tant au niveau fonctionnel qu'au niveau temporel. Ces évolutions ne suffisant pas face aux contraintes imposées par les applications du traitement d'images temps réel, aussi, les concepteurs de calculateurs de vision recourent à la technique de la parallélisation d'éléments de calcul. Plutôt que de ne mettre en oeuvre qu'un seul microprocesseur, ces architectes utilisent un ensemble de microprocesseurs, organisés en structure multiprocesseur ou structure parallèle, afin de concevoir des calculateurs d'une puissance graduelle, fonction du nombre d'éléments de calcul mis en oeuvre. De nombreux calculateurs de ce type existent actuellement. Certains, appartenant à des familles différentes, sont dédiés à la vision mais se heurtent à des problèmes d'efficacité du parallélisme liés à la nature des traitements effectués. En effet, on peut considérer simplement que les opérations de traitement d'images se répartissent en deux types, soit elles travaillent au niveau des points

de l'image, on parle alors de traitement iconique, soit elles travaillent au niveau des entités représentatives de l'image, on parle alors de traitement symbolique. En analyse d'images, les traitements iconiques sont effectués avant les traitements symboliques. De manière générale, si un système multiprocesseur dédié à la vision s'avère efficace pour le traitement d'images iconique, il ne l'est pas pour les traitements symboliques et vice versa.

Ainsi, l'objectif des travaux présentés dans cette thèse est de proposer et de valider une structure parallèle adaptée aux opérations de traitements d'images iconiques et symboliques. Plus précisément, cette thèse présente l'adéquation du calculateur SYMPATI2, développé dans notre laboratoire en collaboration avec l'IRIT de Toulouse, aux traitements symboliques. Elle propose des modifications à apporter à ce calculateur afin qu'il puisse supporter cette nouvelle classe d'algorithmes de traitements d'images. Cette étude a donné naissance à un nouveau calculateur parallèle appelé SYMPATIX. Son élaboration et sa validation se sont effectuées grâce à la modélisation du système en un langage de description de matériel, ceci, afin de fabriquer le système par une méthode de conception descendante.

Dans un premier chapitre, sont présentés les opérations de traitement d'images, les divers types de calculateurs parallèles ainsi que les différents schémas de parallélisation des opérations du traitement d'images. Le second chapitre développe, suite à une étude algorithmique, les contraintes liées au passage du traitement iconique vers le traitement symbolique. Ainsi établies, les limites et les carences de SYMPATI2 ont permis de fonder la structure de la nouvelle architecture, SYMPATIX, proposée dans cette thèse. Le langage choisi, pour modéliser les comportements de SYMPATIX, à savoir ses fonctionnalités, de même que les intérêts de ce type d'approche de conception sont exposés dans le troisième chapitre de ce mémoire. Enfin, les résultats obtenus, tant au niveau des performances de ce nouveau calculateur qu'au niveau de la méthodologie de conception utilisée sont présentés dans un dernier chapitre.

Chapitre I

Traitement d'Images et Architectures Parallèles.

Ce premier chapitre présente les opérations de traitement d'images, les calculateurs multiprocesseurs, dits parallèles, et la mise en oeuvre des opérations de traitement d'images sur les structures parallèles. A la fin de ce chapitre, le lecteur sera ainsi plongé dans le contexte de l'étude entreprise lors de cette thèse.

I 1 : LES DIFFERENTES PHASES DU TRAITEMENT D'IMAGES.

Le traitement d'images regroupe une grande quantité d'opérations allant du filtrage à la reconnaissance de formes en passant par des opérations de codage d'images. Le but de ce chapitre n'est pas de présenter en détail les algorithmes mis en oeuvre pour effectuer ces opérations, ceux-ci étant détaillés dans [BAL82, GON87, JAI89], mais de classifier ces opérations qui sont réparties sur trois classes : le traitement d'images bas niveau, le traitement d'images moyen niveau et le traitement d'images haut niveau. Ces trois niveaux sont rapidement présentés ici, et doivent être maîtrisés par les concepteurs d'architectures dédiées à l'imagerie.

I.1.1 : Les opérations de traitement d'images bas niveau.

Les opérations de traitement bas niveau transforment une image, généralement brute, en une autre image appelée image traitée. L'image brute provient d'un capteur, l'image traitée est visualisée, compressée ou segmentée (figure I.1). Le rôle de ces opérations est brièvement présenté ci dessous.

Commençons par les *opérations de transformation d'images* qui sont souvent utilisées dans d'autres algorithmes présentés par la suite. Ces transformations bi-dimensionnelles permettent de passer d'une représentation, temporelle par exemple, à une autre représentation dans laquelle des opérations de filtrage (ou autres) peuvent intervenir. Pour régénérer l'image de départ, il est nécessaire de connaître et d'effectuer la transformée inverse. Citons par exemple, la transformée de Fourier et la transformée en cosinus qui permettent de passer du domaine temporel, ou spatial, au domaine des fréquences.

Les *opérations d'amélioration ou de filtrage d'images* accentuent ou adoucissent les caractéristiques d'une image afin d'en améliorer sa visualisation ou son analyse. Par exemple, des algorithmes de filtrage et de pseudo-coloriage sont utilisés. Citons les filtres de type

"passe haut" de Sobel, Canny, Deriche, Shen entre autres efficaces pour extraire des contours, les filtres "passe bas" (exemple : filtre de Butterworth) sont eux utilisés pour lisser une image (bruitée) afin de la visualiser par exemple. Ce lissage est également souvent effectué par des filtres particuliers (exemple : filtre médian). L'utilisation de ces filtres nécessite une bonne connaissance à la fois des opérations utilisables et des images à traiter. Ainsi le filtre médian améliore parfaitement une image contenant un bruit binaire alors qu'il présente de mauvais résultats sur une image dégradée par un bruit gaussien.

Restaurer une image consiste à recouvrer une image déformée par un phénomène connu a priori. Ainsi, pour les *opérations de restauration d'images*, la fonction de transfert de la déformation doit être connue et modélisée. L'image subit alors la transformation inverse de cette déformation et retrouve ainsi directement sa description originale. Cette approche est différente de celle, de type heuristique, utilisée lors des opérations de filtrage et pour des déformations non connues a priori. Ces deux approches sont donc complémentaires et non concurrentes.

I.1.2 Les opérations de traitement d'images moyen niveau.

Ces opérations de traitement d'images moyen niveau transforment une image (tableau de données) en des structures de données de type liste, graphe, vecteur etc. Le codage, la compression et la segmentation d'images effectuent ce type de transformation du domaine iconique vers le domaine symbolique. Les opérations de reconstruction d'images, elles transforment une série de projections en une image (figure I.1). Voyons ces différentes opérations.

Les *opérations de codage/compression d'images* interviennent lorsqu'il est nécessaire de comprimer des images, de transmettre des images ou d'extraire les caractéristiques d'une image (segmentation). Les codages s'effectuent dans le domaine temporel, celui des fréquences, ou autres, et ils permettent de se préserver des erreurs lors des opérations de transmission d'images, ils permettent également de compresser des images à transmettre (différence d'images, transformée en cosinus, ...) ou à stocker("run length codes", ...).

Les *opérations de segmentation d'image*s supportent la phase essentielle de reconnaissance automatique de formes ou d'analyse d'images. Les algorithmes utilisés extraient d'une image les objets représentatifs afin de les organiser, puis de les présenter en une forme acceptée par les opérations de haut niveau. Les discontinuités d'une image tels les points remarquables, les lignes et les contours peuvent être détectés par des opérations de

filtrage ou des transformations particulières (transformée de Hough). Des approches de segmentation orientée région ou par le mouvement existent également.

Enfin, les *opérations de reconstruction d'images* qui sont singulières car contrairement aux précédentes, elles génèrent une image à partir de données organisées en structures particulières. Nous avons classé ces opérations dans cette section car elles effectuent l'inverse des opérations de codage : d'un ensemble de projections transaxiales, une image est crée. Ces opérations sont utilisées en tomographie, en observation par résonance magnétique...

I.1.3 : Les opérations de traitement d'images de haut niveau.

Les opérations mises en oeuvre lors de cette phase sont complexes car elles constituent la phase intelligente du traitement d'images. Ainsi certaines opérations s'appuient sur des algorithmes connexionnistes (recuit simulé par exemple), d'autres manipulent des bases de données regroupant les objets à identifier. Ces opérations demandent généralement la ré-exécution de certaines opérations de bas et moyen niveau sur une zone particulière de l'image, ou commande l'exécution de nouveaux algorithmes. C'est dans cette phase que les opérations exécutées dépendent le plus des données.

I.1.4 : Conclusion.

Ce paragraphe présente et classifie les différentes opérations de traitement d'images. Le lecteur pourra ainsi s'y référer tout au long de cette thèse, afin de situer les algorithmes présentés par la suite, de même que le domaine d'application de la structure proposée. La figure I.1 reprend les différentes opérations exposées dans ce paragraphe. Il est important de noter, que les opérations de haut, voire de moyen, niveau peuvent demander la réexécution d'opérations de bas niveau. Nous n'avons présenté que les opérations de traitement d'images, lesquelles ne constituent qu'une partie de l'imagerie. Ce domaine comporte également les opérations de génération graphique et les opérations de manipulation d'images. Ces dernières sont des opérations d'imagerie de bas niveau car transforment une image en une autre après rotation, déformation... tandis que les opérations de génération graphique peuvent être assimilées aux opérations de reconstruction d'images car une image est créée à partir de listes ou de symboles.

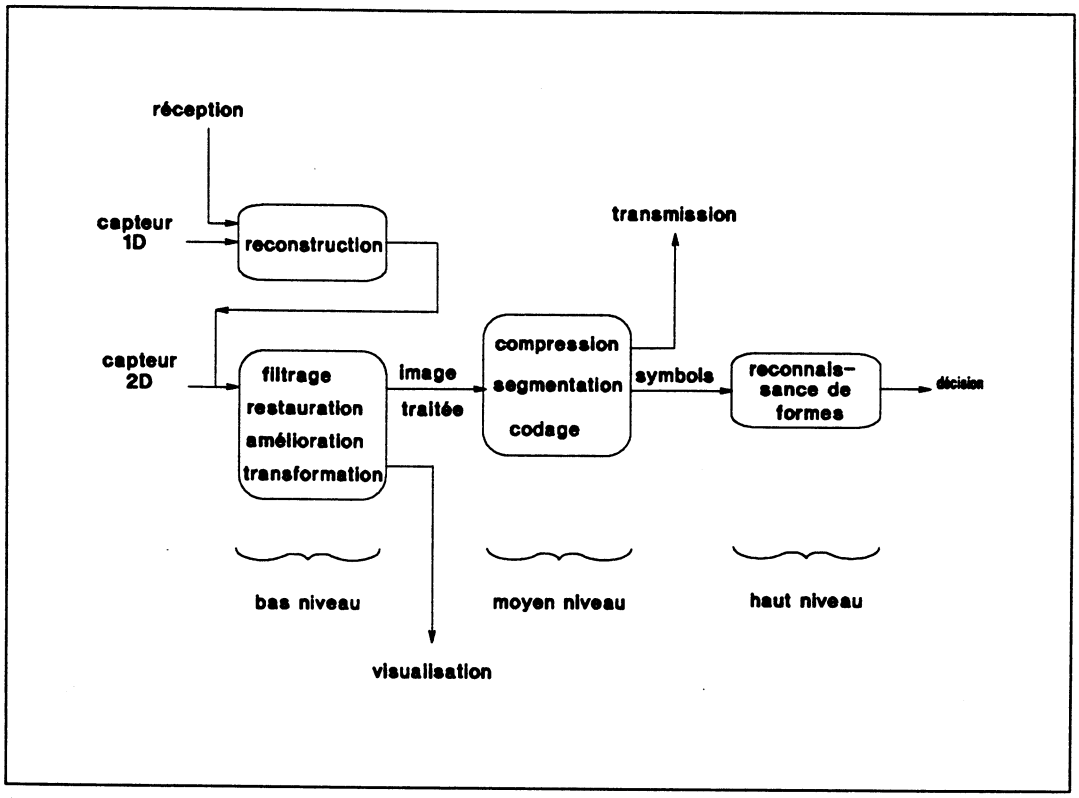

***Figure I.1** : Les différentes opérations du traitement d'images. Ces opérations se répartissent en trois classes, le traitement de bas niveau, le traitement de moyen niveau et le traitement de haut niveau.*

I.2 : LES ARCHITECTURES PARALLELES.

Quand le parallélisme des tâches est-il né? A l'origine de l'homme. Et le parallélisme au niveau des calculateurs? Depuis l'origine des calculateurs, ou plutôt on y pensait [UNG58]. Ainsi le mythe des calculateurs parallèles est devenu réalité simplement depuis leur faisabilité. En effet, pour réaliser de telles structures commercialisables, il est nécessaire de pouvoir intégrer sur un même circuit un nombre de portes logiques conséquent (10 000 portes), ce qui fut possible dès les années 80 avec l'apparition des circuits LSI (Large Scale Integration circuit).

Le parallélisme devient de plus en plus nécessaire car, les besoins en puissance de calcul croissent plus vite que ce que les évolutions de la technologie des semi-conducteurs apportent. Certains voient en ENIAC (1946) le premier calculateur parallèle. En effet, il possède un degré très faible de parallélisme au niveau des unités de calcul, degré trop faible pour être considéré comme une architecture parallèle. Ainsi, dans cette présentation ne sont

considérées que les architectures à fort degré de parallélisme. L'origine des calculateurs parallèles se situe donc vers la fin des années 70 avec la réalisation de calculateurs vectoriels pipelinés dont notamment le premier calculateur de Seymour Cray, le CRAY-1. Depuis cette naissance, de nombreux ordinateurs parallèles ont inondé les publications scientifiques et le marché, jusque là timoré. Nous proposons ici, non pas de détailler toutes ces machines, ceci étant partiellement effectué dans [HWA84, HOC88], ni de présenter les techniques de programmation de ces structures, mais de les classifier.

Plusieurs classifications existent, la plus connue étant celle de Flynn. Dans cette classification, le domaine des architectures parallèles est divisé en quatre familles regroupant les architectures classiques de type SISD (Single Instruction stream, Single Data streams), les architectures à flot d'instructions unique et flots de données multiples SIMD (Single Instruction stream, Multiple Data streams), les architectures à flots d'instructions multiples MISD (Multiple Intruction streams, Single Data stream) et les architectures à flots d'instructions et de données multiples MIMD (Multiple Instruction streams, Multiple Data streams). Cependant, cette classification ne nous satisfait pas, car certains calculateurs tels les pipelines n'y trouvent pas leur place. La classification de Hockney, s'appuyant sur la structure de l'architecture, est mieux détaillée que la précédente mais fait malheureusement abstraction du modèle de programmation ce qui n'est pas le cas de la taxonomie proposée par **Ducan** dans [DUC90]. Nous reprenons ainsi ce modèle de classification qui différencie les architectures synchrones, des architectures asynchrones de type MIMD et des architectures mixtes.

I.2.1 : Les architectures synchrones.

Les architectures synchrones ne possèdent qu'une seule horloge globale et qu'un contrôleur de programmes. Le parallélisme y est obtenu, soit par décomposition des opérations en différentes phases successives supportées par des unités de traitement différentes (cette technique est appelée "pipeline"), soit par duplications des unités de traitement. Nous présentons ainsi les architectures vectorielles pipelinées, les architectures SIMD et les architectures systoliques.

I.2.1.a : Les calculateurs vectoriels pipelinés.

L'apparition des ces calculateurs date de la fin des années 70, et marque la naissance des architectures parallèles. Le qualificatif pipeline concerne l'unité de traitement, où une opération est découpée en différentes étapes. Le nombre de ces étapes représente la longueur du pipeline, et les opérations s'effectuent sur des scalaires. Le qualificatif vecteur, quant à lui,

ne concerne uniquement que les registres et l'organisation mémoire. Ainsi une longueur (N) de vecteurs est définie, et les opérations s'effectuent sur ces vecteurs. Ainsi, lors de ces exécutions, les vecteurs opérandes sont décomposés en séries de scalaires, qui peuvent être de tous types, et injectés de façon séquentielle dans l'unité de traitement pipelinée. Le vecteur résultat est reconstitué à partir de la série de scalaires obtenue (figure I.2.1).

Deux configurations sont à considérer, selon que les transferts s'effectuent de mémoire à mémoire, ou de registre à registre. La première configuration exige la connaissance de l'organisation mémoire, de la longueur des vecteurs et demande donc une électronique d'adressage sophistiquée. Pour la seconde configuration, de registre à registre, cette électronique n'est pas nécessaire mais le nombre de registres internes doit être conséquent.

Les calculateurs CDSTAR-100, CYBER-205 de Control Data corporation et le TI-ASC de Texas Instrument reprennent le mode de transfert de mémoire à mémoire, le Fujitsu VP-200 et le très célèbre CRAY-1 effectuent des transferts de registre à registre.

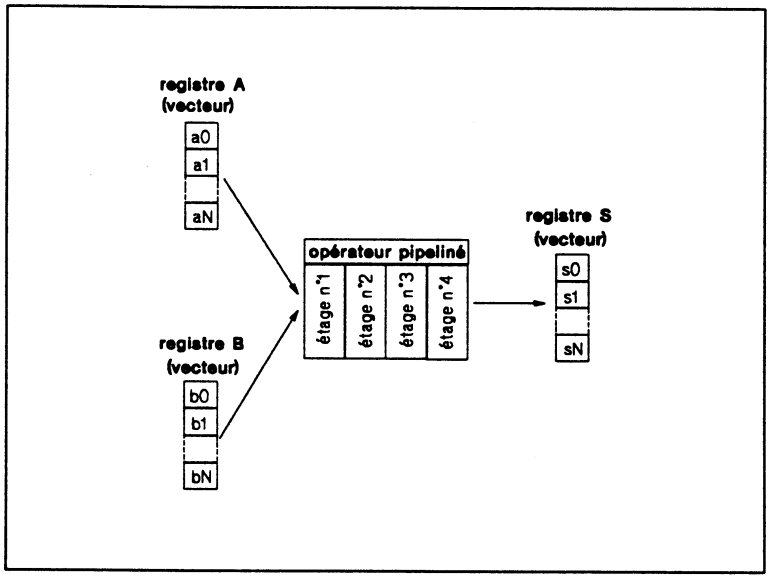

***Figure I.2.1** : Principe de fonctionnement des calculateurs vectoriels pipelinés. Ici le pipeline est constitué de 4 étages, les vecteurs sont de longueur N et les transferts s'effectuent de registre à registre, A et B sont les registres opérandes, S le registre résultat.*

I.2.1.b : Les calculateurs SIMD.

Les calculateurs SIMD ou calculateurs à flot d'instructions unique et flots de données multiples (SIMD : Single Instruction stream, Multiple Data streams) se sont développés à partir des années 80, et continuent à nourrir le marché des calculateurs parallèles génériques

ou dédiés. Ces architectures s'appuient sur un modèle de programmation à parallélisme de données, contrairement aux calculateurs asynchrones qui répondent à un parallélisme de contrôle et éventuellement de données.

Ces architectures SIMD sont caractérisées par leur nombre de processeurs élémentaires (PE), par l'organisation de ces processeurs et par le fait qu'elles ne comportent qu'un seul compteur ordinal, appelé Unité de Contrôle (UC). Cette unité de contrôle envoie, à un instant donné, la même instruction à tous les processeurs élémentaires, lesquels travaillent sur des données différentes, stockées dans des bancs mémoires différents (figure I.2.2).

Un réseau d'interconnexion permet d'organiser les processeurs de différentes façons, en cube, en matrice, en étoile... Pour ce type de calculateur, le parallélisme est obtenu en appliquant directement les traitements sur la structure. Ainsi les opérations matricielles se mettent en oeuvre aisément sur des architectures en matrice 2D de processeurs. Supposons qu'il faille effectuer l'addition de deux matrices A et B de taille N×N sur une structure de N×N processeurs organisés en tableau. Il suffit alors de distribuer les coefficients Aij, Bij respectivement des deux matrices A et B, sur le processeur PEij qui effectue ainsi aisément l'addition Aij+Bij. Tous les coefficients de la matrice résultat sont ainsi calculés en parallèle sur les N^2 processeurs. Toutefois, pour effectuer de telles opérations, il est nécessaire de distribuer les opérandes sur les PEs, de même qu'il est nécessaire de récolter le résultat distribué afin de l'exploiter. Ces architectures doivent donc proposer des mécanismes d'entrées/sorties rapides afin de ne pas dégrader les performances.

Ces calculateurs sont ainsi utilisés dans des applications faisant appel à des calculs réguliers (calcul matriciel par exemple). La simulation numérique, le traitement d'images, la génération d'images, etc, font appel à de telles structures. Les structures SIMD génériques doivent être programmables en un langage de haut niveau à parallélisme de données (exemples : C*, MPL, MultiC et PompC [PAR92]), et doivent posséder un compilateur puissant permettant de paralléliser au maximum les traitements, ceci afin d'optimiser l'utilisation du calculateur. Ainsi le développement de compilateurs associés à des structures parallèles est un domaine porteur. Les performances d'un calculateur SIMD sont autant dues à l'efficacité du compilateur qu'à la qualité de l'architecture.

La Connection Machine (CM2) de l'Université de Carnegie Mellon, le calculateur MasPar de Digital et la MPP de Goodyear Aerospace sont des structures SIMD de type générique, tandis que les calculateurs CAAPP de l'université de Massachussetts et DAP de Active Memory Technology sont eux dédiés au traitement d'images.

Notons que ce type de calculateur regroupe également les processeurs associatifs, tels STARAN de Goodyear Aerospace et PEPE des laboratoires Bell, ayant été conçus pour des applications particulières dans le passé.

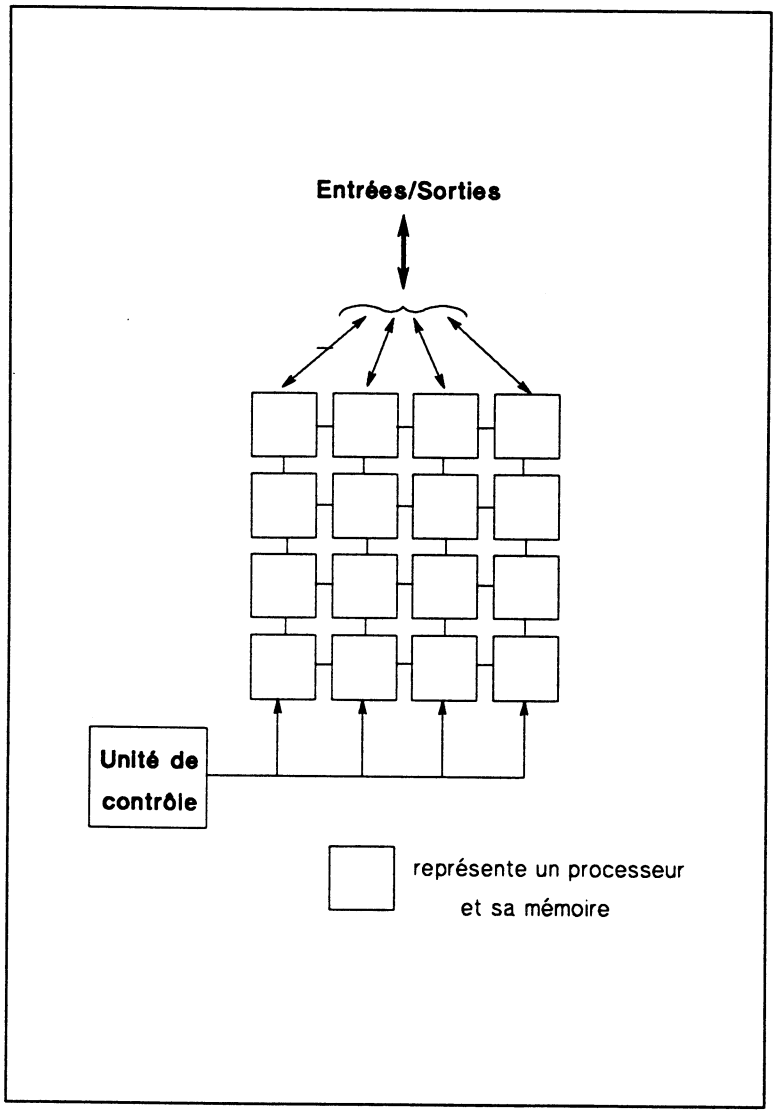

***Figure I.2.2** : Principe d'une architecture SIMD. Exemple d'une matrice de 4×4 processeurs, chacun étant associé à sa mémoire. L'unité de contrôle envoie la même instruction à tous les processeurs de même que le bus d'entrées/sorties reçoit et envoie des données de et à tous les processeurs.*

I.2.1.c : Les architectures systoliques.

Dans le paragraphe précédent, il est dit que les architectures SIMD doivent avoir un système d'entrées/sorties efficace afin de ne pas pénaliser la structure. C'est pour éviter de tels

problèmes que les architectures systoliques ont vu le jour dans les années 80, leur père étant H.T. KUNG de l'Université de Carnegie Mellon [KUN82].

Un peu de médecine. Systole : *"Contraction du coeur par laquelle le sang est chassé dans les artères, qui commence par la contraction simultanée des deux oreillettes (systole auriculaire) suivie de celles des deux ventricules (systole ventriculaire)."*

Ainsi dans une architecture systolique, le flot de données peut être assimilé au sang qui traverse le coeur c'est-à-dire le calculateur. A chaque battement de coeur ou coup d'horloge, un nouveau résultat est fourni alors que de nouvelles entrées sont chargées dans le calculateur.

Un calculateur systolique est un tableau de processeurs pipelinés, traversé par un (ou des) flot(s) de données. La puissance de la structure peut varier graduellement avec toujours la même vitesse, ou bande passante nécessaire, au niveau des entrées/sorties (figure I.2.3). Ce type de structure supporte les algorithmes réguliers faisant appel à des échanges locaux. Les processeurs du calculateur effectuent toujours la même séquence d'instructions entre les "tops" horloge de décalage. Un haut degré de parallélisme est obtenu en utilisant des tableaux de processeurs organisés en deux dimensions. Ce type d'architectures, très régulières, se prête bien à une intégration dans un circuit VLSI ou WSI [FOR87]. Malheureusement, beaucoup de calculateurs systoliques sont dédiés à une application particulière, toutefois des calculateurs systoliques "general purpose" existent tels le WARP de l'Université de Carnegie Mellon [ANN87] et Matrix1 de Saxphy Computer Corporation [FOU87] élargissant le domaine d'intérêt de ce type de machines.

Les domaines d'application de tels calculateurs parallèles couvrent le calcul matriciel rencontré en traitement du signal, en traitement d'images, en simulation de réseaux de neurones ainsi que d'autres applications non numériques telles la reconnaissance de langage, les structures de données et l'intelligence artificielle [KUN88].

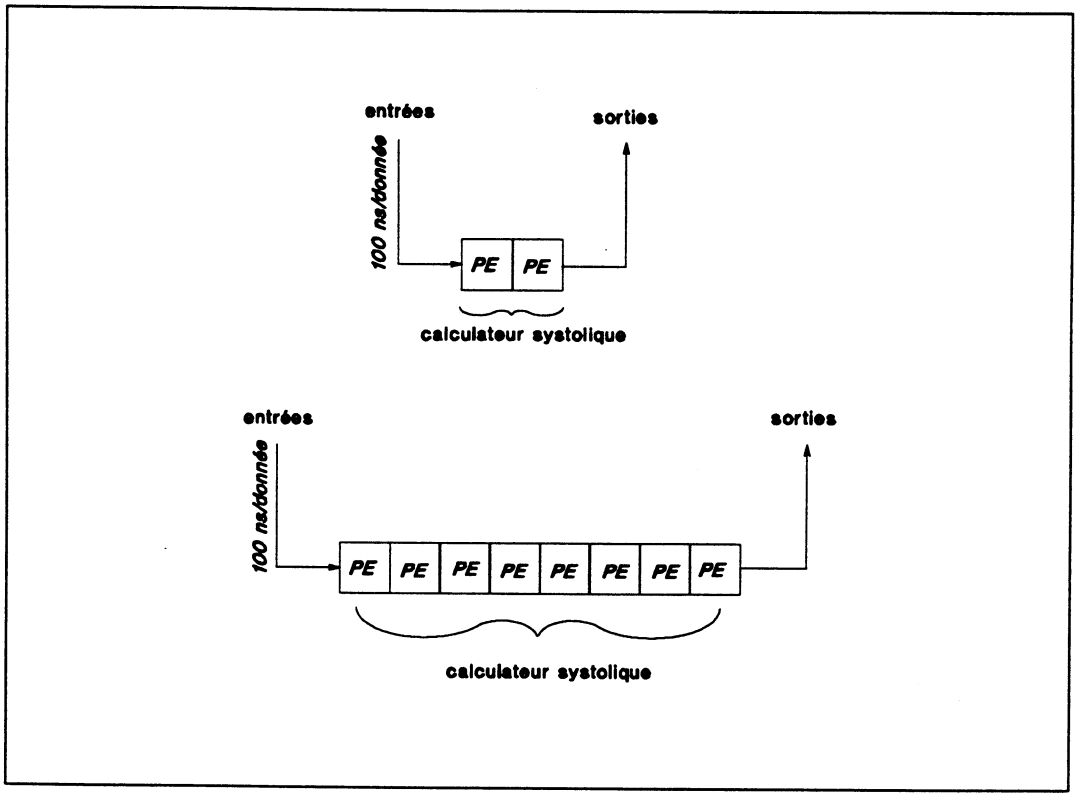

Figure I.2.3 : Principe des architectures systoliques. L'augmentation de la puissance de calcul (nombre de processeurs) est possible tout en conservant la même bande passante. Ici la bande passante est de 100ns par donnée.

I.2.2 : Les architectures asynchrones.

Dans ce paragraphe, les architectures purement asynchrones sont présentées. Ces calculateurs à flots de données multiples et flots d'instructions multiples (MIMD) sont composés d'un ensemble de processeurs exécutant des instructions, des programmes différents. En effet, dans ce type de structure, le contrôle des applications est distribué sur les processeurs contrairement aux architectures SIMD où le contrôle est centralisé. Toutefois, les processeurs ne sont pas totalement indépendants les uns des autres au niveau des données, et doivent communiquer entre eux. Cette communication se fait soit par passage de messages entre processeurs, dialogue ayant comme support un réseau d'intercommunication, ou par partage de données communes, partage supporté par un réseau d'interconnexion. Deux grandes familles de calculateurs MIMD s'opposent, les architectures asynchrones à mémoire partagée et les architectures asynchrones à mémoire distribuée.

I.2.2.a : Réseau d'interconnexion et réseau d'intercommunication [FEN81].

Les réseaux d'interconnexion permettent aux processeurs d'une architecture parallèle d'accéder à un ensemble de données visibles par tous les processeurs. On parle dans ce cas d'architectures à mémoire (données) partagée. Ces connexions entre les processeurs et la, ou les, mémoire(s) s'effectuent par liaison directe entre les éléments dialoguants. Ainsi les échanges peuvent transiter par un bus commun, par un ensemble de bus, au travers d'un crossbar ou d'un réseau multi-étages, chacune de ces solutions offrant des performances différentes [BHU91].

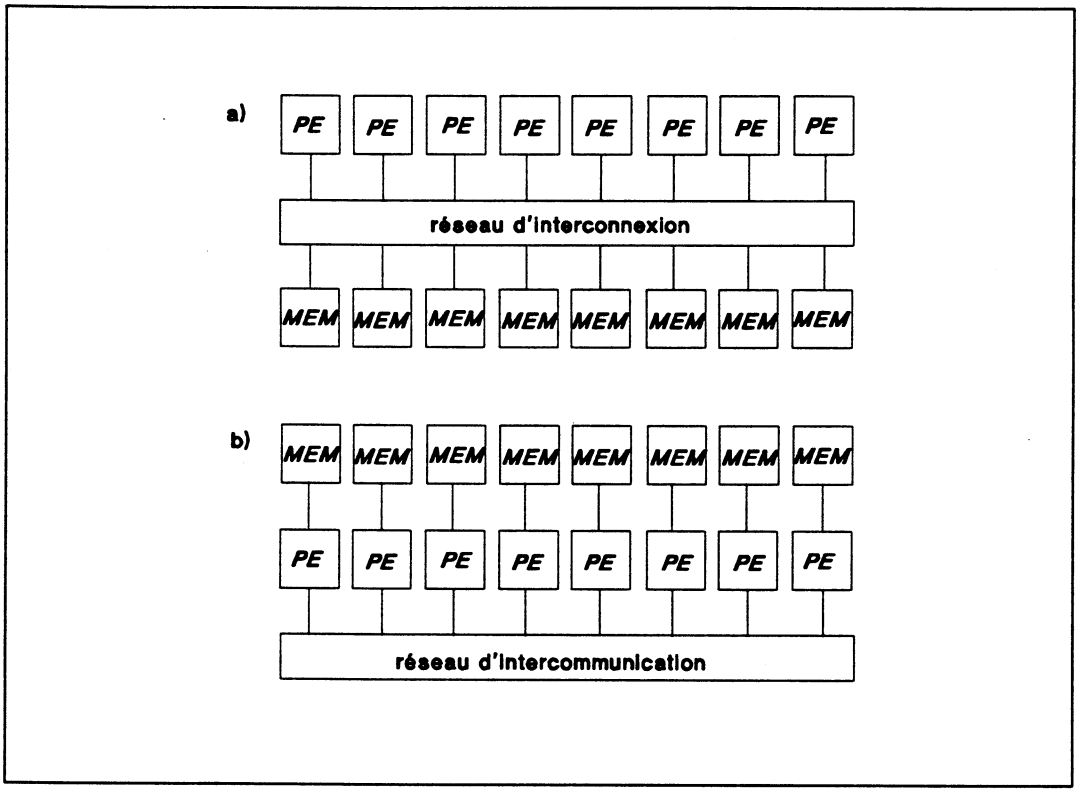

***Figure I.2.4** : Réseau d'interconnexion (a) versus réseau d'intercommunication (b). Il y a connexion directe entre les processeurs et les mémoires pour les réseaux d'interconnexion, le dialogue entre processeurs se fait par passage de messages sur un réseau d'intercommunication.*

Dans les architectures à mémoire distribuée, il ne s'agit pas de réaliser une connexion de processeur à mémoire, mais il s'agit de créer un chemin de communication d'une unité de traitement à une autre unité de traitement. Les réseaux d'intercommunication permettent à ces unités de dialoguer entre elles par passages de messages. Chaque unité de traitement est constituée, au minimum, d'un processeur associé à une mémoire de contrôle et à une mémoire de données. Le réseau d'intercommunication fige la topologie de l'architecture à mémoire

distribuée. Certains réseaux sont reconfigurables et peuvent donc offrir différentes topologies à la structure, tel le crossbar C104 de Thomson-INMOS.

Quel que soit le système de communication choisi, il est nécessaire, avant la conception du système, de déterminer la méthode de synchronisation (par sémaphore, par passage de messages...) des processeurs [DIN89], méthode tributaire du réseau et des processeurs choisis. Les architectures à mémoire partagée et à mémoire distribuée sont présentées dans les paragraphes suivants.

I.2.2.b : Les architectures à mémoire partagée.

Ces architectures sont composées de processeurs exécutant des programmes (tâches) différents sur des données stockées dans une ou des mémoires accessibles par chacun des processeurs. Un réseau d'interconnexion permet de réaliser ce type d'architecture où les processeurs et les mémoires sont fortement couplés. Ce couplage est réalisé directement par bus, par Crossbar ou par réseau multi-étages (figure I.2.5).

Les systèmes MIMD à couplage par bus sont limités, au niveau du nombre de processeurs, et du nombre de mémoires intégrables dans la structure : au delà d'un certain nombre, des problèmes de contentions de bus interviennent et dégradent les performances de la structure. Ce nombre critique dépend de la bande passante du bus, de la rapidité des processeurs et des mémoires (nombre de transferts possibles par seconde), et de l'application invoquée sur le système. Notons qu'actuellement ces architectures, au vu de ces critères, ne peuvent accueillir qu'une vingtaine de processeurs au maximum, mais ce nombre risque d'évoluer avec l'apparition de technologies rapides (AsGa...) permettant de concevoir les bus avec une logique de contrôle rapide. "Malheureusement", pour les défenseurs de ce type d'architectures, la technologie des microprocesseurs, donc leur vitesse, évolue également si bien que ce gain ne sera qu'éphémère.

Pour les architectures à connexion directe, la liaison entre processeurs s'effectuent idéalement grâce à un Crossbar. Ce type de réseau permet à chaque processeur de se connecter directement à n'importe quelle mémoire. Pour une architecture à M processeurs et M mémoires, le crossbar associé nécessaire doit comporter M×M commutateurs ("switches"). Une telle solution est très coûteuse en matériel et en signaux de commande, si bien qu'on lui préfère une architecture comportant un réseau d'interconnexion multi-étages, où tous les processeurs n'ont pas accès à toutes les mémoires à un instant donné, solution offrant un compromis coût/performances acceptable, et donc souvent accepté, mais non optimal. Citons

le calculateur ALLIANT et le NYU Ultracomputer développé à l'Université de New York, qui sont des structures basées sur ce principe de partage de mémoire.

Notons que si la réalisation d'un Crossbar en électronique conduit à une solution "énergivore" et volumineuse, il n'en est pas de même si le crossbar est réalisé de façon optique (grâce à des hologrammes), solution étudiée actuellement et prometteuse, car dans ce type d'interconnexion, le parallélisme lui est inhérent et le nombre de communications non limité [CRO91, HWA88].

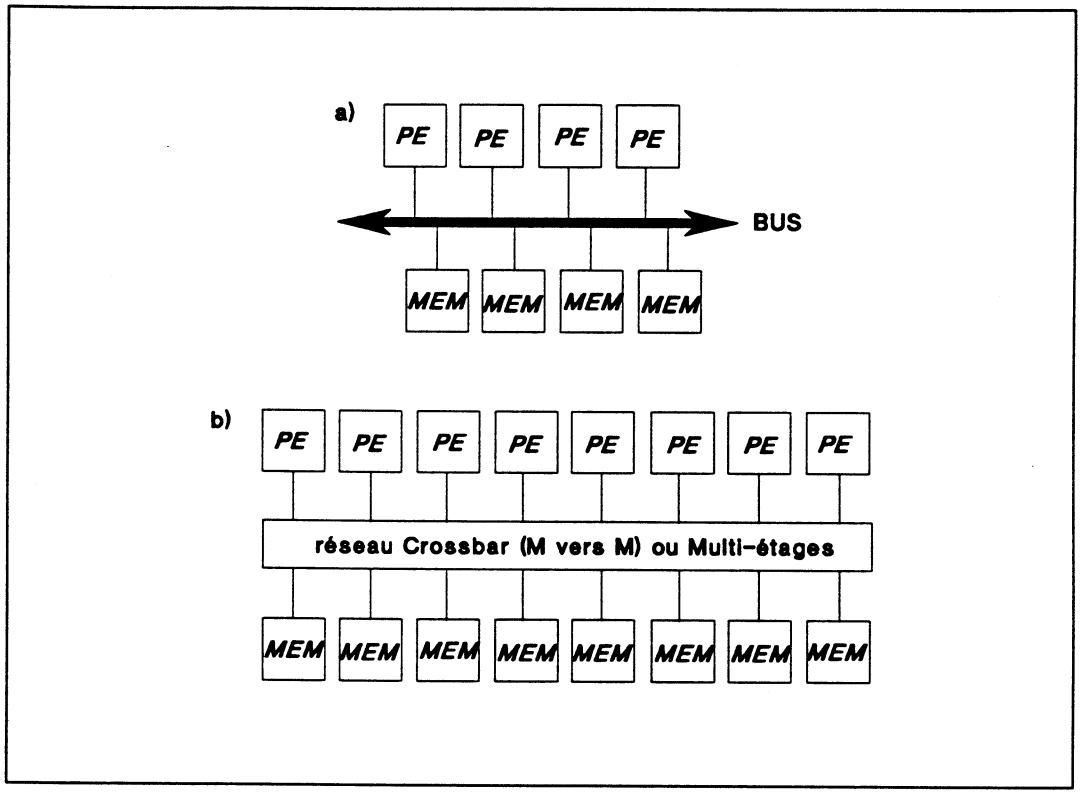

***Figure I.2.5** : Les architectures MIMD à mémoire partagée. L'interconnexion s'effectue soit par bus (a), soit par réseau (b) de type Crossbar (M vers M) ou de type multi-étages (M vers K avec K<M).*

I.2.2.c : Les architectures à mémoire distribuée.

Pour ce type d'architectures, chaque processeur est connecté à sa propre mémoire. Les processeurs exécutent des programmes différents et utilisent un réseau d'intercommunication pour dialoguer entre eux par échanges de messages. Il ne s'agit plus de connexion de processeur à mémoire, mais de connexion de processeur à processeur ou plutôt de "cluster" à "cluster", un "cluster" étant constitué d'un processeur et d'une mémoire au minimum. Ce dialogue s'effectue par passages de messages entre "clusters".

La topologie de l'architecture est définie par le réseau d'intercommunication permettant d'organiser les "clusters" en anneau, en matrice, en hypercube, en arbre... (figure I.2.6). C'est le concepteur qui choisit la topologie de son architecture en fonction des applications qui seront développées sur la structure, évaluation du voisinage nécessaire, du coût et de la modularité exigée. En effet pour ce dernier critère, il est important de noter, qu'une structure de processeurs en anneau permet d'augmenter graduellement la puissance de l'architecture (ajout de processeurs 1 à 1 possible, croissance en M), ce qui n'est pas le cas d'une structure matricielle où le nombre de processeurs doit être une puissance de M^2 (pour augmenter la puissance d'une structure à 4×4 processeurs, il est nécessaire de passer à une structure à 5×5 processeurs, ici la croissance est en M^2). Citons des calculateurs développés à partir du Transputer de Thomson-Inmos, tels la machine Volvox de Archipel, le Tnode de Telmat ou encore les calculateurs de Quintek ou Parsytec.

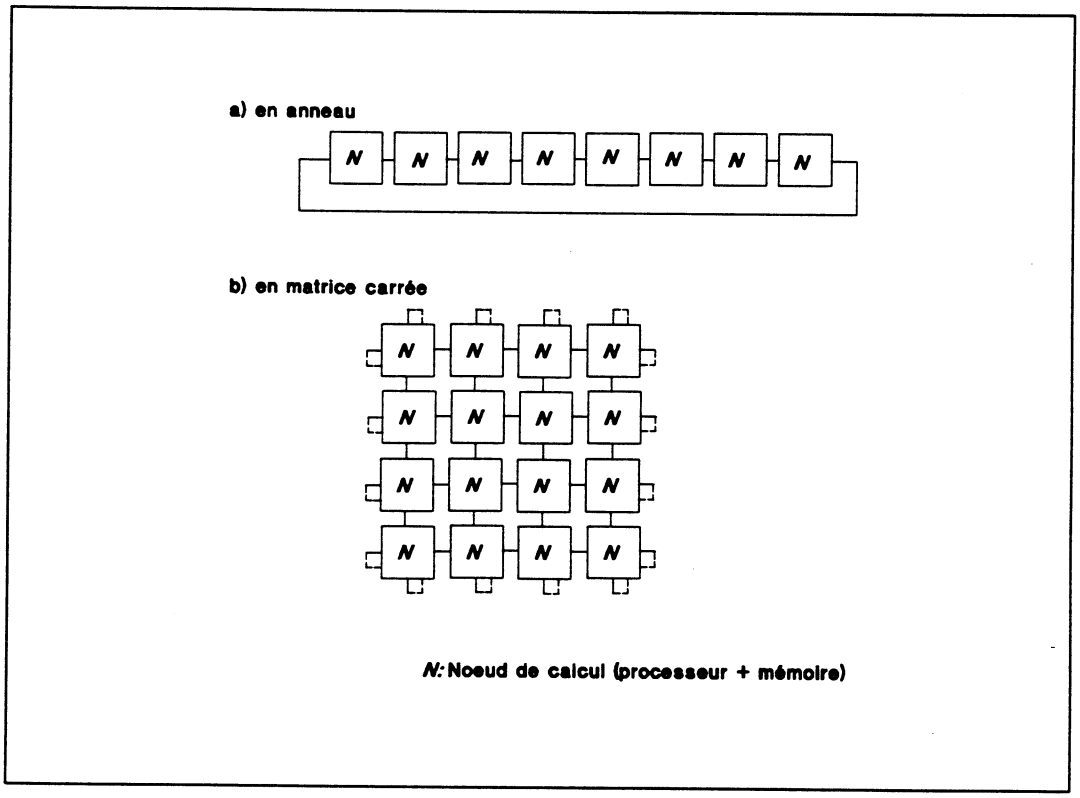

Figure I.2.6 : Les architectures à mémoire partagée. Le réseau d'intercommunication fige la topologie de la structure, ici en anneau (a) ou en matrice bi-dimensionnelle (b).

Signalons que ces calculateurs sont présentés comme totalement asynchrones, mais se programment généralement, pour des raisons de facilité, en mode SPMD (Single Program stream, Multiple Data streams) : le même programme est déroulé sur tous les processeurs, lesquels en contrôlent le déroulement. On garde alors le parallélisme au niveau des données

en réduisant fortement le parallélisme au niveau du contrôle. A capacité de processeurs identique, ces structures sont globalement plus rapides que les structures SIMD (Les instructions incluses dans les conditions non remplies ne sont pas exécutées en SPMD), au prix d'une électronique bien plus volumineuse (rapport 2 au minimum). En effet, le même programme est stocké dans les mémoires programmes de chaque processeur, lesquels possèdent également un compteur ordinal. Ces mémoires programmes, ainsi que ces contrôleurs, ne sont pas associées à chacun des processeurs dans une solution SIMD, où il n'existe qu'une seule unité de contrôle.

I.2.3 : Les architectures mixtes.

Nous présentons ici les architectures parallèles asynchrones non strictement MIMD, les architectures contrôlées par les données (dataflow), et les architectures à la fois systoliques et dataflow, les "wavefront" processeurs.

Certaines architectures mélangent les fonctionnements en mode SIMD et en mode MIMD pour former soit des architectures de type maître/esclaves appelées "MSIMD", soit des architectures de type reconfigurable. En effet, l'ensemble synchrone, processeurs plus unité de contrôle, peut être dupliqué N fois afin de former une structure Multiple SIMD (MSIMD). Citons les machines ILLIAC IV, conçue de 4 processeurs synchrones ILLIAC de l'Université d'Illinois de type MSIMD, ainsi que les calculateurs PASM de l'Université de Purdue et TRAC de Texas Instrument, de type reconfigurable, pour lequel la structure est tantôt SIMD, tantôt MIMD. Ce type d'architecture, très attrayant, retrouve le problème des architectures MIMD, c'est-à-dire la "programmabilité" pour un nombre de noeuds asynchrones (processeurs MIMD) important, mais supportent un panel conséquent d'algorithmes.

Les architectures "dataflow" ne sont pas contrôlées par une unité spécialisée, mais sont directement contrôlées par les données elles mêmes. Une instruction ne peut s'effectuer que si ses opérandes sont prêts à être traités. Dans ces structures, la notion d'horloge de cadencement globale n'existe pas, si bien que les problèmes de synchronisation, de retards ... sont évités même si le nombre de processeurs est important. Deux grandes classes de calculateurs à flot de données existent : les architectures statiques à topologie figée, et les architectures dynamiques à topologie reconfigurable, en fonction et au cours de l'application [SRI86]. Ce type de calculateur permet de programmer des applications constituées de différentes tâches asynchrones, le dialogue et la synchronisation entre tâches s'effectuant grâce à des envois de données. Citons le processeur Manchester Data Flow Computer de l'Université de

Manchester, le MIT Tagged Token Data Flow architecture, tous les deux de type dynamique et le calculateur LAU, système développé à Toulouse, de type statique.

Les architectures "wavefront array" [KUN87, KUN88] sont un mélange des architectures systoliques et des architectures dataflow. Ainsi elles apparaissent comme des structures synchrones, éventuellement matricielles, semblables aux calculateurs systoliques, composées de processeurs et d'un réseau d'intercommunication permettant aux processeurs de dialoguer entre eux. C'est ce dialogue (les données), qui comme dans les architectures dataflow, gère le contrôle du programme, et non une horloge commune. Ce dialogue entre processeurs s'effectue grâce à un protocole bi-directionnel (hand shake), il permet donc la synchronisation : les processeurs signalent à leurs voisins, avec lesquels ils doivent échanger une donnée, leurs requêtes. Ceux-ci les autorisent à émettre dès qu'ils sont en attente de ces données. Par rapport aux architectures systoliques, les wavefront array processors en gardent les mêmes avantages, notamment au niveau des entrées/sorties, et peuvent intégrer un nombre de processeurs très important, du fait qu'il n'existe pas de signal de synchronisation commun. De plus leur programmation se fait en langage de haut niveau tel le Occam ou le MDFL, et la mise en oeuvre peut se faire à partir de processeurs du commerce tels le Transputer de Thomson-Inmos ou le µPD7281 de Nec.

I.2.4 : Conclusion.

Voici rapidement présentée la classification de Ducan permettant de comprendre les différents mécanismes de parallélisation existants. Le tableau I.1 classe les différents calculateurs parallèles présentés dans ce paragraphe. Dotées d'un de ces mécanismes, les architectures conçues gagnent en performances mais ces performances sont tributaires du couple application/architecture. Un calculateur parallèle, se réclamant de type générique, doit être accompagné d'un compilateur intelligent gérant au mieux les ressources disponibles sur la structure.

Le gain en performances d'un calculateur peut être considéré comme le produit des gains relatifs à trois critères :

- le degré de parallélisation possible c.a.d. le nombre de processeurs de la structure ;

- la puissance du compilateur c.a.d. le degré de parallélisme obtenu ;

• la technologie utilisée, par exemple l'apport de l'AsGa par rapport au CMOS [COR91], l'apport des connexions optiques [CRO91], voire même l'apport du tout optique [LOU91].

TYPE de CALCULATEUR		Exemples de Machines
SYNCHRONE	Vectoriels	CRAY1; CRAY XMP; FUTJITSU VP200
	Pipelines	Cyber 205; ETA-10
	Tableau SIMD Associatif	ILLIAC; Connexion Machine; Maspar; Loral Massively Parallel Processor; SYMPATI2 PEPE; STARAN
	Systoliques	WARP; SAXPY'S; MATRIX-1
ASYN	distribuée MIMD partagée	CM5; TNODE; VOLVOX; DADO2; NON-VON; Cosmic-Cube; NCube; CHIP ALLIANT FX18; BBN; NYU Ultracomputer.
MIXTE	MSIMD	ILLIAC IV;
	Reconfigurable	PASM; TRAC
	Data Flow	Manchester Data Flow Computer; LAU; MIT Tagged Token Data Flow Archi
	Wavefront	Royal Signal; Johns Hopkinsun; Standard Communication Compagny

***Tableau I.1** : Quelques exemples de calculateurs parallèles classés dans la taxonomie de Ducan.*

I.3 : PARALLELISATION DES OPERATIONS DE TRAITEMENT D'IMAGES.

Notre objectif est de disposer d'un système dédié au traitement d'images capable de supporter toutes les opérations de traitement d'images. Ce paragraphe présente les différentes approches permettant de concevoir une architecture dédiée au traitement d'images et totalement parallèle.

I.3.1 : Nécessité de la parallélisation.

La plupart des applications de traitement d'images exigent des temps de réponse rapides. A titre d'exemple, pour les opérations d'analyse d'images utilisées en contrôle industriel, la décision doit obligatoirement être prise en temps réel (fréquence de fabrication d'une pièce), en robotique, le temps de réponse du système doit être de l'ordre de la seconde, et pour les applications vidéo (simulateurs de vols, manipulation d'images, transmission vidéo...), la fréquence des traitements doit être la fréquence vidéo (50 Hz pour l'Europe). De plus, la quantité des points dans les images à traiter croit, d'images de 256×256 points codés en 256 niveaux de gris, on se dirige vers des images de 1024×1024 points codés en 3 couleurs chacune sur 256 niveaux, voire plus.

Ainsi pour un cas typique actuel, il est nécessaire de disposer d'un système capable de digérer 10^9 bits à la seconde. Cette capacité n'est exigée que pour la phase de traitement bas niveau où les opérations sont simples. Ensuite pour les opérations de moyen et de haut niveau, la bande passante nécessaire est plus faible, l'information étant condensée sous forme de symboles, mais les traitements sont plus complexes, et en grossière approximation, le produit de la bande passante par la complexité du traitement peut être vu comme une constante (figure I.3.1) [CHA90].

Afin d'accélérer toutes ces opérations, notamment celles de bas niveau, deux solutions architecturales s'opposent actuellement, l'une basée sur des circuits spécialisés effectuant chacun une opération particulière, l'autre s'appuyant sur un calculateur parallèle permettant d'effectuer tout ou partie des traitements. Nous présentons ici un comparatif de ces deux solutions. Toutefois, il faut signaler que certains processeurs parallèles rejoignent les processeurs câblés (exemple : processeurs systoliques), et que certains processeurs câblés font appel au parallélisme. Sont considérés ici comme processeurs câblés les circuits possédant au plus quelques étages de pipeline interne à architecture figée, et comme calculateurs parallèles, les structures de type vectoriel, de type SIMD ou de type MIMD.

En ce qui concerne les *performances*, il est difficile de faire une comparaison, car si ces performances sont connues et quasiment identiques pour les différents processeurs câblés, elles dépendent du nombre de processeurs mis en oeuvre pour les structures parallèles si bien que l'utilisateur adapte le nombre de processeurs à son application. Pour les processeurs câblés, la fréquence des traitements est comprise entre 10 Mhz et 50 Mhz, selon la complexité du circuit donc de l'algorithme mis en oeuvre. Cette fréquence correspond à la vitesse maximale d'entrée des points image dans le circuit, ainsi le temps d'exécution est égal au produit du nombre de points par la période du circuit associée à cette fréquence. Pour les

structures parallèles le temps d'exécution dépend du nombre de processeurs et de la longueur en micro-instructions de l'algorithme exécuté.

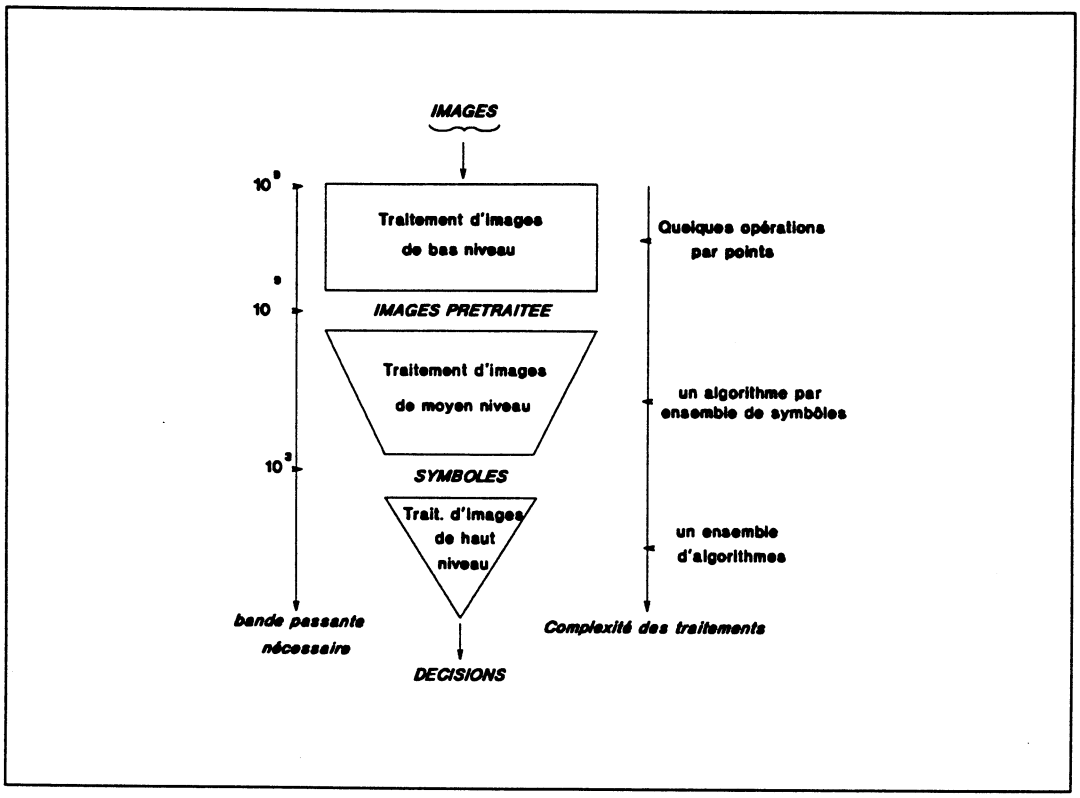

***Figure I.3.1** : Les différentes phases de l'analyse d'images. La bande passante nécessaire diminue alors que la complexité des traitements augmente lors de l'exécution des différents niveaux du traitement d'images.*

Le *coût* d'un processeur câblé est inférieur à celui d'une structure parallèle car la réalisation de cette dernière demande la mise en oeuvre de plusieurs modules matériels généralement identiques (d'autant plus que le nombre de processeurs du système est élevé).

La *mise en oeuvre* des circuits spécialisés est facile, car leur domaine de programmabilité est restreint et le modèle est souvent de type Von Neuman. Cette opération est plus complexe sur les processeurs parallèles car il est impossible de faire abstraction du modèle de la structure, excepté pour les calculateurs vectoriels, pour lesquels il existe des compilateurs de langage classique (Fortran, C...) effectuant la vectorisation de façon automatique. Toutefois, des langages de haut niveau tels le Occam, le C*, le MPL... facilitent la programmation des calculateurs parallèles [PAR92].

La *flexibilité* des structures parallèles est très grande par rapport à celle des circuits câblés. En effet pour ces derniers, le domaine d'application d'un circuit se réduit à quelques

algorithmes semblables (exemples convolutions avec masques différents), et pour mettre en oeuvre un algorithme d'un type différent, un circuit différent est nécessaire, tandis qu'une architecture parallèle supporte toute une gamme d'opérations de traitement d'images.

L'*intégration* des circuits spécialisés est facilement réalisable grâce aux circuits VLSI et ULSI (figure I.3.2). Pour les systèmes parallèles, les architectures très régulières et ne mélangeant pas les technologies, tels les processeurs systoliques, sont intégrables à très grande échelle en WSI. Par contre, les structures SIMD et MIMD mélangent des processeurs et des mémoires de technologies différentes, ce qui rend leur intégration en WSI délicate [PIR90], mais celle-ci est possible et réalisable grâce aux modules "multi-chip" (MCM : Multi Chip Module).

En ce qui concerne la *tolérance aux pannes*, critère souvent omis, elle n'est réduite qu'aux possibilités de la technologie utilisée pour les structures non parallèles, tels les circuits dédiés, tandis que pour les architectures parallèles, la tolérance aux pannes dépend et de la technologie et surtout, de la redondance offerte par l'ensemble des processeurs identiques mis en oeuvre, ce qui rend possible leur test fonctionnel. Ainsi, dans une structure SIMD par exemple, des séquences de test peuvent être envoyées sur les processeurs élémentaires, et les éléments défaillants sont détectés par vote majoritaire entre les processeurs.

Enfin les *entrées/sorties* d'un circuit spécialisé dépendent de la fréquence de fonctionnement du circuit, celui ci est connecté en ligne sur le signal vidéo, et ne mémorise que quelques lignes de l'image. Ce nombre de lignes représente le retard introduit par le circuit. En ce qui concerne les architectures parallèles, les images doivent être stockées dans les modules mémoire (distribuée ou partagée) de la structure. Le système dédié aux entrées/sorties doit donc être rapide et ne jamais ralentir la structure. Si le retard introduit par un circuit spécialisé n'est que de quelques lignes, il peut être de quelques images pour les structures parallèles SIMD ou MIMD (plus court pour les processeurs systoliques), généralement de 3 images (acquisition, traitement et restitution). Toutefois, si le système de traitement d'images est suffisamment générique (cf paragraphe suivant), ce n'est pas une image qui est restituée, mais un ensemble de symboles ou directement un résultat.

Le prix d'un système à base d'un processeur dédié est faible par rapport à celui d'un système parallèle, mais pour les applications de traitement d'images, il s'avère que de plus en plus d'algorithmes sont mis en oeuvre, donc plusieurs circuits dédiés doivent être envisagés, alors qu'un seul système parallèle peut effectuer tous ces calculs. Une solution, à base de circuits dédiés, devient donc rapidement plus onéreuse qu'un calculateur parallèle.

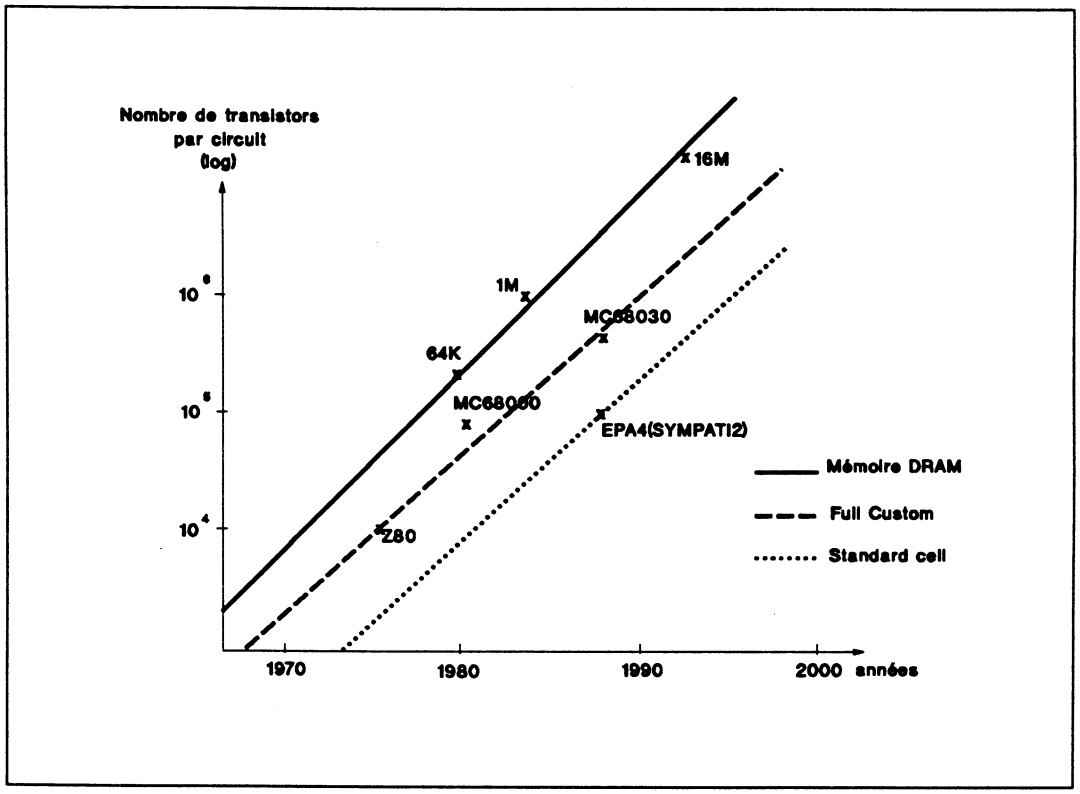

***Figure I.3.2** : Evolution des densités d'intégration des mémoires dynamiques (DRAM), des processeurs (et circuits à la demande "full custom") et des circuits précaractérisés ("standard cell"). Les mémoires voient leur densité d'intégration doubler tous les 2 ans (croissance de 1.5 annuelle) tandis que les circuits spécifiques (full custom et standard cell) ont une croissance annuelle de 1.35.*

Avant de présenter notre choix, il est nécessaire de le justifier par une présentation des besoins exigés pour des applications typiques de l'imagerie.

Dans le domaine du *contrôle industriel*, le coût des systèmes de traitement d'images est primordial. Précédemment, les sociétés commercialisant des cartes dédiées au contrôle de fabrication par la vision, proposaient des systèmes à base de circuits spécialisés (Matrox, Data Cube ...). Effectivement, cette solution est judicieuse si les traitements à effectuer restent invariables au cours du temps, mais s'avère obsolète pour des ateliers flexibles, où la chaîne de production doit pouvoir commuter rapidement la fabrication de différents produits, afin de répondre à la demande, et au "'zéro stock". Dans ce cas, il est préférable de disposer d'un calculateur parallèle pouvant recevoir les algorithmes adaptés à chaque type de pièces en fabrication, plutôt que d'avoir une grande quantité de circuits dédiés, solution volumineuse, "énergivore" et difficilement évolutive. De plus, si la production d'une pièce nécessite le développement d'un algorithme complexe, il est plus souple, plus rapide, et moins onéreux de développer un programme, même parallèle, qu'un nouveau circuit!

En ce qui concerne le domaine des *systèmes embarqués* dédiés à la vision, les circuits spécialisés conviennent aux systèmes "rudimentaires", c'est-à-dire les systèmes déroulant toujours les mêmes opérations. Pour ces applications, les structures parallèles ne peuvent concurrencer les architectures câblés. Toutefois, il est demandé aux systèmes embarqués de disposer d'une plus grande autonomie, autonomie obtenue grâce à des algorithmes complexes s'appuyant donc sur des structures programmables. Ainsi il est judicieux de penser que les structures parallèles équiperont et les robots et les missiles du futur. Notons également que robots et missiles évoluent généralement en milieu hostile (irradié), si bien que les capacités des structures parallèles au niveau de leur tolérance aux pannes pourront être utilisées.

Pour le domaine de la *génération graphique* et de la manipulation d'images, une structure parallèle permet aux utilisateurs de créer un nombre infini d'images, ce qui n'est pas possible avec des processeurs câblés. Notons également que les constructeurs de systèmes graphiques se mettent au parallélisme (Silicon Graphics), de même que les fondeurs de DSP (Texas Instrument avec le TMS320C40).

Suite à ces justifications en faveur du parallélisme, on peut se demander pourquoi il n'y a que très peu de calculateurs parallèles dédiés à la vision? A cette question il faut mettre en avant et la peur du parallélisme et la jeunesse de celui-ci. En effet, programmer une structure parallèle effrayait les utilisateurs il y a quelques années, mais le Transputer et son langage de programmation Occam, depuis 1988, ont bien popularisé cette discipline. Et il y a le coût, le volume et la consommation des systèmes parallèles qui compte tenu des évolutions de la technologie deviennent acceptables.

Pour ces différentes raisons, nous choisissons l'approche calculateur programmable parallèle dédié à l'imagerie, approche souple, prometteuse, et jeune car elle devient industrialisable. L'intégration d'un calculateur massivement parallèle est possible grâce à la technologie MCM (Multi Chip Module) permettant d'associer ensemble des puces de technologies différentes. Dans le paragraphe suivant, sont présentées les différentes approches possibles afin de paralléliser toute une chaîne de traitement d'images, de même que quelques calculateurs parallèles dédiés à la vision.

CRITERES	Processeurs Câblés	Processeurs Parallèles
PERFORMANCES (rapidité)	★★★	de ★★ à ★★★★
COUT	★★★★	★★★
MISE EN OEUVRE	★★★★	★★★
FLEXIBILITE	★	★★★★
INTEGRATION	★★★★	★★★
TOLERANCE aux PANNES	★★	★★★★
ENTREES/ SORTIES	★★★	★★★

***Tableau I.3.1** : Récapitulatif de l'adéquation des processeurs cablés et des processeurs parallèles pour des critères propres à la conception d'un systèmes électronique. (★ pas intéressant, ★★ peu intéressant,★★★ intéressant, ★★★★ très intéressant).*

I.3.2 : Les différentes approches.

Les opérations de traitement d'images se répartissent en trois classes : le bas, le moyen et le haut niveau. Pour les opérations de bas niveau, où une image est transformée en une autre, les algorithmes sont réguliers et se prêtent bien à une mise en oeuvre sur des calculateurs de type tableau de processeurs synchrones. Les opérations de traitement d'images de haut niveau, complexes et non régulières, sont, quant à elles, supportées par des structures parallèles asynchrones. Si les structures MIMD peuvent exécuter des opérations de traitement d'images de bas niveau, avec toutefois des résultats médiocres, il n'en est pas de même pour les structures SIMD, qui ne peuvent recevoir des algorithmes de haut niveau complets.

Quant aux opérations de traitement moyen niveau, leur parallélisation est délicate et donc rarement effectuée. Mais pour concevoir un système parallèle supportant toutes les opérations de traitement d'images, le moyen niveau doit être considéré.

S.L. Tanimoto propose dans [DUF86] quatre approches concurrentes, aboutissant à la réalisation de systèmes parallèles dédiés à la vision. Ces quatre approches (inhérente, unifiée, descendante et montante) sont présentées ci-dessous (figure I.3.3).

Pour l'*approche inhérente*, un type de calculateurs est associé par niveau de traitement, l'architecture globale se présente donc sous la forme d'une pyramide à trois étages, dans laquelle les informations (images, symboles) circulent. Le calculateur "Image Understanding Architecture" de l'université de Massashusset est basé sur ce principe. Il est donc composé de trois calculateurs, le CAAPP (Content Adressable Array Parallel Processor), matrice de 512×512 processeurs synchrones travaillant en SIMD, dédié aux opérations de bas niveau, le NPA (Numeric Processor Array), matrice de 64×64 processeurs de signal à mémoire distribuée de type MIMD, dédié aux opérations de traitement moyen niveau, et le GPPA (General Purpose Processor Array), matrice de 8×8 processeurs asynchrones de type MIMD également, supportant le haut niveau du traitement d'images [SHU88].

Dans la solution précédente, les trois calculateurs ne sont pas toujours utilisés à plein temps (problèmes des structures pyramidales), notamment s'il n'y a pas de pipeline d'images dans le système. Ce gâchis est évité dans l'*approche unifiée*. En effet, un seul calculateur parallèle effectue les trois niveaux du traitement d'images. Une même structure figée, de type MIMD, peut être utilisée pour arriver à réaliser les différentes opérations nécessaires, mais les performances notamment pour le bas niveau, sont médiocres. Aussi, plusieurs calculateurs parallèles à base de Transputers ont été développées sur ce modèle [COL88]. Une autre approche, est de créer une structure reconfigurable, de type SIMD à grain fin, (beaucoup de processeurs) en type MIMD à gros grain. Cette approche a été retenue par les concepteurs de la machine PASM dans laquelle les processeurs synchrones SIMD sont regroupables quatre à quatre, pour former un seul processeur asynchrone [KUE86].

D'autres calculateurs de type MIMD supportent les opérations de haut et de moyen niveau du traitement d'images, et s'appuient sur un autre calculateur parallèle de type synchrone pour effectuer les opérations de bas niveau, on parle alors d'*approche descendante*. Certaines architectures à base de Transputers, utilisent cette solution architecturale (Transvision, Atila [PIR90]). Dans Transvision, un ensemble de modules spécialisés (carte Data Cube) est connecté à un réseau de Transputers organisés en "ferme" de processeurs (structure maitre esclave") fonctionnant en mode SPMD [DER92].

Le pendant de la précédente approche, est l'*approche montante* où un calculateur parallèle est dédié au bas et au moyen niveau, le haut niveau étant supporté par une autre structure parallèle.

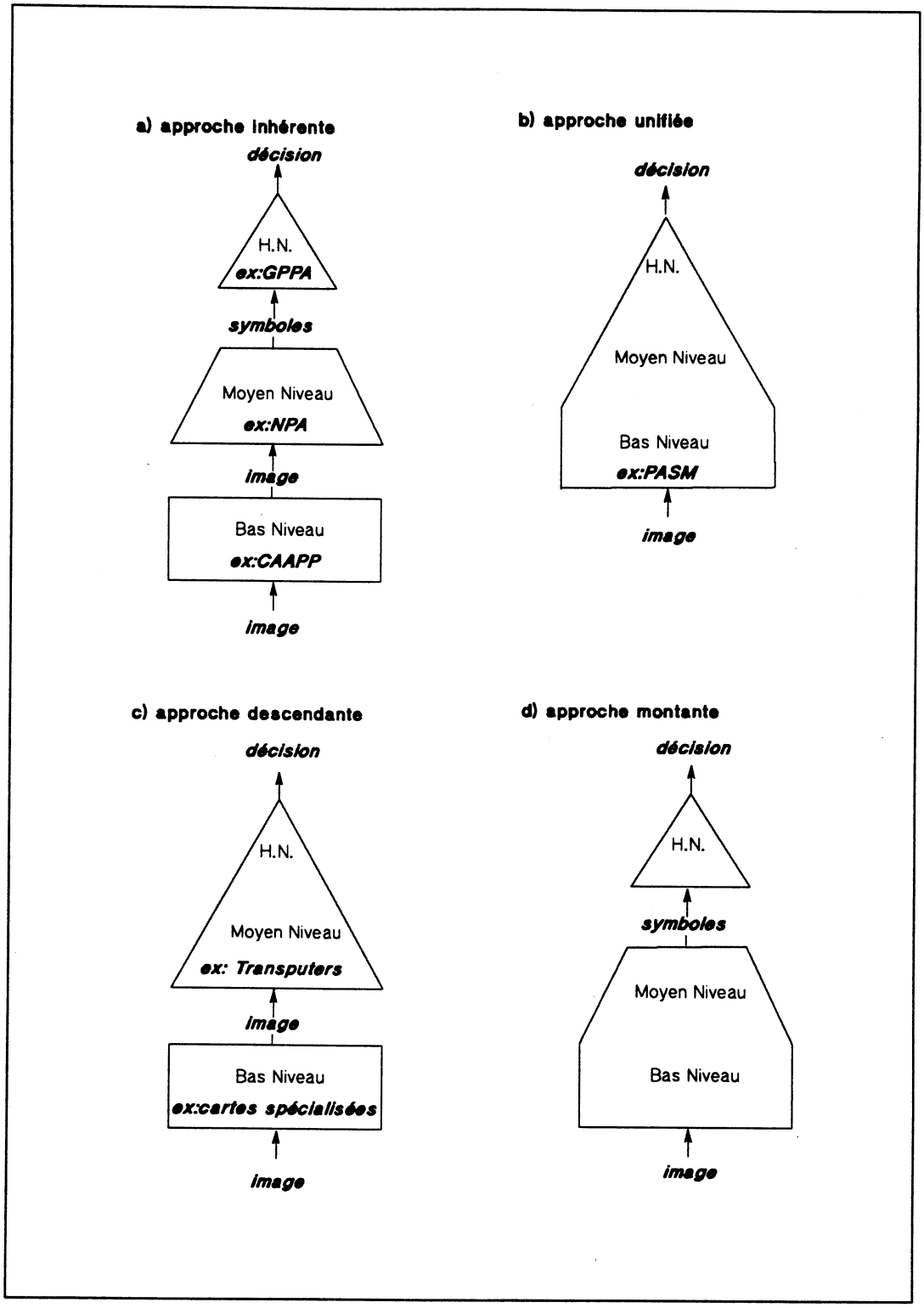

***Figure I.3.3** : Les 4 approches proposées par S.L. Tanimoto. Pour l'approche inhérente, la machine Image Understanding Architecture est citée en exemple, de même que PASM pour l'approche unifiée et Transvision pour l'approche descendante.*

De toutes ces approches, on ne peut dire qu'une est préférable aux autres, tout dépend des calculateurs mis en oeuvre. Par exemple, pour l'approche descendante (et inhérente), il est nécessaire de disposer d'un calculateur avec une bande passante importante car une image doit être transférée vers le(s) calculateur(s) de moyen et de haut niveau, ce qui n'existe pas pour les approches montante et unifiée. Certains calculateurs parallèles de type SIMD peuvent, suivant l'organisation des processeurs, être plus ou moins adaptés à certaines opérations, notamment celles de moyen niveau [CYP89, FOU86]. A titre d'exemple, une structure fortement distribuée, où il y a autant de processeurs que de points image, ne peut supporter efficacement (100 % de parallélisme) les opérations de traitement d'images de moyen niveau, car le nombre de symboles à traiter est bien inférieur au nombre de points image, les processeurs se trouveraient alors en surnombre. Notons également que le haut niveau demande parfois une ré-exécution d'opérations de bas et moyen niveau sur des zones particulières de l'image, il est donc nécessaire de conserver en mémoire les points image, tant que le traitement n'est pas terminé. Cette remarque est à prendre en compte pour les structures reconfigurables.

Développer une machine à base de calculateur(s) parallèle(s) dédiée à la vision demande un investissement très conséquent, et de nombreux laboratoires préfèrent ne réaliser qu'une brique de ces structures, en ne pensant malheureusement que trop rarement à leur intégration dans un système complet. Nous présentons dans le tableau I.3.2 les structures existantes en précisant leur domaine d'application pour le traitement d'images. Ce tableau, loin d'être exhaustif, confirme, au vu des exemples, que la parallélisation des opérations de traitement bas niveau se fait efficacement par des structures synchrones, généralement de type SIMD, les traitements moyen et haut niveau s'effectuant sur des structures non synchrones.

De nombreuses structures dédiées au traitement d'images de bas niveau ont été développées, mais leur interconnexion avec un système de haut niveau pose des problèmes, car il est nécessaire de restituer en sortie toute l'image, ce qui exige une bande passante importante. Ainsi les structures exécutant les algorithmes de traitement bas et moyen niveau nous semblent préférables de par la compression de données effectuée lors de l'exécution des opérations de traitement de moyen niveau.

Traitements	Architectures Synchrones	Architectures Asynchrones	Architectures Mixtes
Bas Niveau	GAPP; DAP; CAAPP; ILLIAC; MPP; CLIP4; AIS-5000; PICAP3; SYMPATI2; PIPE	Multi DSP	WARP; DIP; Cyto-HCS
Moyen Niveau	ISMAP; SNAP	NPA	
Haut Niveau	GPPA;	Transputers	
BN & MN			SPHINX; PAPIA
MN & HN		Transvision(Transputer); Multi-cluster; Multi-DSP	
BN & MN & HN		Transputer	PASM; Cosmic Cube; NCube; GFLOPS

Tableau I.3.2 : Domaine d'application des architectures parallèles existantes par rapport aux différents niveaux du traitement d'images.

I.3.3 : Conclusion.

Sont exposées dans ce paragraphe les raisons du choix du parallélisme, de même que les différentes approches possibles afin de paralléliser toutes les opérations de traitement d'images.

Par rapport aux processeurs dédiés, les architectures parallèles présentent de nombreux avantages tels que, leur flexibilité, leur domaine d'application, leur modularité, et leur tolérance aux pannes. Ces structures équiperont tous les systèmes de vision de demain, que ce soit en contrôle industriel, en navigation ou en graphisme et seront d'autant plus utilisées et utilisables qu'elles deviendront programmables en un langage de haut niveau.

En ce qui concerne les différentes approches proposées pour paralléliser les opérations de traitement d'images, il est important de noter que la parallélisation des opérations de traitement de haut niveau est difficile, et peu de structures effectuent actuellement cette tâche.

Par contre, de nombreuses structures, essentiellement de type synchrone SIMD, sont proposées pour paralléliser les opérations de traitement bas niveau, mais de bas niveau uniquement ce qui les rend parfois difficilement intégrables dans un système de traitement d'images complet. Il est important de noter que la phase de traitement d'images moyen niveau constitue l'étape charnière de la parallélisation des opérations de traitement d'images, car elle modifie la structure des données, une image est transformée en tables, opération s'avérant délicate à effectuer sur des calculateurs parallèles distribués, car toutes les données changent d'emplacement mémoire ce qui constitue un important goulot d'étranglement.

I.4 : CONCLUSION.

Les opérations de traitement d'images se répartissent en 3 classes, le bas niveau caractérisé par une régularité spatiale et une simplicité des traitements, le moyen niveau où les données arrangées sous forme d'image (matrice 2D) sont réorganisées en un autre format, et le haut niveau caractérisé par des traitements complexes.

De nombreux calculateurs parallèles ont été développés pour supporter les opérations de traitement d'images de bas niveau, peu de structures proposent la parallélisation des trois niveaux. Après ces présentations de l'état de l'art de la parallélisation des opérations de traitement d'images, nous présentons dans la suite de cette thèse, une architecture parallèle permettant de supporter toutes les opérations de traitement d'images.

Chapitre II

Evolution de l'Architecture de SYMPATI2.

Dans le chapitre précédent, sont présentées les différentes opérations de traitement d'images, les différents types de calculateurs parallèles, ainsi que les intérêts, et les méthodes de la parallélisation des différentes phases du traitement d'images. Notre laboratoire a développé, en collaboration avec l'IRIT (Université de Toulouse), un calculateur parallèle dédié à l'imagerie, appelé SYMPATI2 (SYstème Multiprocesseur Adapté au Traitement d'Images). Cette structure de type synchrone SIMD est décrite dans ce chapitre, ainsi que ses performances et ses limites. Pour supprimer ses limites, une nouvelle structure, SYMPATIX, est proposée suite à une étude algorithmique présentée également dans ce chapitre.

II.1 : SYMPATI2.

SYMPATI2, pour SYstème Multiprocesseur Adapté au Traitement d'Images, est un calculateur parallèle de type SIMD, opérationnel depuis Mars 1990, après 4 années d'études incluant la réalisation d'un circuit VLSI (figure II.1.1). Le but de ce produit est de proposer un système dédié au traitement d'images de bas, et éventuellement de moyen niveau, à un coût acceptable, afin d'être substitué aux systèmes à base de circuits spécialisés.

Actuellement le produit est en cours de transfert industriel, et sera commercialisé dès le premier trimestre 1993 par la société Centralp Automatismes [CEN92]. Sont présentées dans ce paragraphe, sous forme de synthèse, l'architecture matérielle et logicielle de SYMPATI2, ainsi que ses performances, synthèse construite à partir de [BAS85, ESS88, JUV88].

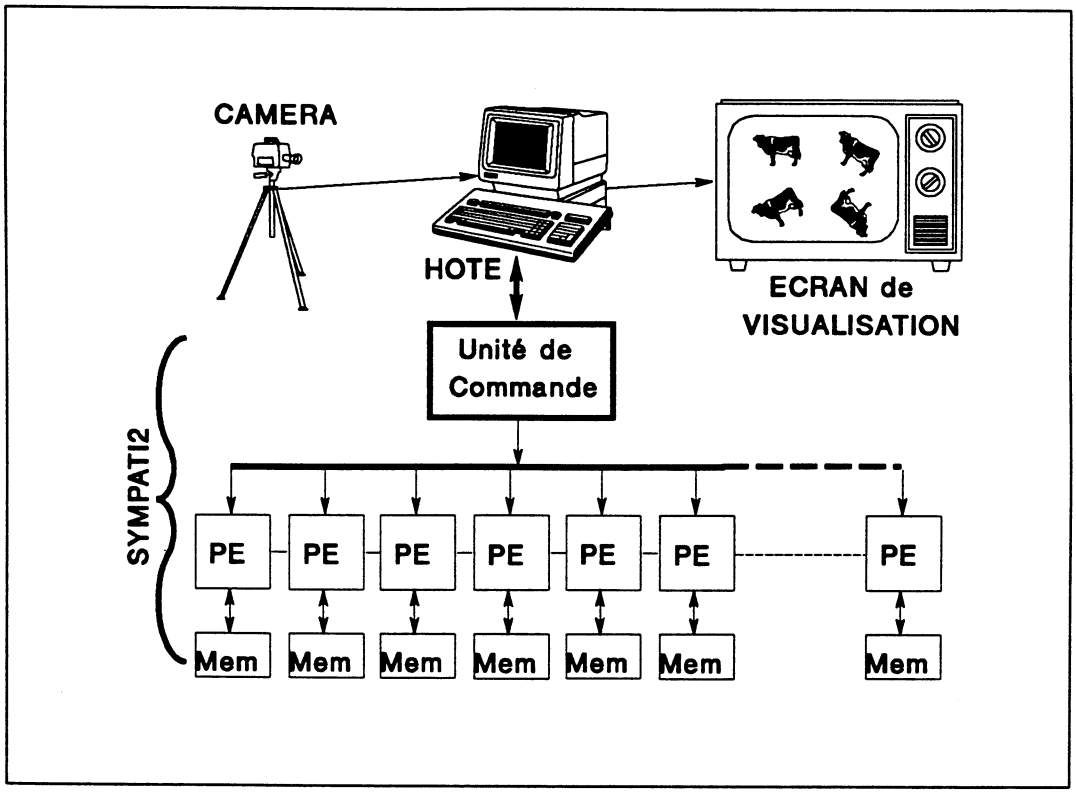

***Figure II.1.1** : Le système SYMPATI2 dans son environnement. Chaque "PE" représente un Processeur Elémentaire, le nombre de ces PEs peut atteindre 256. Les images traitées peuvent être visualisées en temps réel.*

II 1.1 : L'architecture de SYMPATI2.

SYMPATI2 est une structure *synchrone* de type *SIMD*, où les images sont *réparties* de façon *hélicoïdale* sur les processeurs, organisés en *ligne*. Les justifications de ces choix sont présentées ci-dessous.

Tout d'abord, de par la régularité des traitements mis en oeuvre lors des opérations de traitement d'images de bas niveau, et de par le nombre important de points dans une image, il est préférable d'utiliser une structure *synchrone*, permettant d'aboutir à un calculateur à grain plus fin que ce qu'il n'est possible d'obtenir avec une structure asynchrone (à volume électronique identique). En effet, les structures synchrones ne comportent qu'un seul compteur ordinal, ce qui n'est pas le cas des structures asynchrones, où il y a autant de compteurs de programmes et de mémoires programme que de processeurs.

Parmi les structures synchrones, le type *SIMD* est préférable au type pipeline vectorisé, mal adapté aux opérations de traitement d'images bas niveau, et au type systolique trop spécialisé, offrant donc un domaine de mise en oeuvre des opérations de bas niveau très

restreint. Dans ce mode SIMD, les processeurs peuvent ainsi traiter chacun une partie de l'image qui leur est propre. Il y a autant de parties que de processeurs, et l'ensemble de ces parties forme l'image à traiter. Le même algorithme est déroulé sur tous les points de l'image et donc sur toutes les parties de l'image, on a bien tous les processeurs effectuant la même opération sur des données (points) différentes, d'où le mode SIMD.

Deux possibilités existent pour répartir les points d'une image sur une structure parallèle. Soit les processeurs sont distribués sur l'image, dans ce cas ces processeurs voient une sous-image de l'image complète, il y a autant de sous-images que de processeurs, soit les images sont distribuées sur les processeurs, ou plutôt sur les mémoires associées aux processeurs, dans ce cas la matrice de processeurs est appliquée sur l'image autant de fois que nécessaire afin de la décrire entièrement (figure II.1.2). Ici, les points voisins sur l'image ne sont pas contenus dans le même processeur. Cette solution où l'*image est distribuée sur les processeurs* a été retenue pour SYMPATI2, car elle évite les problèmes dus aux effets de bords rencontrés aux frontières des sous-images dans la première solution, permet de paralléliser des algorithmes récursifs et facilite les opérations d'entrées/sorties, les points voisins d'une ligne sont stockés dans des processeurs différents. En contre-partie, cette solution, où les points voisins sont stockés dans des processeurs voisins, demande un fort couplage entre les processeurs afin de ne pas pénaliser la structure pour les opérations de voisinage, et demande également un style de programmation particulier. Le couplage s'effectue par un réseau d'interconnexion.

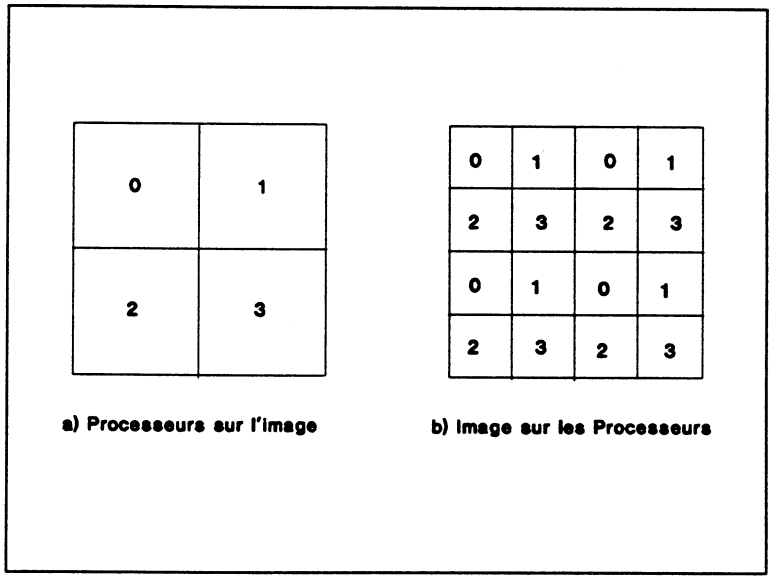

***Figure II.1.2** : Répartition des processeurs sur l'image (a) et répartition de l'image sur les processeurs (b). La taille de l'image est de 4×4 points et le nombre de processeurs est de 4 (0, 1, 2 et 3).*

En ce qui concerne l'organisation des processeurs, à savoir la topologie du réseau d'interconnexion, plusieurs solutions sont envisageables pour de telles structures. La plus fréquente est l'organisation des processeurs en deux dimensions, la structure obtenue colle alors au mieux à l'image, celle-ci étant organisée de la même façon. Cette structure organise les processeurs du calculateur MPP (à répartition de l'image sur les processeurs) et du calculateur DAP (à répartition des processeurs sur l'image). Ces structures, parfaitement adaptées aux opérations de bas niveau (traitements iconiques), ne possèdent qu'une modularité en M^2, M étant la largeur du côté de la matrice de processeurs supposée carrée. Avec une telle modularité, le rapport coût/performances n'évolue pas de manière graduelle, ce qui ne convient pas à un calculateur qui se doit d'être industriel, tel SYMPATI2. Il doit être doté d'une modularité plus fine, linéaire par exemple, afin de coller au mieux aux exigences du client en lui offrant toute une gamme de taux coût/performances. Une modularité linéaire est obtenue avec *une organisation linéaire* des processeurs, organisation retenue sur SYMPATI2. Une comparaison des structures multiprocesseur SIMD en ligne et en tableau, indique que les processeurs lignes présentent sensiblement les mêmes capacités que les processeurs en tableau 2D et effectuent les entrées/sorties plus rapidement que ces derniers [JON90].

Dans un calculateur ayant ses processeurs élémentaires organisés en ligne, un processeur pourrait contenir une colonne (ou une ligne) de l'image dans sa mémoire, mais une telle solution conduirait à une perte complète du parallélisme pour des balayages de l'image avec le barreau de processeurs à la verticale (respectivement à l'horizontal). En effet, toutes les données se trouveraient alors dans le même processeur, celui possédant la colonne (respectivement la ligne) en cours de traitement, il ne serait donc possible d'accéder aux points à traiter qu'un à un. Plutôt que de s'interdire cette possibilité de balayage fort utile, le mode de *rangement hélicoïdal* a été préféré. Son principe est décrit ci-dessous :

une image organisée classiquement de la façon suivante :

...		
(i-1,j-1)	(i-1,j)	(i-1,j+1)
(i,j-1)	(i,j)	(i,j+1)
(i+1,j-1)	(i+1,j)	(i+1,j+1)
...		

est transformée en une image à structure hélicoïdale organisée comme ceci :

...				
(i-1,j-1)	(i-1,j)	(i-1,j+1)		
(i,j-2)	(i,j-1)	(i,j)	(i,j+1)	
(i+1,j-3)	(i+1,j-2)	(i+1,j-1)	(i+1,j)	(i+1,j+1)
...				

Ainsi, pour une simple image composée de 4×4 points, la répartition des 16 points image sur une structure composée de 4 processeurs est la suivante :

image de départ				image hélicoïdalisée				Adresses
(0,0)	(0,1)	(0,2)	(0,3)	(0,0)	(0,1)	(0,2)	(0,3) . . .	0
(1,0)	(1,1)	(1,2)	(1,3)	(1,3)	(1,0)	(1,1)	(1,2) . . .	1
(2,0)	(2,1)	(2,2)	(2,3)	(2,2)	(2,3)	(2,0)	(2,1) . . .	2
(3,0)	(3,1)	(3,2)	(3,3)	(3,1)	(3,2)	(3,3)	(3,0) . . .	3
				PE0	PE1	PE2	PE3	

Les 4 processeurs, PE0... PE3 possèdent en mémoire les points des colonnes de l'image à structure hélicoïdale. Ces points sont stockés aux adresses correspondantes. Ainsi les verticales (j=constante) sont réparties sur les 4 processeurs, permettant l'exécution d'algorithmes par balayage de l'image en sens horizontal et en sens vertical. Par contre, cette solution demande un module de calcul d'adresses particulier (non linéaire) car, à titre d'exemple, les points d'une même verticale sont stockés à des adresses différentes suivant la position du processeur. L'adresse est obtenue par l'équation suivante [ESS88] :

$$Ad=((NC/M) \times i + j) \qquad (II.1.1.1)$$

où Ad représente l'adresse mémoire de stockage relative à un processeur, M le nombre de processeurs, NC le nombre de colonnes du plan image, i et j les coordonnées du point à stocker. Les processeurs stockent en leur mémoire des points différents, si bien que ce calcul doit être réparti sur tous les processeurs. Dans les processeurs de SYMPATI2, il existe donc un module spécialisé effectuant à chaque cycle cette opération de calcul d'adresses. Outre ce mode d'adressage hélicoïdal permettant au barreau de processeurs de décrire les images à traiter, l'adressage des mémoires peut se faire également en indexé : chaque processeur dispose d'un registre d'index utilisé pour ce mode d'adressage, appelé adressage tabulaire.

Au côté de ce module d'adressage (figure II.1.3), se trouve l'Unité Arithmétique et Logique (UAL) contenue dans chacun des processeurs. La précision de cette UAL est entière de 16 bits, elle est épaulée d'un multiplieur 8 bits et d'un circuit effectuant les décalages. En interne de chaque processeur, se trouve un ensemble de registres appelé "scratch pad"

contenant 8 registres de 8 bits et 2 registres de 16 bits. Les registres de 8 bits sont concaténables en 4 registres de 16 bits.

Le réseau d'interconnexion permet à chaque processeur de dialoguer avec ses voisins distants de 1, 2 ou 3 à gauche ou à droite. Les dialogues, ou plutôt les lectures, à distance 1 et 2 à gauche ou à droite sont permises, soit dans le scratch pad du processeur lu, soit dans sa mémoire associée, tandis que les lectures à distance 3 ne s'effectuent uniquement que dans le "scratch pad". Pour des distances supérieures, plusieurs cycles sont nécessaires. L'UAL est de type "orientée accumulateur", c'est-à-dire que pour les opérations de 2 opérandes, un des opérandes est l'accumulateur appelé L (registre 16 bits), l'autre étant un point mémoire ou registre accessible par le réseau. Le stockage des résultats ne peut se faire qu'en local, à savoir dans l'accumulateur L, dans le scratch pad ou dans la mémoire associée (dans tous les cas distance 0). En interne du processeur, la largeur du chemin de données est de 16 bits, en externe que ce soit entre processeurs ou mémoires, il est de 8 bits. Pour cette structure SIMD les échanges entre processeurs s'effectuent de façon synchrone, à savoir quand un processeur dialogue avec son voisin gauche distant de 2 par exemple, tous les autres processeurs effectuent le même échange avec leur voisin gauche distant de 2 respectif. Grâce à ce réseau, tous les points image nécessaires à l'exécution d'un filtre 3×3 sont accessibles chacun, par les processeurs, en un seul cycle.

Dans le but de rendre plus souple notre structure et afin d'adoucir les contraintes d'une programmation SIMD brute, deux atouts viennent compléter le processeur. Tout d'abord, les processeurs ont la possibilité de s'auto-inhiber grâce à 4 indicateurs présents dans chacun des processeurs. Ces indicateurs peuvent, au gré du programmeur, prendre la valeur des drapeaux de l'UAL (zéro, carry, signe...). Ainsi, les processeurs peuvent s'auto-contrôler par leurs données. Ceci permet de ne faire travailler que les processeurs devant effectuer une série d'opérations. Ensuite, afin d'éviter d'effectuer des parties de programme dans lesquelles aucun processeur ne travaillerait, il est possible d'insérer dans le programme, des instructions GOTO, "non strictement SIMD". Certes ce goto est conditionné par l'état des indicateurs et ne peut s'effectuer que lorsque tous les processeurs ont l'indicateur de conditionnement dans le même état, ou lorsque l'indicateur de conditionnement d'au moins un processeur est dans l'état différent des autres. Ceci est rendu possible par le fait que tous les processeurs retournent l'état de leurs indicateurs à l'Unité de Contrôle, cellule déroulant les programmes en gérant la structure. Détaillons rapidement les principales fonctions de cette UC.

Le rôle de l'unité de contrôle est de manager tous les processeurs, ou unités de traitement. Elle est constituée de deux modules, un module de contrôle et un module

d'entrées/sorties. Le module de contrôle est chargé de distribuer, vers les processeurs, les micro-instructions à exécuter, gère le déroulement du programme en cours d'exécution (gestion des goto...), et gère également le balayage des images ce qui le rend transparent vis-à-vis de l'utilisateur. Le module d'entrées/sorties assure le transfert et la récupération des images distribuées sur les processeurs. La réalisation de ces différents modules est décrite dans [SCH91].

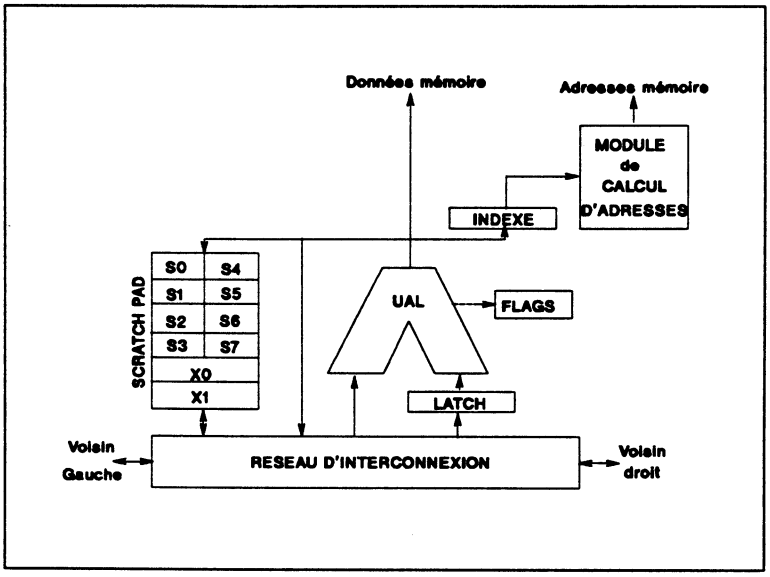

***Figure II.1.3** : L'architecture de l'unité de traitement (ou processeur élémentaire) de SYMPATI2. L'unité arithmétique et logique (UAL) contient également un multiplieur. Le chemin interne des données est de 16 bits, les échanges avec les voisins gauches et droits s'effectuant sur un format de 8 bits.*

La programmation de la structure de SYMPATI2 se décompose en deux phases, la configuration de l'unité de contrôle, et la programmation des processeurs. Lors de la configuration de l'unité de contrôle, les paramètres statiques nécessaires à la gestion du balayage de l'image et l'initialisation des processeurs sont définis. Quatre types de balayages sont possibles, soit linéaire, soit par bandes, avec pour ces deux cas un segment soit vertical, soit horizontal, avec une kyrielle de configurations possibles, choisies par l'utilisateur (traitement d'une ligne sur 2 par exemple). La programmation des processeurs s'effectue en 4LP (Low Level Language for Line Processor), assembleur de haut niveau [ADM88]. L'utilisateur écrit son programme en raisonnant au niveau d'un point de l'image à traiter, et c'est l'unité de contrôle qui ensuite, lors de l'exécution du programme, réitère les instructions autant de fois que nécessaire, en calculant automatiquement tous les paramètres utilisés par les processeurs pour qu'ils adressent les bons points image. Des travaux sont en cours dans

notre laboratoire concernant la programmation de la structure en un langage de haut niveau à parallélisme de données, le T++ [ESS92].

La réalisation de SYMPATI2 a conduit à fabriquer un ASIC intégrant 4 processeurs élémentaires, appelé EPA4 [HAN91]. Ce circuit fondu chez AT&T est de type standard cell 1.2 µm CMOS et intègre un équivalent à 90 000 transistors. Les mémoires associées aux quatre processeurs intégrés se connectent sur EPA4, et actuellement la maquette, présente dans notre laboratoire, contient 32 processeurs, soit 8 EPA4, et 32 boîtiers mémoire, l'ensemble occupant une carte au format triple Europe (280mm×370mm). L'unité de contrôle, quant à elle, n'est intégrée actuellement que par des produits programmables (PLD) et occupe la même surface que la carte 32 processeurs. Cette unité est en cours d'intégration en vue de la commercialisation du produit.

II 1.2 : Les Performances de SYMPATI2.

Bien évidemment, les performances de SYMPATI2 dépendent du nombre de processeurs mis en oeuvre, et pour de nombreux algorithmes, elles croissent linéairement avec ce nombre. La structure, de par sa régularité, est bien adaptée aux opérations de traitement d'images de bas niveau, où une image est transformée en une autre. Le système a été évalué en 1989, alors qu'il n'existait pas encore sous forme de maquette, sur un exemple représentatif des opérations de traitement bas niveau, extraction du squelette d'une croix noyée dans un bruit gaussien. Les résultats publiés dans [PRES89] classent SYMPATI2 (solution à 128 processeurs) en première position avec la CAAPP (Content Adressable Array Parallel Processor) de l'université de Massachussetts (Matrice de 512×512 processeurs). La mise en oeuvre de SYMPATI2 est venue valider les estimations faites pour ce benchmark. Le tableau II.1.1 présente quelques résultats représentatifs du traitement d'images obtenus sur SYMPATI2. Notons que cette liste ne peut être exhaustive car la structure est programmable et rend cette liste infinie.

A titre de comparaison, la société LSI Logic propose dans [LSI92], un circuit dédié, le L64243, en technologie 1.5 µm HCMOS, effectuant la convolution d'une image par un masque 3×3 quelconque à la fréquence pixel de 40 Mhz. Ainsi pour une image de 256×256 pixels, cette convolution s'effectue en 1,6 ms. SYMPATI2, doté de 128 processeurs, effectue cette opération plus rapidement au prix d'une électronique plus conséquente, mais offrant un domaine d'application bien plus important. En effet SYMPATI2 peut effectuer toute les opérations pré-citées (et bien d'autres), alors qu'il est nécessaire de se doter d'autant de circuits que d'opérations pour une solution à base de processeurs spécialisés.

OPERATIONS	32 PEs	128 PEs
Convolutions 3×3	4ms	1ms
Convolutions 5×5	17ms	3.75ms
Opérations logiques (ET, OU, SEUIL...)	1ms	0.25ms
Morphologie Mathématique (érosion, dilatation)	3ms	0.75ms
Calcul d'histogramme	10ms	2.5ms
égalisation d'histogramme	4.5ms	2.5ms
Transformée de Hough (256 directions)	1.2s	0.5s
Rotations (angle quelconque)	12.5ms	8ms

***Tableau II.1.1** : Quelques résultats d'opérations de traitement d'images obtenues sur SYMPATI2, configuré avec 32 Processeurs Elémentaires (PEs) ou 128. Tous les résultats sont donnés pour des images de 256×256 points codés sur 8 bits. Excepté pour la rotation d'image et la transformée de Hough, les performances croissent linéairement avec le nombre de processeurs. Ce nombre peut aller jusquà 256. Ces résultats sont à comparer avec ceux proposés dans [SLE88] obtenus par une matrice de 4×4 Transputers. A titre d'exemple, cette matrice effectue un convolution 3×3 en 144 ms.*

Ainsi la structure proposée offre des performances aussi attrayantes que les circuits dédiés, au prix d'une électronique plus conséquente, mais dotée d'une flexibilité plus grande. Si pour les opérations présentées ci dessus, le parallélisme y est très fréquemment de 100%, il n'en est pas de même pour les opérations de traitement moyen niveau.

II.1.3 : Choix de l'approche et conclusion.

Quelle solution architecturale adopter pour intégrer SYMPATI dans une chaîne complète de traitement d'images? Des 4 approches proposées par Tanimoto (paragraphe I.3.2), laquelle choisir?

Notre processeur dispose d'une UAL de bonne précision (16 bits) pour les traitements au niveau point, il serait judicieux d'utiliser cette UAL pour effectuer les opérations de traitement moyen niveau. Dans ce cas, la précision de cette UAL sera-t-elle suffisante?

La granularité d'un calculateur parallèle se mesure au nombre d'unités de traitement mises en oeuvre. Aussi, les architectures parallèles dédiées au bas niveau sont généralement

très distribuées, il y a autant de processeurs que de pixels à traiter, on parle alors de stucture à grain très fin. Ces architectures ne peuvent, si elles restent dans la même configuration, effectuer efficacement les opérations de moyen niveau, car le nombre de symboles est inférieur au nombre de points dans l'image, donc de processeurs, d'où perte de parallélisme. L'architecture de SYMPATI2 est moyennement distribuée et à grain moyennement fin, elle semble donc capable de supporter, avec un bon taux de parallélisme, les opérations de moyen niveau.

Enfin, des programmes SIMD non réguliers peuvent être supportés par la structure grâce à trois caractéristiques de SYMPATI2 : le masquage des processeurs, l'adressage indexé et les GOTOs conditionnels.

Tous ces arguments nous laissent à penser que le bas niveau et moyen niveau peuvent être supportées par SYMPATI2, ou une nouvelle structure à base de SYMPATI, ce qui repoussent l'approche inhérente et l'approche descendante. On a vu que le haut niveau ne peut être exécuté que sur des structures asynchrones, alors soit la possibilité de se reconfigurer est offerte à SYMPATI, on se dirigerait alors vers l'approche unifiée, soit ce niveau est soumis à un autre calculateur, on parlerait alors d'approche montante.

Pour la reconfiguration de SYMPATI, une solution est de transformer SYMPATI2 en une structure MIMD dotée de processeurs pocédant des unités de traitement ayant une grande précision, ceci afin de supporter les opérations de traitement d'images de haut niveau, et de part la même, toutes les opérations de traitement d'images. Au niveau matériel, la solution est de regrouper entre eux quelques processeurs élémentaires actuels (4 par exemple) pour former un processeur autonome et doté d'une UAL de bonne précision (flottant). Une structure MIMD à gros grain peut alors être formée, à la condition d'associer à chacun de ces grains MIMD une unité de contrôle. Une telle solution aboutit à un temps de cycle de ce nouveau processeur long par rapport à ce que l'on peut obtenir avec les processeurs du marché (T9000 de Thomson-Inmos, Alpha de DEC...). A titre d'exemple, si les processeurs actuels de SYMPATI2 sont regroupés par paquets de 4 pour former des processeurs plus performants travaillant en flottant, la puissance estimée serait de 0.5 MFlops, 50 fois moins rapide que le T9000 doté de 25MFlops. De plus une telle solution exigerait de développer un compilateur de langage de haut niveau, outil indispensable à ce niveau, de développer des protocoles de dialogue entre les processeurs asynchrones... ce qui nous conduirait à "réinventer l'eau tiède"! Toutefois le seul argument en faveur de cette solution pouvant être considéré, est qu'une telle structure conduit à un système peu encombrant, donc peu "énergivore", critère primordial pour les systèmes embarqués. Mais attention, nous avons dit que le traitement d'images haut

niveau demande la ré-exécution des opérations de bas et moyen niveau sur tout ou partie de l'image, si bien que le système doit se reconfigurer plusieurs fois, la reconfiguration doit donc être rapide, et surtout les données pixels doivent être conservées. Il apparaît ainsi judicieux de séparer le calculateur de haut niveau du calculateur dédié au bas et au moyen niveau. De plus l'utilisation de processeurs du marché, très efficaces, conduit à n'en mettre en oeuvre qu'un petit nombre, nombre forcément bien inférieur à celui nécessaire pour les processeurs reconfigurés (bricolés), atout non négligeable quand on connaît les difficultés de programmabilité d'une structure multiprocesseur asynchrone, difficultés proportionnelles au nombre de processeurs mis en oeuvre.

Ainsi nous retenons **l'approche montante** pour notre structure, dans la mesure de sa faisabilité. Le but du paragraphe suivant est de montrer les limites de SYMPATI2 pour les opérations de traitement moyen niveau, afin de proposer, si cela est possible, une nouvelle structure parfaitement adaptée au traitement d'images bas et moyen niveau, et facilement interfaçable à un processeur de haut niveau.

II.2 : SYMPATI2 ET LE TRAITEMENT MOYEN NIVEAU.

Pour utiliser SYMPATI2 dans une approche montante, il est nécessaire d'évaluer cette structure dédiée au bas et moyen niveau, pour les opérations de traitement d'images moyen niveau, le bas niveau étant bien supporté par notre calculateur. Une telle méthodologie de travail nous a permis de mettre en avant les faiblesses de SYMPATI2 vis à vis de ces opérations, à la suite de quoi il fut envisageable d'apporter des modifications à la structure actuelle.

Nous ne présentons pas l'étude de la parallélisation de tous les algorithmes de traitement d'images de moyen niveau, mais de quelques uns les plus représentatifs de ce niveau. Ces algorithmes se répartissent en trois classes, suivant qu'ils travaillent sur des points, des contours, ou des régions.

II.2.1 : Les Algorithmes Considérés.

Peu d'algorithmes travaillent sur les points uniquement, nous n'en n'avons retenu qu'un seul permettant d'isoler des points caractéristiques d'une image. Le traitement très simple se décompose en deux phases, une première travaillant au niveau iconique, pendant laquelle un

balayage complet de l'image est effectué et où les points caractéristiques à détecter sont marqués. Lors de la seconde phase, les points caractéristiques du même type, sont envoyés sur le même processeur, il y a donc autant de processeurs réceptifs que de types de points. A titre indicatif, les points anguleux peuvent constituer un type. Cet algorithme est simple mais reste très représentatif de la transformation d'un format de données iconique vers un format symbolique.

En ce qui concerne les algorithmes travaillant sur les contours, ils sont de différents types suivant qu'ils travaillent sur les points de contours pris en vrac (Transformée de Hough), ou qu'ils travaillent sur des ensembles de points appartenant à un même contour. Dans ce cas une première phase des traitements est d'effectuer ces regroupements. C'est le cas des codages de contours tels, le codage de Freeman, le codage par approximation polynomiale, la transformation en psi-curves etc. Contrairement à ce qui est proposé dans [LIT89] et [CHAa90], nous classons la transformée de Hough comme opération de moyen niveau et non de bas niveau. En effet, elle donne les équations des courbes représentatives de l'image, ce qui correspond bien à un changement de structure de données.

Enfin les algorithmes choisis travaillant sur les régions sont l'étiquetage d'une image et la division/fusion. L'étiquetage de régions d'une image binaire, regroupe entre eux les points appartenant à une même région, appelés points connexes. Le rôle de l'algorithme de division/fusion est le même mais il travaille sur des images en niveaux de gris et le critère de connexité est défini par l'utilisateur et se détermine par rapport à un seuil.

POINTS	Extraction de points caractéristiques;
CONTOURS	Codage de Freeman; Codage par approximation polynomiale; Codage par transformation en psi-curves; Codage par polylines; Transformée de Hough;
REGIONS	Etiquetage; Division/fusion.

Tableau II.2.1 : *Les différents algorithmes étudiés. Ils travaillent sur des points, des contours ou des régions.*

II.2.2 : Les problèmes Rencontrés.

Les solutions proposées pour paralléliser les algorithmes pré-cités sont détaillées en Annexe A1 et dans [COLa91]. Cette étude permet de dresser la liste des problèmes rencontrés lors de la parallélisation des opérations de traitement moyen niveau sur SYMPATI2.

Les algorithmes étudiés sont plus ou moins facilement parallélisables, l'annexe A1 nous présente des solutions de parallélisation sur des structures de type SIMD. Si le parallélisme total est aisément obtenu sur une structure "SYMPATI2 améliorée" pour des algorithmes réguliers telle la transformée de Hough, il n'en pas de même pour des algorithmes plus délicats, telle la transformée en psi-curves, où le parallélisme total est difficile à atteindre, taux de parallélisme tributaire de la nature des images. Il est important de noter que cette limite dépend de l'algorithme, très séquentiel, et non de la structure SIMD du calculateur.

Dégageons maintenant les conclusions de l'étude présentée en annexe A1. Au cours de ce travail, il est clairement apparu que le système d'interconnexion de SYMPATI2, très efficace pour les opérations iconiques régulières, ne convient plus pour ces algorithmes de traitement moyen niveau, où les points sont transformés en listes, approximés par des polynômes où associés à des régions. En ce qui concerne le coeur des unités de traitement de SYMPATI2, à savoir leur UAL, elle semble ne pas toujours convenir également. A ce sujet, il est important de noter que les algorithmiciens programment certains algorithmes en représentation flottante, format disponible sur leur station de travail, en ne se souciant pas de savoir si une précision entière suffirait. Enfin la parallélisation de certains algorithmes semble plus efficace sur des structures non SIMD, ou sur des structures SIMD possédant un autre type de distribution des données. Voyons ces différents points plus en détails.

Regardons tout d'abord les problèmes de précision. On les retrouve pour les algorithmes récursifs telle l'approximation polynomiale par les moindres carrés, et pour les algorithmes effectuant des calculs précis telle la division/fusion pour des critères complexes, la transformée de Hough pour la détection de contours curvilignes et l'approximation polynomiale par des courbes.

Ensuite les problèmes d'architecture peuvent se répartir en 2 classes : la distribution des données sur les processeurs et le mode de fonctionnement des processeurs. Certains peuvent considérer la distribution des données points image sur les processeurs comme un problème d'interconnexion entre processeurs. Certes ces 2 problèmes peuvent se rejoindre, mais choisir entre une distribution de l'image sur les processeurs, telle la répartition hélicoïdale adoptée pour SYMPATI2, et une distribution des processeurs sur l'image, où l'image est découpée en

imagettes, reste un choix architectural, mais il est vrai qu'un système d'interconnexion peut supporter le passage d'un type de répartition à un autre. Pour les algorithmes de codage de Freeman et de division/fusion, une répartition des processeurs sur l'image conviendrait mieux que la distribution actuelle. En ce qui concerne le mode de fonctionnement des processeurs, pour le codage de Freeman, les processeurs doivent être capables de se "passer le relais" afin d'effectuer le chaînage complet d'un contour ce qui nécessite un système de contrôle particulier. Pour l'étiquetage, les processeurs doivent être capable de prendre le contrôle de la structure, les processeurs doivent pouvoir fonctionner en processeurs associatifs, et le barreau doit être scindable en plusieurs parties de longueur non connue a priori.

Enfin, les problèmes d'interconnexion entre processeurs sont nombreux. Pour les résoudre, il est nécessaire de disposer d'un système d'intercommunication permettant aux processeurs de dialoguer entre eux de manière asynchrone, à savoir lorsque le processeur numéro k dialogue avec le processeur numéro l, le processeur numéro (k+1), par exemple, doit pouvoir dialoguer avec le processeur m et ceci pour tout k, l et m. Ainsi pour l'algorithme de détection de points caractéristiques, la répartition des points détectés sur les processeurs se fait par envois asynchrones des coordonnées de ces points sur un processeur particulier. Un mode d'échanges asynchrones point à point, ou d'un processeur à un autre, est donc nécessaire. Le même mode d'échanges est exigé pour les algorithmes à parallélisme de contour tel le codage de Freeman, le codage par psi-curves et par moindres carrés et pour l'étiquetage lors de la phase de distribution des segments sur les processeurs après leur détection. Cette même opération d'étiquetage ainsi que la transformée de Hough nécessitent un système d'interconnexion supportant les opérations de globalisation d'un résultat, une donnée stockée dans un processeur est envoyée à tous les autres processeurs (mode "broadcast" ou diffusion de donnée). Notons que la plupart des algorithmes nécessitent, en fin d'exécution, une opération de restitution des résultats, ou symboles, présentés sous forme de listes, de graphes, d'arbres vers un calculateur hôte. Ce cas de figure correspond à des échanges en mode point à point particuliers, car les échanges s'effectuent avec un élément autre qu'un des processeurs synchrones SIMD. Ce type d'échange est également indispensable pour l'algorithme de l'étiquetage à base d'une solution matérielle, où les "run lenght codes" de l'image sont envoyés vers un processeur spécifique. Pour cet algorithme, les segments sont ré-envoyés dans les processeurs, cette connexion point à point particulière doit donc être bidirectionnelle.

Pour tous les algorithmes étudiés, le mode d'adressage indexé est indispensable et très largement utilisé. Les processeurs de SYMPATI2 disposent d'un tel mode d'adressage si bien qu'aucun problème n'est à signaler à ce niveau, mais il n'est pas exclu d'apporter des

modifications aux processeurs afin qu'ils effectuent ce type d'adressage plus rapidement qu'actuellement, où un cycle d'adressage en mode indexé coûte trois cycles processeur.

Le tableau II.2.2 récapitule les différents résultats obtenus pour les algorithmes présentés en annexe A1 ainsi que pour les opérations de transformée de Fourier, descripteur de Fourier, signature polaire et "strip-tree" [BAL81].

	PROBLEMES RENCONTRES		
ALGORITHMES	Interconnexion	Précision	Architecture
Extraction de points	Oui	Non	Non
Codage de Freeman	Oui	Non	Oui
Transformée en psi-curves	Oui	Non	Non
Signature Polaire	Oui	Non	Non
Moindres carrés	Oui	Oui -	Non
Polylines	Oui	Non	Non
Transformée de Hough	Oui	Non	Non
Descripteurs de Fourier	Oui	Oui	Oui
strip Trees	Oui	Oui -	Non
Transformée de Fourier	Non	Oui	Non
Etiquetage de régions	Oui	Non	Oui
Division/fusion	Oui	Oui -	Oui

***Tableau II.2.2** : Récapitulatif des résultats de l'étude sur l'adéquation de SYMPATI2 aux opérations de traitement moyen niveau. Un (Oui -) signifie que des problèmes surviennent pour certaines configurations de l'algorithme considéré.*

II.2.3 : Conclusion.

L'étude menée pour mesurer l'adéquation de SYMPATI2 aux opérations de traitement moyen niveau nous, conduit à énoncer les résultats suivants. Pour les algorithmes étudiés,

- 20% d'entres eux exigent une précision plus importante,

- 30% d'entres eux ne sont pas adaptés au mode SIMD ou au type de répartition des points sur les processeurs,

- et 90% d'entres eux exigent un système d'intercommunication entre les processeurs plus fonctionnel.

Ces résultats, sont très encourageants car ils ne remettent en cause la structure de SYMPATI2 que faiblement. L'organisation des processeurs en ligne, le séquencement des processeurs en mode SIMD et les capacités des processeurs élémentaires conviennent aux opérations de traitement moyen niveau. Il est clair que, sans les capacités d'adressage local des processeurs de SYMPATI2, à savoir le mode d'adressage indexé et la précision des UAL, l'approche montante n'aurait pu être envisagée. Cet atout est indispensable pour le traitement moyen niveau et intervient également pour les opérations géométriques [KOM90].

Suite à cette étude, notre priorité était d'améliorer les capacités d'intercommunication de SYMPATI2 tout en ayant à l'esprit les problèmes de précision et d'architecture, notamment en ce qui concerne la distribution des images sur les processeurs. La nouvelle structure doit comporter un système d'interconnexion capable d'effectuer des échanges entre processeurs de manière asynchrone, en mode point à point, et en mode diffusion (on parlera alors d'échanges asynchrones). De plus ce système doit être ouvert vers l'extérieur afin d'accélérer les entrées/sorties du système SYMPATI2, et afin d'effectuer ces entrées/sorties sous tous types de formats.

II.3 : SYMPATIX, L'ARCHITECTURE PROPOSEE.

L'objectif de ce travail est de proposer une architecture dédiée aux traitements d'images bas et moyen niveau, afin de proposer un système de vision s'appuyant sur l'approche montante. Le paragraphe précédent fait ressortir les limites de SYMPATI2 pour les opérations de traitement moyen niveau. Ces limites ne nous semblent pas insurmontables, et argumentent notre choix de l'approche montante pour SYMPATI2. L'organisation ligne des processeurs, leur mode d'adressage indexé, la précision de leurs UALs conviennent. Il est toutefois nécessaire de proposer un système d'intercommunication adapté aux échanges sollicités lors des opérations de traitement moyen niveau. Dans ce paragraphe, un système d'interconnexion est proposé et connecté à SYMPATI2, ce réseau forme SYMPATIX le nouveau calculateur SIMD proposé.

II.3.1 : Du Besoin vers l'Idée.

Rappelons les besoins exigés au niveau interconnexion pour les opérations de traitement moyen niveau. Chaque processeur élémentaire, d'une matrice SIMD, doit être capable d'échanger de manière asynchrone des données avec les autres processeurs de la matrice SIMD. Les échanges asynchrones sont caractérisés par le fait que lorsqu'un processeur, numéro k, dialogue avec un autre processeur, numéro l, son voisin, numéro (k+1) dialogue avec un autre processeur, numéro m, ceci pour tout k, l et m. Les M processeurs de la matrice peuvent dialoguer avec le même processeur, comme un processeur peut envoyer une donnée à tous les autres processeurs, on parlera alors de "broadcast de données" ou diffusion de données. Nous avons vu également, que les processeurs doivent être capables de communiquer rapidement vers l'extérieur, donc vers un composant désynchronisé par rapport aux processeurs SIMD.

Ces échanges peuvent être supportés de deux manières différentes, soit par connexion physique des unités dialogantes, soit par connexion virtuelle. Les connexions physiques entre éléments s'effectuent par un réseau d'interconnexion. Une liaison électrique est alors réalisée entre les éléments communicants si bien que les échanges s'effectuent très rapidement ce qui ne bloque pas les processeurs. Des différents réseaux d'interconnexion présentés dans le paragraphe I.2.2 de cette thèse, seul le crossbar conviendrait le mieux à notre besoin. Mais un tel réseau est coûteux en matériel, donc "énergivore", coût proportionnel au carré du nombre de processeurs, et exige beaucoup de signaux de commande. Enfin un tel réseau peut très difficilement être partitionné en modules associés aux processeurs, en vue de leur intégration avec ceux-ci.

La seconde méthode, basée sur des connexions virtuelles entre processeurs, s'appuie sur un réseau d'intercommunication (c.f. paragraphe I.2.2). Dans ce cas, la connexion virtuelle s'effectue par passage de messages entre les unités dialogantes. Le problème de ces liens virtuels réside dans la vitesse à laquelle s'effectue les échanges, vitesse faible par rapport aux réseaux à connexion physique. Disposons nous de temps pour effectuer les communications? Oui et Non! Oui si ces échanges s'effectuent en parallèle du traitement, non si ce n'est pas le cas. En effet, pour les opérations de traitement d'images moyen niveau, seules les caractéristiques, ou symboles, des images transitent lors de ces échanges, et le nombre de ces caractéristiques est très faible par rapport au nombre de points de l'image. En d'autres termes, un symbole est construit à partir d'un nombre de points image important. Si bien que lorsque les processeurs élaborent les symboles (n+1), les symboles (n) calculés précédemment peuvent être échangés entre les processeurs à la condition d'effectuer en parallèle les

opérations de traitement et d'échange. Toutefois, il ne faut pas que le temps nécessaire à l'exécution d'une tâche de communication soit plus long que celui nécessaire à l'exécution d'une tâche de traitement, ce qui bloquerait les processeurs (figure II.3.1).

Il est ainsi nécessaire de réaliser un système d'intercommunication ne bloquant pas les processeurs, donc autonome par rapport à ceux-ci. Le paragraphe suivant présente le réseau d'intercommunication retenu.

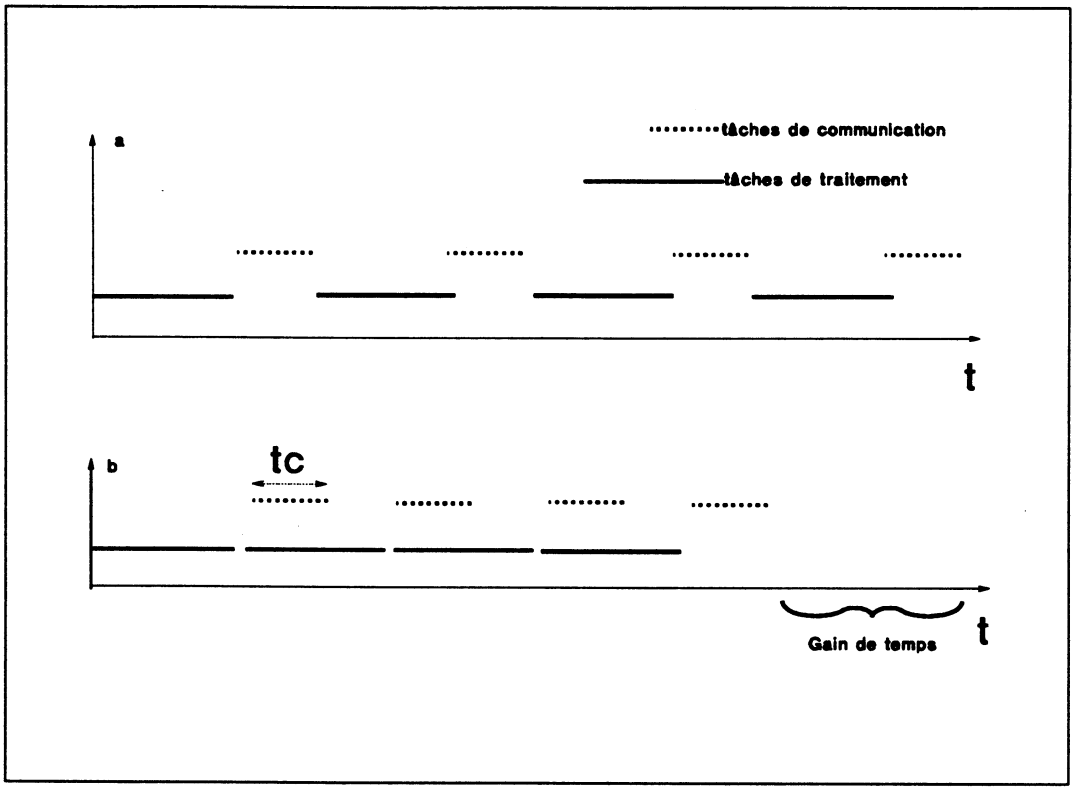

***Figure II.3.1** : Principe de l'idée retenue. Actuellement (a) les processeurs effectuent une tâche de traitement puis une tâche de communication indispensable pour distribuer les résultats. L'idée est d'effectuer cette tâche de communication en parallèle avec la tâche de traitement suivante (b). Le nombre de tâches est très conséquent lors des opérations de traitement d'images (quelques milliers).*

II.3.2 : Le Réseau Intelligent Ouvert.

Les types d'échanges que doit remplir le système d'intercommunication à développer, sont les suivantes :

- communication point à point entre processeurs;

- communication entre k processeurs vers un seul processeur (extension du mode point à point);

- diffusion de données, un (ou des) processeur(s) envoie(nt) une donnée vers tous les autres;

- communication avec l'extérieur en entrée et en sortie.

De plus, dans le paragraphe précédent, il est dit que le système d'interconnexion doit travailler en autonome avec la matrice de processeurs SIMD. Le réseau doit pouvoir se configurer, physiquement ou virtuellement, par les données, donc doit être à contrôle réparti. Ainsi nous proposons, pour remplir cette fonction, un réseau d'intercommunication à passage de messages basé sur le principe des réseaux locaux à insertion de registres [OUS87]. Ce type de réseau a déjà été mis en oeuvre sur des systèmes multiprocesseur asynchrones, tels des ordinateurs reliés par réseau ou le système multiprocesseur à base de Z80, le ZMOB.

Le réseau retenu peut être vu comme un ensemble de registres reliés entre eux, les uns aux autres et rebouclé sur lui même pour former une boucle à décalages. Ces registres supportent les échanges par décalages successifs dans la boucle. Les processeurs viennent écrire leur message dans un registre du réseau, ensuite ils travaillent sur d'autres données alors que le réseau, appelé pour cette raison réseau intelligent, effectue en autonome la distribution des messages grâce à la boucle à décalages.

Les messages sont constitués de trois champs :

- un champ données,

- un champ adresses,

- un champ contrôle.

Le champ adresse contient l'adresse du processeur destinataire à qui est envoyée la donnée contenue dans le champ données. Ce champ adresses peut également contenir l'adresse de stockage dans le banc mémoire du processeur destinataire. Le champ contrôle quant à lui contient les informations nécessaires à la gestion du réseau.

En interne le réseau est composé de cellules chacune connectée à un processeur. Les cellules reçoivent les messages des processeurs puis les font circuler rapidement entre elles

dans un sens défini. Lors de cette rotation de l'ensemble des messages, chaque cellule identifie le message arrivant à sa hauteur, le conserve s'il lui est adressé, sinon le transmet à la cellule voisine. Dès que le réseau est vide, une autre écriture dans le réseau est permise. La figure II.3.2 présente le principe de ce réseau. Notons également que le réseau s'ouvre sur l'extérieur par simple adjonction d'une cellule supplémentaire, d'où l'appellation de **Réseau Intelligent Ouvert, "RIO"**.

Face aux problèmes à résoudre, le RIO est très efficace. Présentons cette adéquation en détails. Pour la communication point à point (ou de k processeurs vers 1 seul), il suffit aux processeurs d'émettre un message sur le réseau contenant le numéro du processeur destinataire, un même destinataire pouvant être sollicité à plusieurs reprises au cours d'un même échange. La distribution s'effectuant en séquentiel, il ne peut y avoir de conflits en réception. Les processeurs construisent et émettent les messages sans se soucier de leurs voisins, les échanges sont purement asynchrones.

En ce qui concerne la diffusion d'une donnée, deux solutions sont envisageables, l'une lente par décalage, l'autre très rapide par diffusion électrique. Pour la première, les processeurs reçoivent une instruction particulière de diffusion, envoient un message ne contenant pas de numéro de processeurs destinataires, car tous les processeurs reçoivent le message, puis le message parcourt toute la boucle et est récupéré par tous les processeurs. La seconde solution demande du matériel supplémentaire, car toutes les cellules doivent pouvoir se shunter. Les processeurs effectuent tous l'instruction de diffusion, ensuite toutes les cellules sont court-circuitées pour former un bus électrique sur lequel les cellules, les unes après les autres envoient leur donnée qui alors est diffusée électriquement puis récupérée par les processeurs. Cette solution ne présente pas d'intérêts par rapport à la précédente si tous les processeurs ont une donnée à diffuser, mais est très rapide si un seul processeur désire effectuer cette opération.

Enfin le réseau, donc les processeurs, peut s'ouvrir vers le monde extérieur, il suffit pour cela d'ajouter une cellule assurant le dialogue des processeurs avec l'extérieur.

Les avantages d'un tel réseau sont les suivants : le coût matériel de la structure croit linéairement avec le nombre de processeurs, le réseau est intégrable avec le processeur et il travaille en autonome. Par contre, ce réseau effectue les échanges lentement, mais en parallèle avec les traitements, si bien qu'il ne peut être utilisé pour toutes les applications, mais convient au traitement d'images de moyen niveau, où le temps de communication est inférieur au temps de traitement. La vitesse des échanges dépend de la technologie utilisée. Si les

échanges sont supportés par des liaisons optiques, la vitesse deviendra bien plus rapide et le domaine d'application plus étendu. Dans les chapitres suivants, les vitesses des échanges sont présentées. Il peut être reproché à ce réseau, comme à tous les réseaux à passage de messages, de ne travailler qu'en émission de données, jamais en demande de données de la part d'un processeur. Cette configuration en lecture aléatoire à distance, ne peut être obtenue sur une structure synchrone SIMD, où tous les processeurs effectuent la même instruction. Et enfin, la contre-partie de la croissance du coût en O(M) se ressent au niveau des performances du réseau, car celles-ci se dégradent en fonction du nombre de processeurs. Par exemple dans une structure à M processeurs, s'il faut, au pire cas, un temps T pour acheminer un message d'un processeur à un autre, ce temps sera multiplié par 2 si le nombre de processeurs double. Pour garder le même temps, deux réseaux circulant en sens opposé peuvent être juxtaposés afin de retrouver ce temps T ceci au prix d'un matériel double.

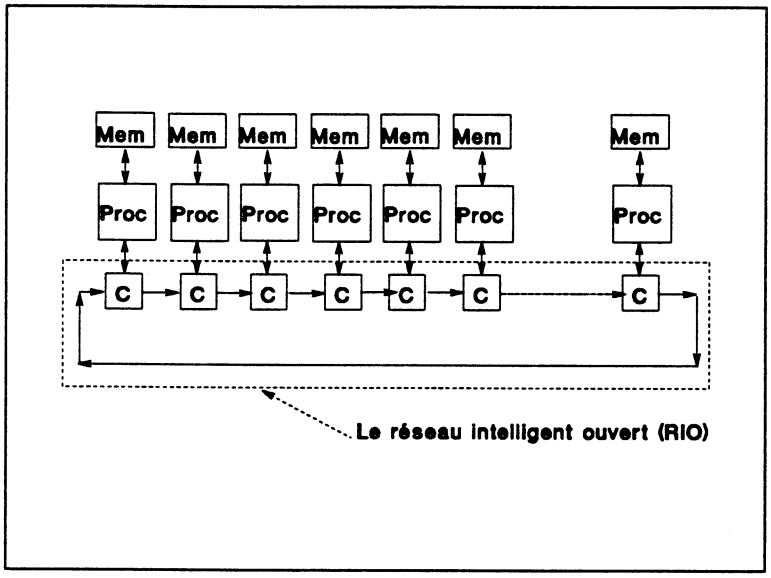

***Figure II.3.2** : le Réseau Intelligent Ouvert. Les Processeurs (Proc), de tous types accèdent au RIO par leurs bus de données et d'adresses. Le RIO est constitué de cellules (C) qui propagent les messages par décalages successifs. Pour accélérer les échanges, 2 réseaux peuvent être mis en oeuvres, les messages étant aiguillés vers le réseau proposant le plus court chemin pour l'envoi du message.*

II.3.3 : SYMPATIX.

Connecté à SYMPATI2, le Réseau Intelligent Ouvert forme SYMPATIX, la structure proposée pour supporter les opérations de traitement bas et moyen niveau [COLa92, COLb92, COLb91]. L'originalité d'une telle proposition réside dans l'interconnexion d'une structure synchrone SIMD à un réseau d'intercommunication effectuant des échanges asynchrones.

L'interfaçage du Réseau Intelligent Ouvert avec une structure synchrone SIMD n'est pas du tout implicite, les qualités nécessaires à cet interfaçage sont présentées ici. Tout d'abord les processeurs doivent être capables de construire en parallèle le message destiné à un processeur. La construction du champ de données s'effectue facilement par n'importe quel type de processeur. Le champ de contrôle quant à lui est généré automatiquement par la cellule. Par contre générer le champ d'adresses pose des problèmes sur les structures ne possédant qu'un mode d'adressage synchrone. Dans ce cas, les processeurs construisent ce champ à partir de données, par écritures successives dans la mémoire. Cette solution est coûteuse en temps d'exécution. Par contre, les processeurs possédant un mode d'adressage indexé, utilisent directement cet index pour générer ce champ adresses. Ensuite, pour nos algorithmes de traitement d'images de moyen niveau, lorsque qu'une instruction "envoi réseau" est effectuée, tous les processeurs ne possèdent pas forcément une donnée à émettre. En effet, ces échanges sont déterminés de façon dynamique, leur contrôle doit donc être distribué sur les processeurs. Comment interdire à un processeur SIMD de ne pas effectuer cette opération? Il faut le masquer, ou plutôt il doit s'auto-interdire cette écriture. Ainsi, l'exécution de cette opération d'envoi de messages doit être contrôlée par le processeur lui même grâce à des indicateurs propres à chaque processeur. Sans cette possibilité, l'interfaçage d'un processeur SIMD avec le RIO est compromise.

Les processeurs élémentaires de SYMPATI2 possèdent un mode d'adressage indexé, appelé le mode tabulaire, et des indicateurs leur permettant de s'inhiber. Grâce à ces deux caractéristiques, les deux modules, SYMPATI2 et RIO, s'interfacent bien, et associés ensembles forment SYMPATIX (figure II.3.3).

Ainsi SYMPATIX, comparé à SYMPATI2 présente les avantages suivants :

- les échanges asynchrones (non réguliers) entre processeurs sont supportés avec efficacité;

- les processeurs peuvent être interconnectés entre eux directement par liaison électrique afin de réaliser des globalisations de résultats;

- la connexion avec l'extérieur s'effectue via le RIO, ce qui permet d'accroître les capacités du système au niveau des entrées/sorties, qui peuvent être maintenent de tous formats (image, listes, arbres...);

• l'intégration d'un module de calcul particulier est possible grâce au réseau intelligent ouvert.

Les échanges s'effectuent par passage de messages entre un processeur émetteur et un (des) processeur(s) récepteur(s). La réception se fait soit par stockage dans un ou des registres, soit par écriture directe dans la mémoire associée au processeur récepteur. Tous les détails de l'architecture sont présentés dans le chapitre suivant consacré à la simulation. Les simulations réalisées, à partir de la modélisation comportementale du calculateur SYMPATIX, nous ont permis de définir plus en détails les caractéristiques du RIO, à savoir la largeur des champs de données, adresses et contrôle, le nombre de registres de réception et bien d'autres fonctions, mises en avant grâce à cette approche.

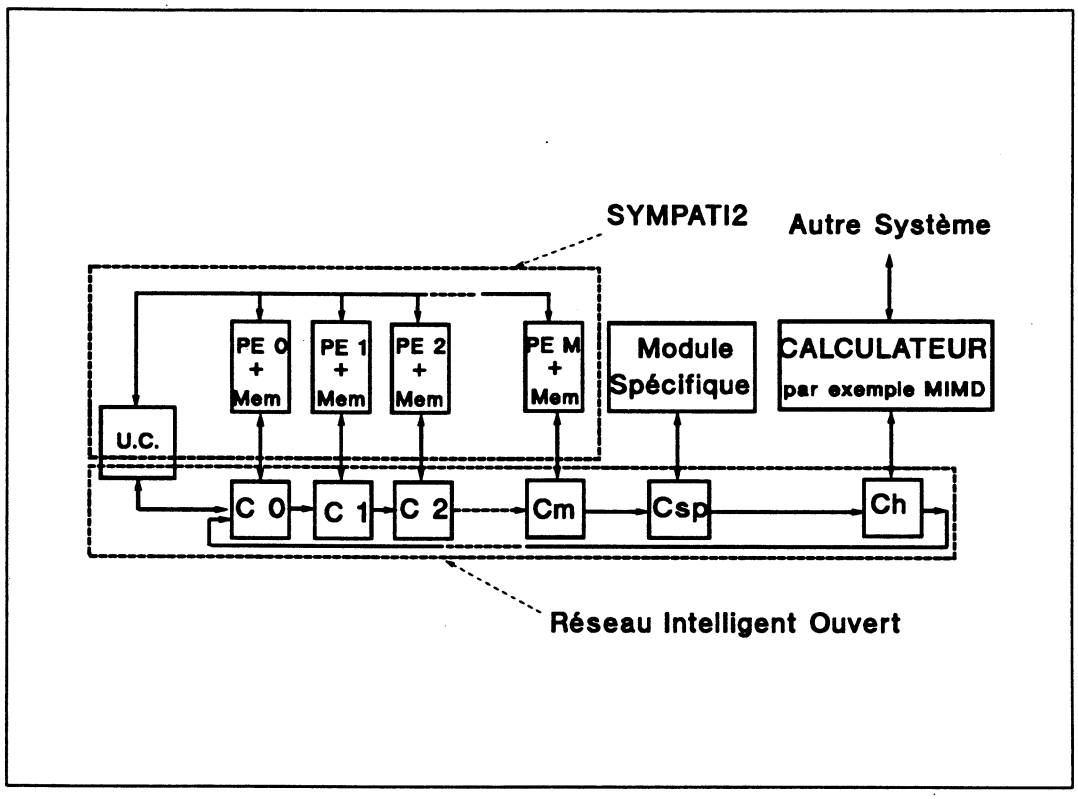

***Figure II.3.3** : Le système SYMPATIX. Le système est ouvert à un (des) module(s) spécifique(s), tel un processeur flottant, de même qu'à un processeur de haut niveau. Des cellules particulières, Csp et Ch, sont alors nécessaires pour interfacer ces modules au RIO.*

II.3.4 : Conclusion.

Ainsi dans une approche montante, SYMPATIX constitue le calculateur parallèle dédié au bas et au moyen niveau du traitement d'images, le haut niveau, quant à lui, doit être

exécuté sur un processeur parallèle de type "general purpose", facilement couplable à notre calculateur au travers du réseau intelligent ouvert (figure II.3.4). Les atouts offerts par les capacités du processeur à s'ouvrir vers l'extérieur, sont de trois types :

- échange de données avec le processeur dédié au haut niveau;

- utilisation, par SYMPATI2, d'un (ou des) coprocesseur(s) spécialisé(s), inséré(s) dans la boucle à décalage (coprocesseur flottant par exemple pour effectuer des opérations flottantes afin d'affranchir SYMPATIX des problèmes de précision);

- organisation des point dans la mémoire, sous différents formats de données, autres que l'organisation hélicoïdale, ce qui permet de résoudre certains problèmes d'architecture vus en II.2.2.

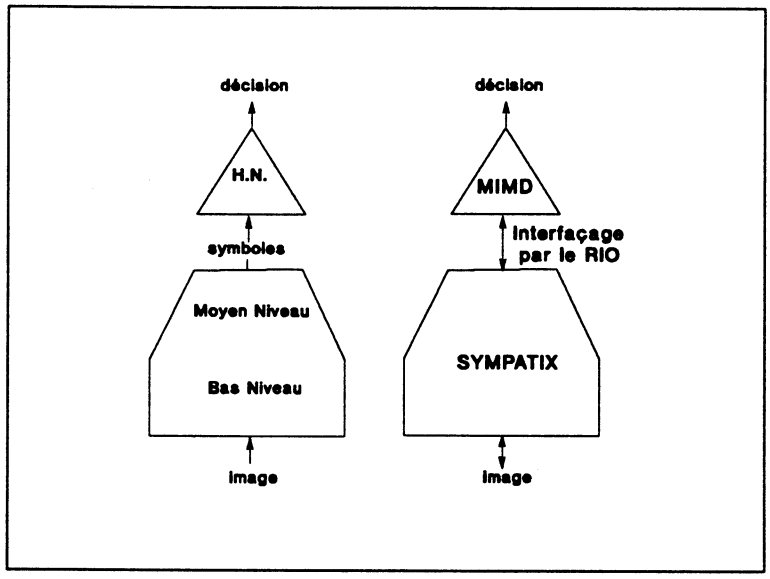

***Figure II.3.4** : SYMPATIX et les opérations de traitement d'images. Le bas et le moyen niveau sont traités sur SYMPATIX, le RIO effectue, entre autres, les transferts entre le calculateur de haut niveau et SYMPATIX.*

II.4 : CONCLUSION.

L'objectif du travail présenté dans cette thèse est de proposer un calculateur dédié au bas et au moyen niveau du traitement d'images, afin de l'utiliser dans un système de vision s'appuyant sur l'approche montante. Ce chapitre dresse les carences de la structure de SYMPATI2 pour le domaine d'application visé, et propose une nouvelle architecture basée sur ce calculateur.

Les limites de SYMPATI2 sont misent en évidence par une étude algorithmique des opérations de traitement moyen niveau. De cette étude, il ressort que les capacités d'intercommunication entre les processeurs de SYMPATI2 doivent être augmentées. Ainsi le Réseau Intelligent Ouvert est proposé et connecté à SYMPATI2, l'ensemble forme SYMPATIX.

Lors des opérations de traitement d'images, où les algorithmes sont découpés en tâches de communication et en tâches de traitement, le temps de calcul de ces opérations est égal au temps d'exécution des tâches de traitement seulement, les tâches de communication s'effectuant en parallèle des traitements grâce au réseau intelligent ouvert. Les échanges supportés par ce réseau sont de type asynchrones, le temps d'échange maximal est proportionnel au nombre de processeurs (O(M)). SYMPATIX est ainsi doté d'un mode d'écriture à accès aux processeurs voisins aléatoire RAW ("Random Access Write"), caractéristique indispensable aux processeurs effectuant des communications dépendantes des données [CYP89].

Les chapitres suivants présentent plus en détails la structure du réseau intelligent, ses performances, ainsi que celles obtenues pour SYMPATIX à partir d'une modélisation comportementale du système complet.

Chapitre III

Méthodologie de Conception et de Simulation.

A ce stade de l'étude, les fonctionnalités de SYMPATIX étaient connues mais non la structure exacte du réseau intelligent ouvert. Il était donc nécessaire de figer cette structure et de mesurer les performances du nouveau calculateur afin de le valider.

Dans cet objectif, plusieurs méthodes ont été envisagées. La première est de construire directement une maquette afin de démontrer réellement les performances de la structure. Cette méthode, fréquemment utilisée quoique démodée, présente l'avantage de proposer une mesure concrète des performances de la structure évaluée, au prix d'un temps de développement important. De plus la structure évaluée est figée si bien que les performances obtenues ne sont pas paramétrables (par exemple bus de largeur variable...).

La seconde solution est de créer un simulateur logiciel de la structure. Cette solution, certes rapide, est complètement déconnectée de la réalisation matérielle, si bien que les performances obtenues ne sont pas garanties après la réalisation du système électronique.

Enfin, une troisième solution est de construire un modèle comportemental de la structure. Ce modèle est décrit grâce à un langage de description de matériel (Hardware Description Language HDL), puis il est affiné par un jeu de simulations pour ensuite être transformé en description électrique nécessaire à la réalisation du circuit. Par rapport à la seconde solution, le modèle obtenu est réutilisé par les concepteurs de circuits et de systèmes, et devient alors la référence du circuit. Cette dernière approche a été retenue pour valider le système et nous sera bien utile pour réaliser SYMPATIX.

Dans ce chapitre, nous présentons la nécessité de la simulation dite de "haut niveau", ensuite est détaillée notre plate forme d'évaluation d'architecture, ainsi que le modèle comportemental de SYMPATIX.

III.1 : LA SIMULATION EST-ELLE UNE NECESSITE?

La conception électronique met en oeuvre des circuits intégrés lesquels incorporent de plus en plus de fonctions logiques, donc de plus en plus de transistors. Cet accroissement de capacité d'intégration est dû à l'utilisation de technologies de moins en moins "énergivores" d'une part, et à la définition de la lithographie devenant de plus en plus fine d'autre part (les grilles ont vu leur largeur évoluer de 25 µm en 1965, 10 µm en 1970, 1 µm en 1987, à 0.4 µm aujourd'hui). Nous sommes ainsi passés d'une capacité de 1 transistor bipolaire par boîtier en 1964 à une capacité de 100 à 1000 transistors par boîtier dans les années 70 pour ce que l'on

appelle les circuits intégrés (CI), puis à 10 000 transistors dans les années 80 avec les circuits LSI (Large Scale Integration) pour arriver aux circuits VLSI (Very LSI) intégrant jusqu'à un million, voire plus, de transistors à effet de champ (MOS) [AGE92].

La conception de circuits spécifiques (ou ASIC : "Application Specific Integrated Circuit") ne peut s'effectuer sans outils informatiques de Conception Assistée par Ordinateur (CAO). Au cours de la conception du circuit VLSI, la spécification du composant est transformée en dessin de masques utilisés lors de la fabrication [SAV88] (figure III.1.2). Les outils de CAO dédiés à la conception de circuits VLSI s'appuient sur des synthétiseurs logiques et électriques, sur des simulateurs numériques et analogiques [LAT90] ainsi que des générateurs de séquences de test et des simulateurs de fautes. Par conséquent, pour cette étape, la simulation est obligatoire, elle permet de vérifier la cohérence entre les différents niveaux de description obtenus, permet donc de fabriquer des circuits conformes à leurs spécifications.

Si la simulation est indispensable pour concevoir des ASICs, qu'en est-il pour la conception de systèmes ou de cartes électroniques? Un système est composé de plusieurs circuits VLSI (voire LSI ou CI), qui sont regroupés pour remplir des fonctions particulières. Les concepteurs assemblent ainsi différents circuits du marché entre eux afin d'élaborer le système final. Cet assemblage s'effectue pour 85% des conceptions sans validation au préalable par des simulations (c.a.d. sans outils de CAO), et ce n'est qu'une fois sous tension que les performances du système sont véritablement déterminées. Pour cette approche relativement empirique, en moyenne 3 à 5 prototypages de circuits imprimés sont à effectuer avant d'obtenir la carte conforme aux spécifications. Avec une telle approche, le temps de développement n'est pas maîtrisé, le coût du prototype peut devenir important et le système peut présenter des problèmes de fiabilité non maîtrisables. Par contre, la simulation du système complet (carte complète) évite de développer autant de circuits imprimés. Alors pourquoi seulement 15% des cartes sont simulées? Parce que les simulateurs coûtent chers d'une part, et parce que d'autre part, du fait que les cartes fabriquées sans simulation préalable sont encore "facilement" débogables (elles offrent de nombreux points de test accessibles par l'oscilloscope), le concepteur arrive toujours à déceler les erreurs si bien que la nécessité de la simulation n'apparaît pas clairement. Cependant, à cause, ou plutôt grâce à l'évolution des techniques d'intégration, le débogage des cartes non simulées, basé sur l'utilisation de points de test, devient de plus en plus complexe. En effet les cartes CMS (composants montés en surface) ou MCM (multi-chip module) regroupent des circuits de plus en plus complexes, et présentent donc peu de points de contrôle intrinsèques, ce qui interdit cette approche de

conception empirique. Il est ainsi prévu qu'en l'an 2000, 80% des circuits soient conçus avec des outils de CAO.

La simulation de système permet notamment de réduire le temps de développement (grâce à la conception concurrente), de s'affranchir au maximum des erreurs de conception et de disposer d'un modèle du système utilisable pour des études de testabilité et réutilisable lors de la conception d'évolutions du système (changement du processeur sur une carte). Par exemple, pour une carte comportant 8 couches au format double Europe (5000 traversées et 5 diamètres de perçage), le gain en temps réalisé pour une itération de prototypage est de 9 semaines (temps de conception + temps de fabrication + temps de montage + temps de débogage) soit une économie d'environ 350 KF et un gain de temps d'environ 40% [CAR92] (figure III.1.1). Le produit peut donc être proposé plus rapidement sur le marché. Pour les sociétés développant leurs ASICs, cette approche est fortement préférable. En effet, actuellement 1 ASIC sur 2 fabriqués avec succès, ne fonctionnent pas dans son environnement complet par manque de simulation du composant avec les autres circuits du système.

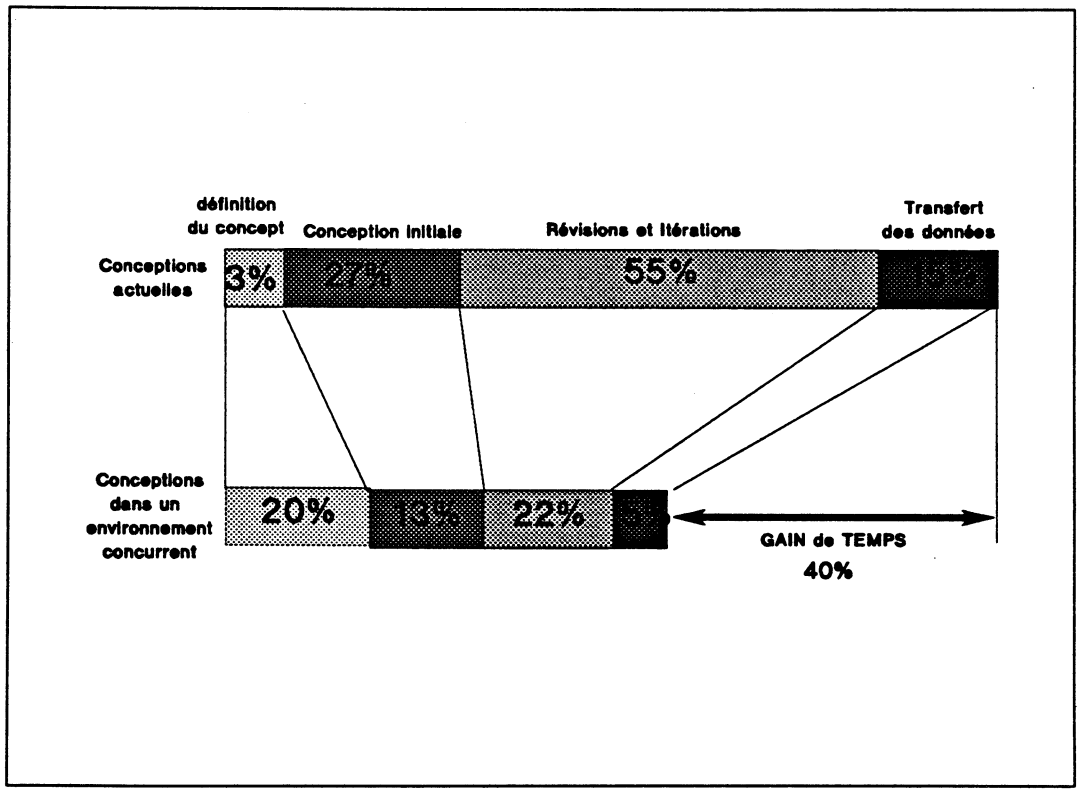

Figure III.1.1 : Profits engendrés grâce à la conception concurrente d'un circuit électronique - sources US Air Force study dans [CAR92].

Mais, qu'en est-il des temps et des coûts de simulation? Il faut différencier les simulations de systèmes composés de composants du marché et les simulations de systèmes à base d'ASICs. Pour la première catégorie, des modèles de ces composants du marché existent si bien que la conception du modèle structurel du système est aisée et les temps de simulation sont acceptables grâce à la qualité des modèles commercialisés par des sociétés spécialisées. Par exemple, pour un système composé d'un 80386, de mémoires et de PLDs, le temps de simulation d'un cycle du système est de 20 secondes sur une station de travail de 4 MIPS. Par contre, pour les systèmes à base d'ASICs, soit le concepteur a écri le modèle de haut niveau de son circuit auquel cas la simulation est possible et présente les mêmes performances que celles présentées ci-dessus, soit le concepteur ne peut disposer d'un tel modèle, auquel cas la simulation ne pourrait se faire qu'avec la description fine du circuit (niveau portes) ce qui conduirait à des temps de simulation prohibés, il faudrait compter plusieurs dizaines de minutes par cycle machine. Il est donc nécessaire de disposer d'un modèle comportemental du circuit à fabriquer, modèle qui peut être écrit avant la conception du circuit ou après. S'il est conçu après, il peut apparaître, trop tardivement, que le circuit s'intègre mal, au niveau fonctionnel, dans le système. Cette solution, appelée conception montante, est donc à éviter. Notons également que le modèle écrit doit représenter rigoureusement le fonctionnement du circuit, et doit être vérifié par une simulation exhaustive, si bien que le temps de développement du modèle est long. Il est ainsi préférable de concevoir ce modèle avant la fabrication du circuit, il sert alors de référence et permet d'évaluer le circuit dans son environnement de fonctionnement, ces évaluations pouvant conduire à modifier ou compléter les fonctionnalités du circuit, donc le modèle. On parle alors de conception descendante.

Ainsi la simulation de système devient une nécessité de par les techniques d'assemblage de circuit VLSI qui deviennent de plus en plus fines, et de par les contraintes commerciales qui demandent aux entreprises de placer sur le marché leurs produits le plus tôt possible ("time to market"). Les avantages de la simulation de systèmes sont les suivants :

- Vérification de la conception;
- Réutilisation du modèle pour la conception de variantes du produit;
- Diminution du temps et du coût de développement;
- Conception concurrente matérielle et logicielle;

• Meilleur niveau de qualité.

Cette approche de simulation système demande aux concepteurs de circuits spécifiques d'élaborer leurs ASICs par une méthode de conception descendante, et cette approche n'est possible que si la société investit dans un système de CAO, système dont le coût est approximativement de 800 KF.

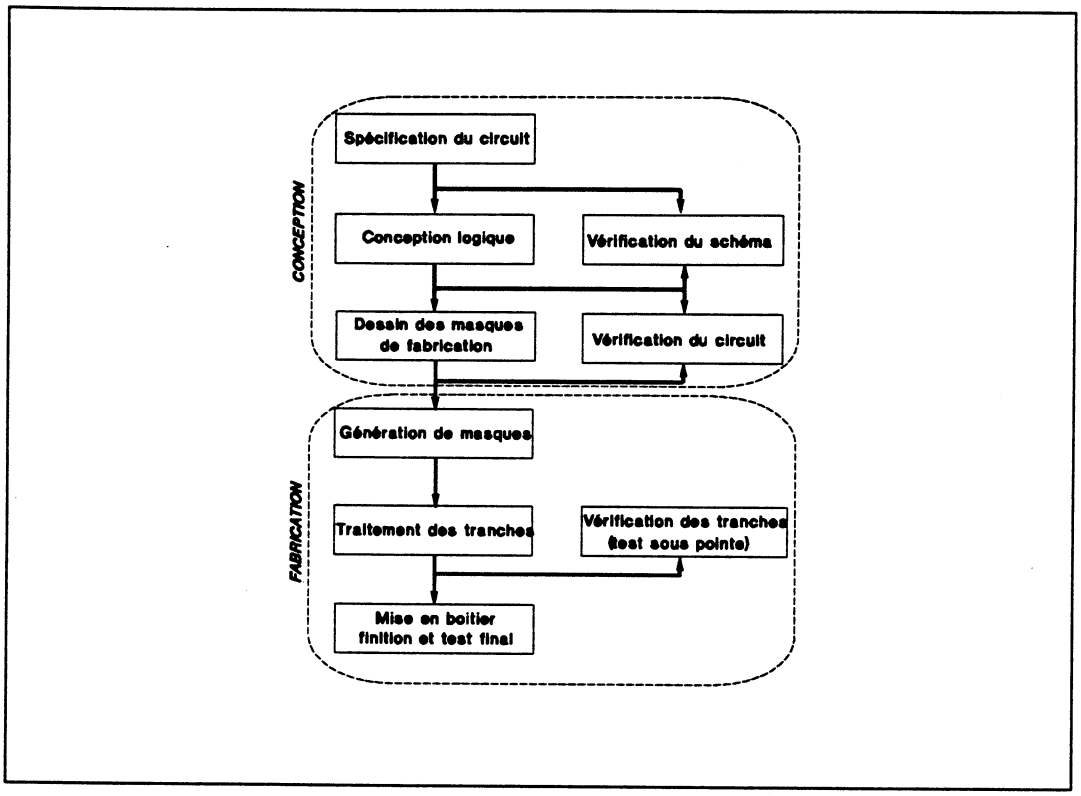

***Figure III.1.2** : Les différentes phases de la conception et de la réalisation d'un circuit électronique.*

III.2 : ENVIRONNEMENT D'EVALUATION.

Il est maintenant clair que la simulation de systèmes est une nécessité, celle-ci devant s'effectuer dans l'optique de la conception descendante. Cette méthodologie demande de concevoir des modèles comportementaux, conceptions s'effectuant grâce à des langages de description de matériel (HDL : "Hardware Description Language"). Ce paragraphe présente la conception descendante, le HDL retenu ainsi que la plate forme de simulation utilisée pour définir et évaluer SYMPATIX, ou d'autres architectures.

III.2.1 : La conception descendante.

Concevoir un système électronique par une approche descendante consiste à obtenir, à partir du modèle du système à réaliser, le schéma électrique des différents circuits à fabriquer. Le passage de la description système à la description électrique ne s'effectue pas directement, plusieurs niveaux intermédiaires existent.

Au plus haut niveau, on trouve la *description système*, niveau dans lequel est modélisé un ensemble de processeurs coopérants. C'est à ce niveau que peuvent être prises en compte les parties non électroniques du système. Puis, le niveau plus fin de description suivant est le *niveau algorithme*, niveau dans lequel les différents blocs de contrôle du système sont décrits grâce à des processus concurrents. Ensuite vient le *niveau transfert de registres* où chacun des blocs de contrôle est représenté par des modules fonctionnels interconnectés entres-eux, modules dont l'exécution est conditionnée par une fonction booléenne connue. Ensuite vient le *niveau portes* où chaque module fonctionnel est décrit par un ensemble de blocs représentés par des équations booléennes. Ensuite, le *niveau "switch"* (commutateur) permet d'obtenir le niveau le plus fin de la description logique, en effet les blocs booléens du niveau porte sont décrits par un ensemble de commutateurs représentant idéalement les transistors nécessaires à la réalisation du circuit. Enfin, vient le *niveau électrique* où chacun des commutateurs du niveau précédent est représenté par le modèle du transistor associé. Les valeurs traitées ne sont plus discrètes, mais continues (tableau III.2.1).

Lors de l'élaboration d'un système électronique par une technique de conception descendant, le travail débute par l'écriture et la mise au point du modèle comportemental du système. C'est à ce niveau que toutes les fonctionnalités sont définies, modélisées et affinées par une méthode de conception incrémentale [HOL91]. Ensuite, les autres niveaux sont atteints par des opérations de découpage en blocs ("partitionning"), pour passer du niveau système au niveau algorithme et au niveau registre de transfert, ou par des opérations de synthèse logique pour arriver à la description électrique du circuit. La première description, niveau système, reste tout le long de la conception la référence fonctionnelle à laquelle chaque autre description doit rigoureusement correspondre (figure III.2.1). Cette correspondance est garantie par les outils automatiques, mais ne l'est pas lors de passage manuel d'un niveau à un autre, typiquement pour le découpage du niveau algorithme afin d'obtenir un modèle au niveau transfert de registres. Seule une vérification des différents modèles par preuve formelle ou simulation exhaustive préserve des erreurs [BOR89].

Les intérêts d'une telle approche sont nombreux, nous ne présentons ici que les trois plus importants. Premièrement, les circuits réalisés sont simulés dans leur environnement de

fonctionnement final, ce qui évite de concevoir des ASICs mal adaptés à leur milieu et permet d'effectuer des analyses du système, analyses en performances, en testabilité, en contraintes de fabrication et de maintenance. Puis, les vecteurs de test fonctionels, définis au niveau du modèle du système, restent les mêmes pour tous les autres niveaux de description, si bien qu'aucune erreur ne peut intervenir sur ces vecteurs. Enfin, une fois le modèle du système défini, il est possible d'effectuer une conception concurrente, au niveau matériel et logiciel, du système ce qui permet d'introduire plus rapidement le produit sur le marché. En effet, si le système contient plusieurs ASICs à concevoir par exemple, il est possible de confier la conception de ces circuits à différentes équipes, la référence contenant tous ces circuits (modèle du système) n'évoluant plus. De même le comportement du circuit étant connu, il est possible aux programmeurs d'écrire des logiciels destinés au système final pouvant être testés sur le modèle comportemental de ce système. C'est la description du système qui offre cet environnement de conception concurrent.

NIVEAU	CONCEPT de MODELISATION	MODELE TEMPOREL	VALEURS OBSERVABLES	GEOMETRIE
SYSTEME	Composants coopérants	Symbolique Causalité	Symboles	Composants interconnectés
ALGORITHME	Algorithmes concurrents	front horloge état logique	vecteurs de bits	Structure de causalité
TRANSFERT DE R REGISTRES	Composants à commandes gardées	front horloge	vecteurs de bits	structure de registres de transfert
PORTES	Equations booléennes	Retard unitaire	bit (0 ou 1)	"netlist"
COMMUTATEUR	Equations discrètes	Domaine continu	Domaine continu avec interprétation logique	"netlist" de commutateurs
ELECTRIQUE	Equations différentielles	Domaine continu	Domaine continu	"Layout" métrique

Tableau III.2.1 : Les différents niveaux de la conception électronique [RAM91].

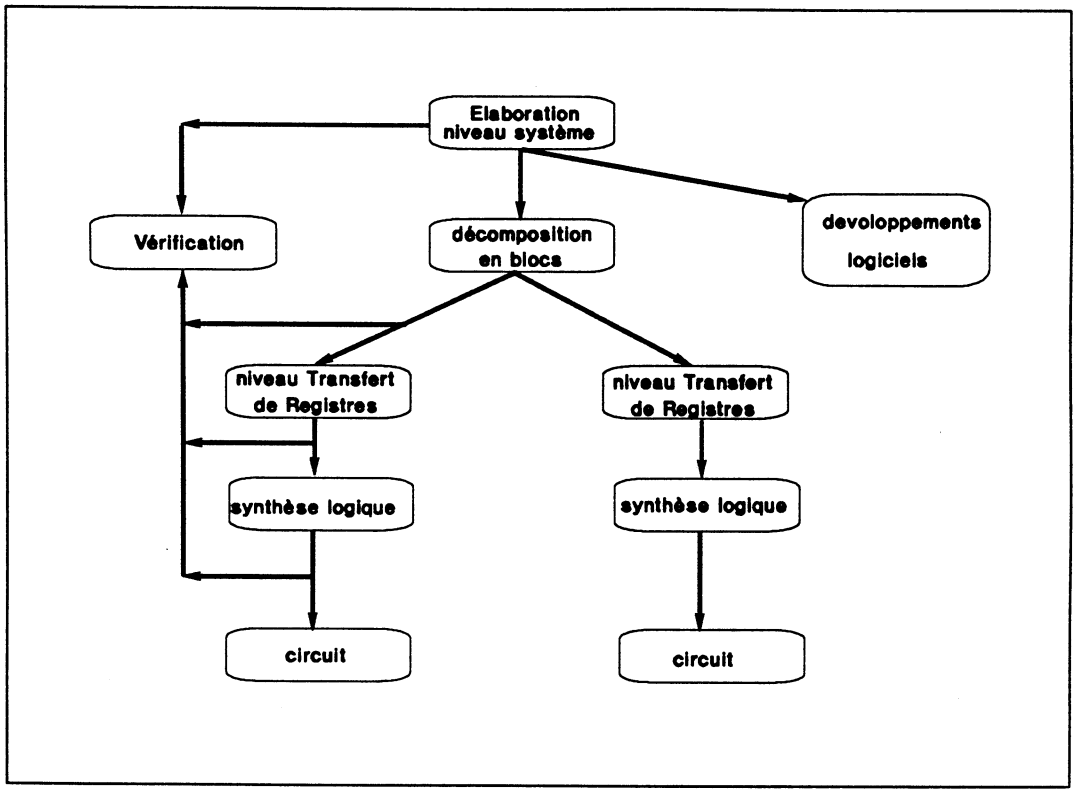

Figure III.2.1 : La conception descendante. Une fois le modèle du système établi, le développement des éléments du système s'effectuent de manière concurrente.

III.2.2 : Le langage de description de matériel retenu.

Le système peut être modélisé de différentes manières, soit par graphe d'états (StateChart, SDL), soit par programmation logique (Prolog), par programmation fonctionnelle (LISP), par programmation orientée objet (modula, ADA) ou par programmation en un langage de description de hardware (DACAPOIII, VHDL). L'intérêt de ces derniers est qu'ils possèdent des instructions propres à la définition des signaux électriques ceci permettant d'effectuer des descriptions à différents niveaux (aussi bien au niveau système qu'au niveau RTL par exemple) et permet donc la conception descendante. C'est la raison qui nous a poussé a retenir un de ces langages comme outil de spécification.

Il existe de nombreux langages de description de hardware, couvrant chacun un certain nombre de niveaux de conception [DAS87]. Il est préférable de se doter d'un langage capable d'effectuer la modélisation des systèmes électroniques et pouvant être utilisé jusqu'au niveau commutateurs (figure III.2.2). Entre autres, DACAPOIII, VHDL et VERILOG répondent à cette contrainte, alors lequel choisir?

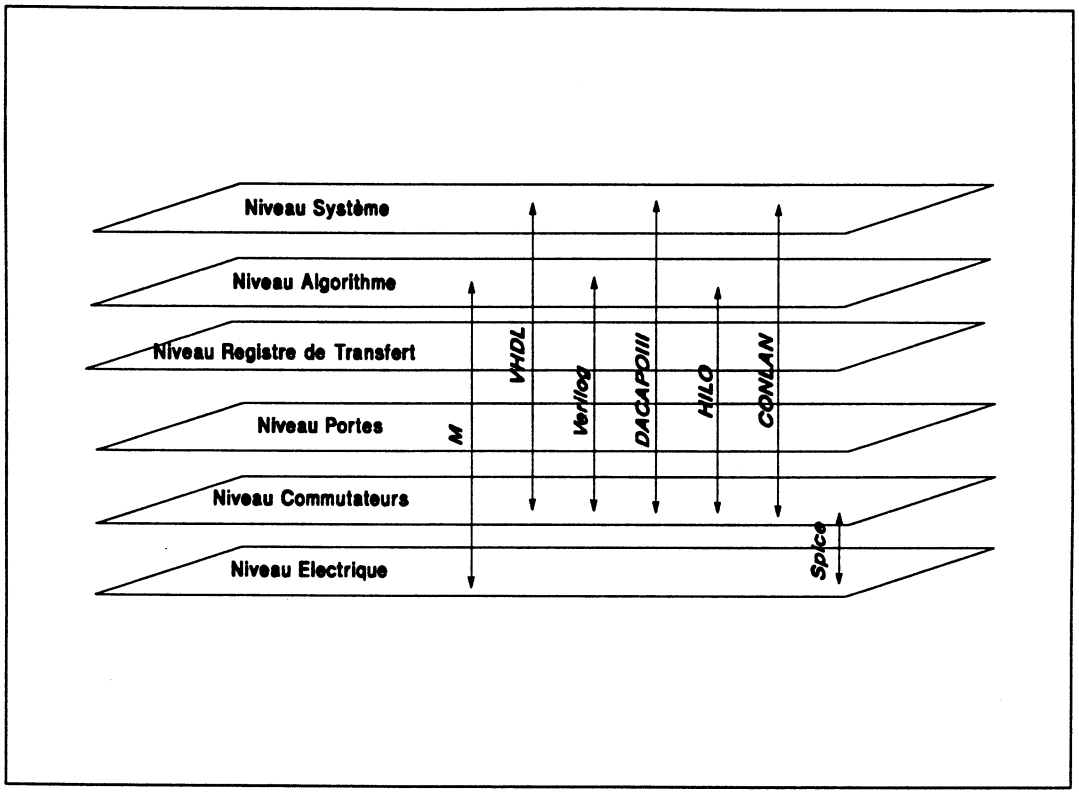

***Figure III.2.2** : Quelques langages de description de matériel avec leur taux de couverture des différents niveaux de la conception électronique.*

Notre choix, effectué en 1990, s'est porté sur le VHDL (VHSIC Hardware Description Language) car à cette époque VHDL devenait le standard des langages de description de matériel, et par la même le format interchangeable entre les différents simulateurs et donc indépendant d'un type de simulateur particulier. En effet, dès le début des années 1980 le Département de la Défense Américain (DoD) a demandé à ce que les composants utilisés dans les systèmes militaires aient leur modèle écrit en un langage de description de matériel standard. VHDL a ainsi été retenu ce qui laissait présager de son avenir. Ainsi actuellement tous les outils de conception assistée par ordinateur (CAO) offrent des possibilités de compilation, de simulation, et voire même de synthèse de modèles VHDL, si bien que notre choix s'avère actuellement judicieux.

VHDL est un dérivé du langage ADA, il se démarque de ce dernier par des instructions propres aux systèmes électroniques (contraintes temporelles, définition de structures...). Nous ne présentons ici que le principe de ce langage au travers d'un exemple de système très simple. Soit un système dont la fonction est d'effectuer soit l'addition, la soustraction, la multiplication ou la division de 2 opérandes A et B. Le résultat est disponible sur une sortie après 1 seconde. Nous allons décrire le modèle fonctionnel comportemental de ce circuit.

Tout d'abord, il est nécessaire de spécifier l'*entité* du modèle à savoir les entrées/sorties. Elles sont au nombre de 4, une sortie S de type entier, 2 entrées opérandes A et B de type entier et une entrée F de commande de type énuméré. L'intérêt de cette entité est de connaître très rapidement les entrées/sorties du système, le composant est alors vu comme une boîte noire. Ensuite, les fonctionnalités de cette boîte noire sont décrites, précisées dans la partie *architecture* du modèle. C'est à ce niveau que les équations qui régissent le système sont écrites dans un ou des processus concurrents. Dans notre modèle, un seul processus est nécessaire à le décrire, et il s'écrit avec des instructions usuelles aux langages de programmation de haut niveau. Les signaux sont les variables globales d'un modèle, ils permettent aux processus de dialoguer entre eux, les variables sont obligatoirement internes à un processus. Une affectation de signal est toujours exécutée après un temps fixé par l'utilisateur ("after 1 s" pour le signal S) ou un temps égal au pas de simulation si aucune contrainte de temps n'est précisée. Le modèle comportemental de ce "système" est le suivant :

```
-- les mots clé VHDL sont en majuscules

ENTITY modèle_simple IS

TYPE fonction IS (add,sous,mul,div);
PORT      ( a,b: IN integer:=0;
            f: IN fonction:= add;
            s: OUT integer);

ARCHITECTURE compt OF modèle_simple IS
-- zone de définition des signaux propres à l'architecture
BEGIN
principal: PROCESS
-- zone de définition des variables (locales)
VARIABLE result_intern : integer :=0;
CASE f IS
            WHEN add => result_intern:= a + b;
            WHEN sous=> result_intern:= a - b;
            WHEN mul => result_intern:= a × b;
            WHEN div => result_intern:= a / b;
            WHEN others ASSERT false
                                    REPORT "ERREUR fonction non définie dans
                                     architecture compt de modèle_simple
                                                SEVERITY ERROR;
            result_intern:=0;
END CASE;
s<= result_intern after 1 s;
WAIT ON a,b,f;
END PROCESS principal;
END compt;
```

Le VHDL permet d'écrire des modèles structurels de systèmes. Chacune des entités des composants décrivant le système est reprise, et les interconnexions à réaliser entre les entrées/sorties de ces composants sont directement écrites grâce à des instructions de "mapping" propres au langage. Par exemple, le modèle créé précédemment peut être réutilisé 3 fois afin de créer le modèle structurel conforme à la figure III.2.3. Ce modèle est le suivant :

```
ENTITY structure IS

TYPE fonction IS (add,sous,mul,div);
PORT    ( a1,a2,b1,b2: IN integer:=0;
          f: IN fonction:= add;
          s: OUT integer);

ARCHITECTURE 3_modèles OF structure IS

-- zone de définition des composants à interconnecter
component modèle_simple
PORT    ( a,b: IN integer:=0;
          f: IN fonction:= add;
          s: OUT integer);

-- zone de définition des signaux internes
signal s1, s2 : integer:=0;

-- zone de définition des modèles utilisés
for M1 :modèle_simple use entity work.modèle_simple (compt);
for M2 :modèle_simple use entity work.modèle_simple (compt);
for M3 :modèle_simple use entity work.modèle_simple (compt);

BEGIN

-- les interconnexions sont figées à ce niveau
M1 : modèle_simple
          port map(a1,b1,f,s1);
M2 : modèle_simple
          port map(a2,b2,f,s2);
M3 : modèle_simple
          port map(s1,s2,f,s);

END 3_modèles;
```

Le lecteur désirant plus de détail sur le VHDL se reportera à [ROU91] pour une rapide présentation du langage, à [LIP89, IEE88] pour la description du langage et à [AIR90] pour ce qui est de la méthodologie de modélisation au travers d'exemples concrets.

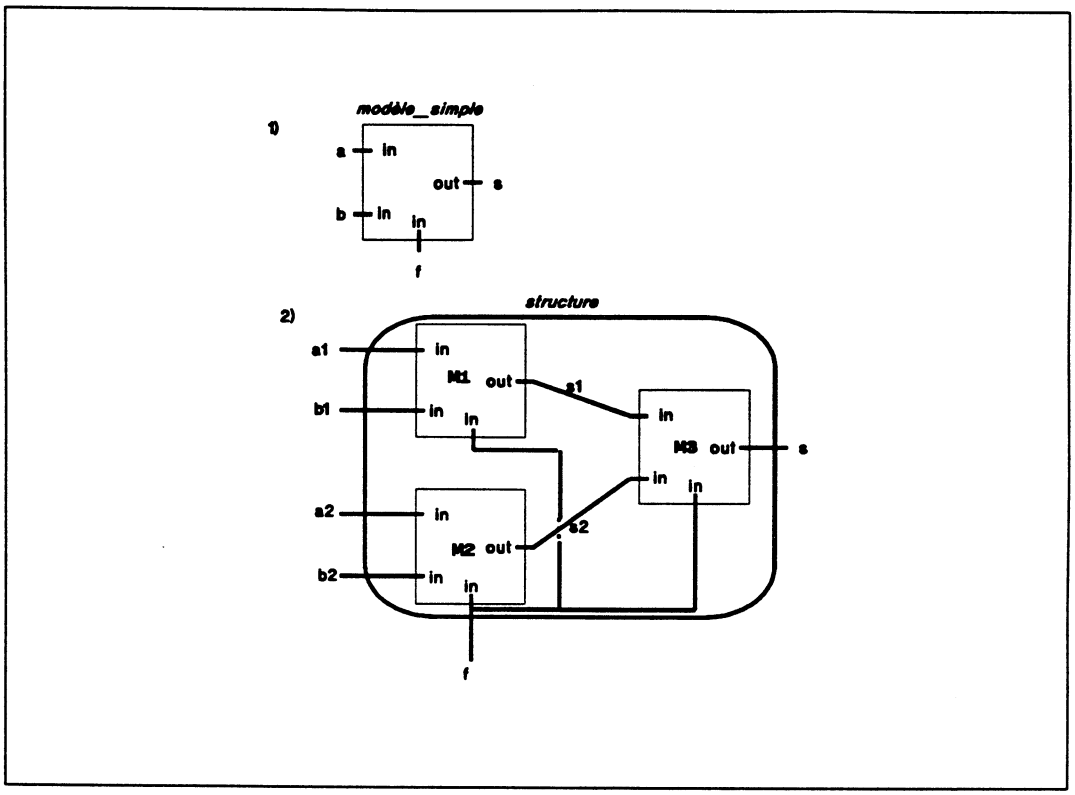

***Figure III.2.3** : Entité du modèle de notre bloc logique, représentation de la structure composée par 3 composants.*

Bien évidement ce langage est depuis deux ans très critiqué, notamment pour les domaines pour lesquels il n'a pas été conçu a priori telle la synthèse électronique. Sont dressés ci-dessous les avantages et les inconvénients actuels de ce langage.

Comme avantages du VHDL, on peut citer :

- le VHDL est un standard ce qui le rend indépendant des outils de CAO, si bien que les modèles réalisés sont exportables;

- il couvre tous les niveaux de la conception électronique et peut ainsi supporter les différentes étapes de la conception descendante;

- la notion de processus concurrents aide énormément à la modélisation.

Par contre, il est reproché au VHDL les défauts suivants :

• le VHDL est un langage complexe, et donc difficilement abordable notamment par les électroniciens ne connaissant pas d'autres langages de programmation de haut niveau;

• le VHDL convient à la simulation mais n'a pas été conçu pour la synthèse, le test, et la preuve formelle alors que les fournisseurs d'outils CAO proposent VHDL comme entrée pour la synthèse logique par exemple. Seul un sous-ensemble du standard VHDL s'avère synthétisable, malheureusement ce sous ensemble est différent d'un outil de CAO à un autre, donc à quand le VHDL synthétisable standard? ("synthesisable VHDL-1076 subset");

• Pour ce qui est de la description structurelle de haut niveau, il serait préférable de disposer d'une couche graphique supérieure afin de créer une représentation visuelle du système;

• il s'avère que la simulation d'une description au niveau portes est très longue par rapport à d'autres solutions;

• lors des opérations de synthèse, le VHDL n'entretient pas un fichier de rétro-annotation permettant de tenir compte de la technologie cible au niveau des retards, comme le fait VERILOG par exemple;

• VHDL étant un langage fermé, il n'est pas possible actuellement d'intégrer des blocs "confidentiels" dans le modèle d'un système, de même qu'il n'est pas possible de faire appel à des routines écrites dans un autre langage;

• actuellement, le standard VHDL 1076 n'est pas supporté à 100% par les différents outils de CAO ce qui rend l'exportabilité des modèles parfois délicate. Notons toutefois que fin 1992, tous les grands fournisseurs de compilateurs/simulateurs VHDL supporteront le langage à 100%.

Le standard 1076 VHDL est soumis à une révision tous les 5 ans, si bien que l'on peut espérer voir cette liste de défauts rétrécir comme une peau de chagrin au cours du temps.

III.2.3 : La plate forme de simulation d'architectures.

Notre objectif est de disposer d'un simulateur d'architectures dédié au traitement d'images. En ce qui concerne la modélisation des architectures, elle doit s'effectuer en VHDL et à n'importe quel niveau de description. Le compilateur/débogeur/simulateur VHDL utilisé est celui commercialisé par la société "Mentor Graphics". Ce simulateur présente l'avantage de supporter la simulation de modèles composés de modules de tous types (pas forcément du type VHDL). Au début de notre campagne de modélisation, le standard VHDL_1076 n'était pas accepté par l'outil, des opérations tels la généricité ou les TEXTIO n'étaient pas disponibles, il fut alors nécessaire de combler ces carences en développant des programmes propres à l'environnement Mentor Graphics.

Pour rendre ergonomique notre simulateur d'architectures, une couche logicielle supérieure a été développée également (figure III.2.4). Ainsi, l'architecture est directement programmable en un langage de haut niveau grâce au compilateur interfacé au simulateur (pour notre cas, ce langage est le 4LP), les images sont visualisables en entrée et en sortie si bien que le simulateur VHDL est transparent pour le programmeur de la structure. Seul le concepteur d'architecture intervient à ce niveau pour tracer et lister des signaux. L'intérêt du VHDL est de pouvoir travailler sur des signaux de type énuméré ce qui rend le débogage du modèle moins contraignant (figure III.2.6).

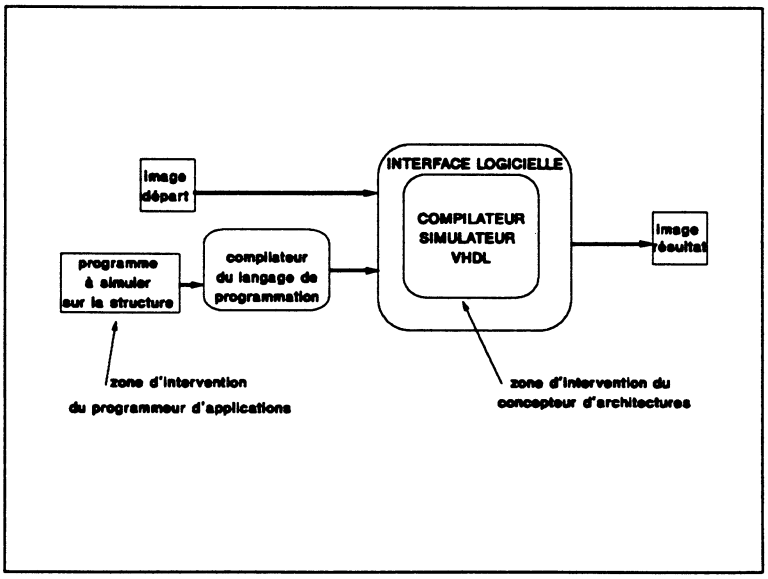

Figure III.2.4 : L'organisation logicielle de notre simulateur d'architectures.

De plus, le compilateur/simulateur VHDL supporte les opérations de la conception descendante, il est ainsi possible de passer de la description de l'architecture à sa mise en oeuvre électronique grâce à un outil de synthèse VHDL. Cet outil transforme chaque bloc

VHDL en son équivalent en portes logiques dans une technologie fixée par l'utilisateur. Ainsi, le concepteur d'architectures mesure directement les conséquences des modifications d'architecture (fonctionnalités supplémentaires) en coût électronique, les propositions d'architectures deviennent alors réalistes et convaincantes (figure III.2.5).

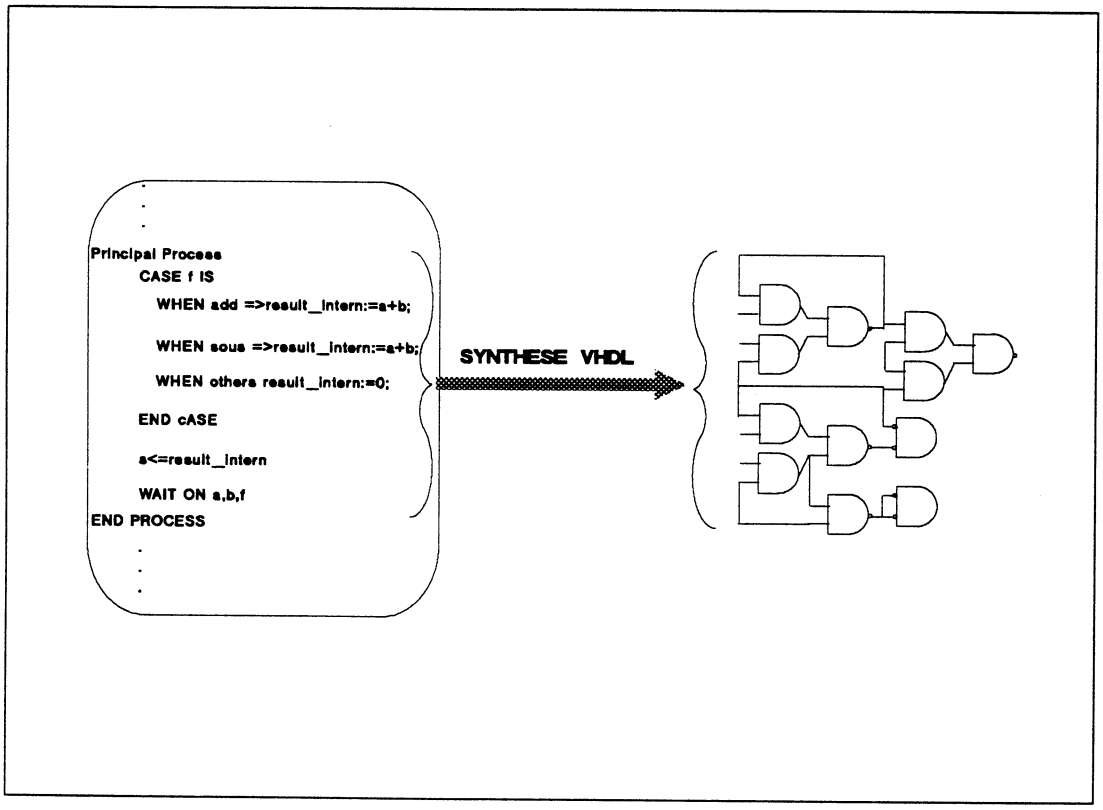

***Figure III.2.5** : L'outil de compilation et de simulation VHDL offre la possibilité d'effectuer la synthèse VHDL des modèles créés et validés. Le concepteur d'architecture peut ainsi mesurer les conséquences au niveau du coût électronique des modifications évaluées.*

La plate-forme d'évaluation d'architectures utilisée pour évaluer SYMPATIX, présente les avantages suivants :

elle est utilisable pour effectuer les développements matériels et logiciels. Pour ces derniers, l'utilisateur n'a pas à connaître VHDL, il se contente de programmer en un langage de haut niveau (ici le 4LP);

l'architecte de systèmes dédiés au traitement d'images peut directement mesurer les conséquences, au niveau de la qualité des images résultat, des idées qu'il a modélisé en VHDL (figure II.2.7);

pour une architecture donnée, il est rapide d'en connaître son schéma électrique, donc sa consommation, sa surface ... ce qui donne facilement une idée de sa faisabilité;

le fait de pouvoir travailler sur des signaux de type énuméré permet d'accélérer le temps de développement des modèles;

enfin, tous les outils de CAO électronique sont connectés à la plate-forme, tel le simulateur de fautes.

Aussi, cet outil d'évaluation d'architectures de traitement d'images permet de tester, de valider et de proposer des systèmes, sans en supporter le développement matériel.

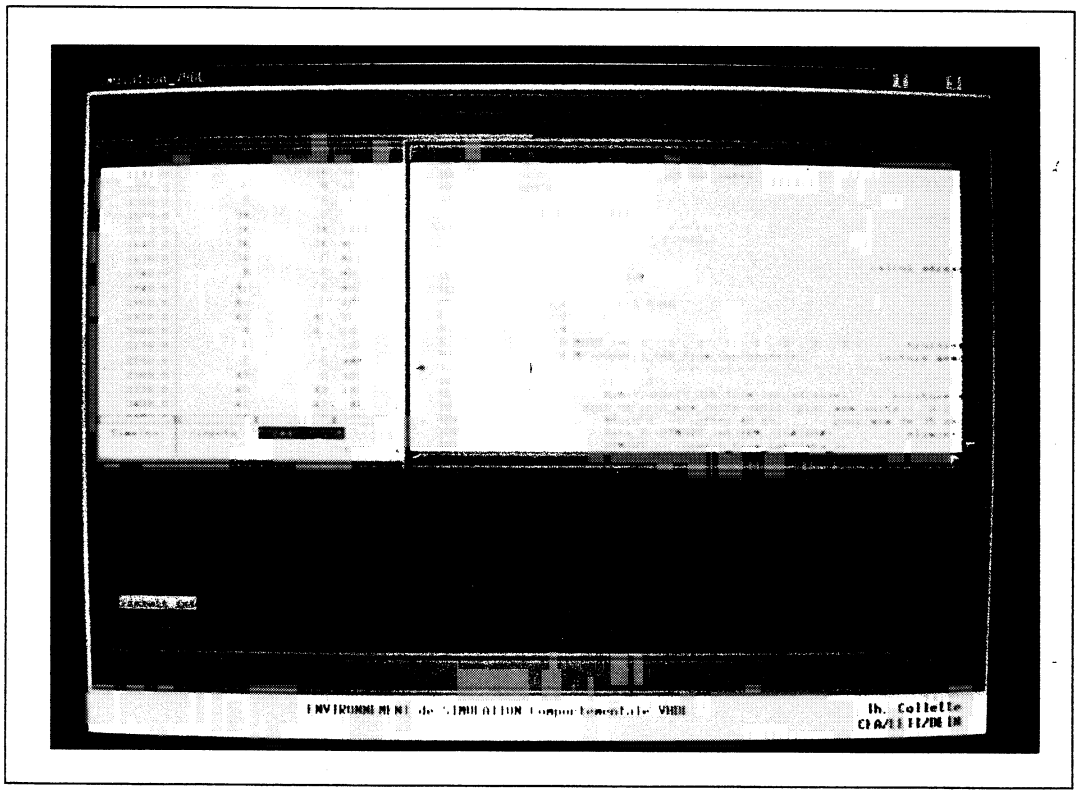

***Figure III.2.6** : Présentation du simulateur VHDL. L'intérêt est de pouvoir visualiser des traces de signaux de type énuméré, par exemple ici le type de fonction effectuée sur les processeurs de même que le type de registre de travail sont tracés sous forme énumérée (mul, add, s5, S0...).*

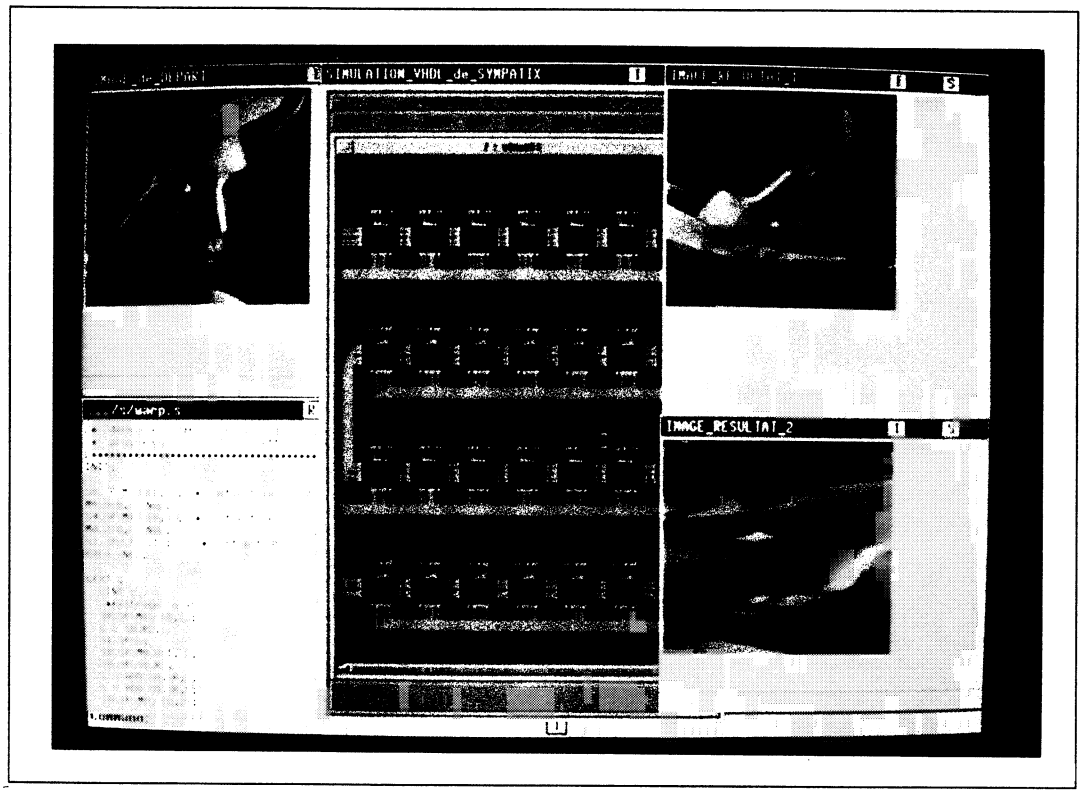

***Figure III.2.7** : Le simulateur d'architectures. Une image ainsi qu'un programme de haut niveau (4LP), le résultat, s'il est sous un format image, est affiché à l'écran.*

III.3 : LA MODELISATION DE SYMPATIX.

Nous présentons dans ce paragraphe notre démarche utilisée pour concevoir le modèle VHDL de SYMPATIX. Ce modèle est en fait constitué de deux parties électroniques distinctes, une première connue jusque dans sa structure (SYMPATI2), la seconde à définir (le réseau intelligent ouvert). Nous montrons ici comment ont été abordées les modélisations de ces deux parties. L'élaboration du second modèle est bien détaillée ainsi que ses fonctionnalités retenues.

III.3.1 : La méthodologie de conception du modèle de SYMPATIX.

Du fait que SYMPATIX est composé et de SYMPATI2, partie bien figée, et du réseau intelligent, partie à développer (figure III.3.1), l'élaboration du modèle de SYMPATIX s'est effectuée en 3 étapes. Dans un premier temps, le modèle de SYMPATI2 a été écrit ensuite validé, puis les fonctionnalités du réseau intelligent ouvert ont été modélisées, enfin le modèle

VHDL complet de SYMPATIX a été construit ce qui nous a conduit à compléter, par une approche incrémentale, le modèle du réseau intelligent ouvert. Cette construction s'est effectuée grâce à des évaluations et des validations de la structure, par différents algorithmes de traitement d'images de moyen niveau entre autres.

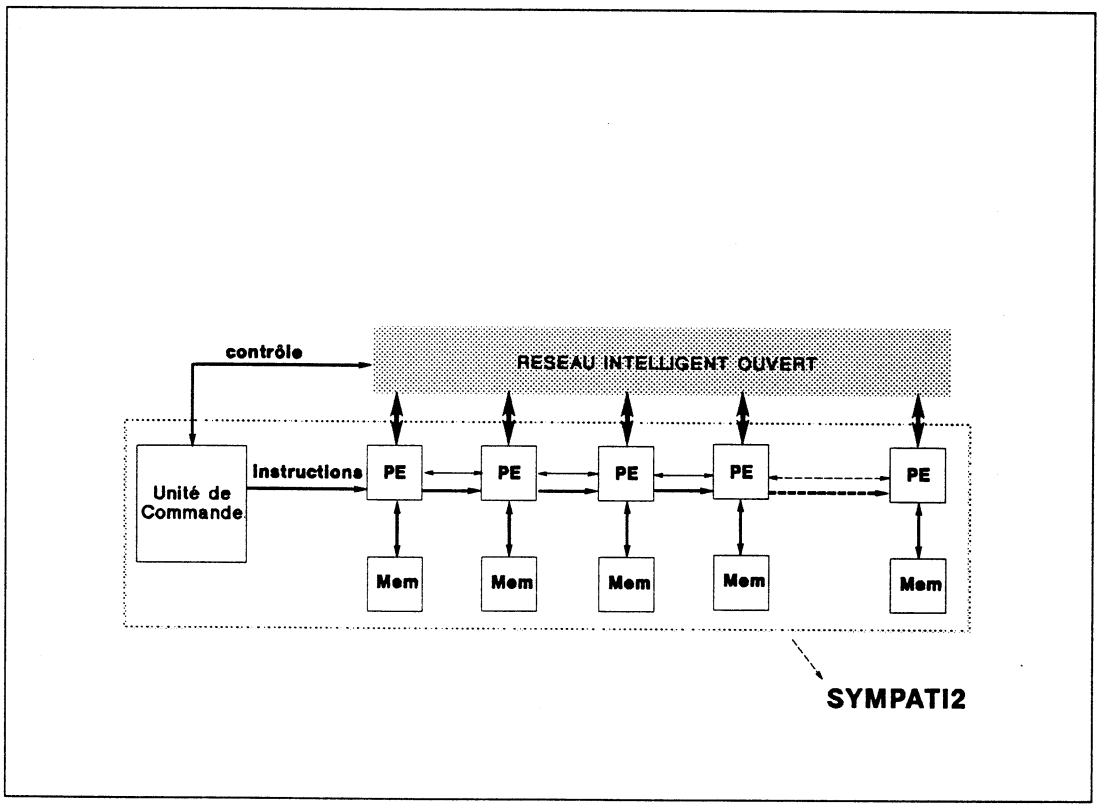

***Figure III.3.1** : Le modèle structurel de SYMPATIX. Il est composé de SYMPATI2, le système existant, et du réseau intelligent ouvert (RIO) partie à définir et à valider.*

Voyons dans un premier temps la méthodologie utilisée pour concevoir le modèle VHDL de SYMPATI2. Ce modèle est présenté sous forme structurelle associant 3 types de circuits, l'unité de commande, le processeur élémentaire (PE) et une mémoire (Mem). Il y a autant de processeurs élémentaires que de mémoires, leur quantité dépend donc du nombre de processeurs de la structure à simuler. Dans un premier temps, nous sommes partis sur un modèle de SYMPATI2 à 32 processeurs. Détaillons maintenant le modèle du processeur élémentaire.

Une solution aurait été de définir un modèle structurel de haut niveau du processeur élémentaire, modèle conforme à la figure II.1.3 du chapitre II. Puis chacun des blocs connectés par ce modèle structurel (ALU, Scratch Pad, Réseau d'interconnexion, module de calcul d'adresses...) aurait alors été décrit par un modèle comportemental, l'ensemble aurait

ainsi formé une description du processeur au niveau transfert de registres. Cette solution n'a pas été retenue car elle fige la structure du processeur ce qui n'est pas souhaitable s'il s'avère nécessaire d'effectuer des simulations avec différentes structures telles des bus plus larges, un nombre de registres différent... Afin de ne pas s'interdire ce type de modifications, le processeur élémentaire de SYMPATI2 a été décrit par un ensemble de processus concurrents (niveau algorithme). Toutefois, ces processus, au nombre de 9, s'inspirent fortement de la découpe structurelle du processeur élémentaire, car cette découpe influe sur le comportement.

Le second modèle est celui de l'unité de commande qui lui ne s'inspire pas de la découpe structurelle du module physique correspondant, il n'est constitué que d'un seul processus déroulant les programmes exécutés sur les processeurs de SYMPATI2. Ce processus récupère les micro-instructions du programme issues du compilateur de la couche logicielle supérieure, transforme ces micro-instructions au format accepté par les processeurs et gèrent également le balayage des images traitées.

Enfin, un modèle de mémoire de type RAM a été créé pour être interfacé au processeur. Le modèle de cette mémoire n'est constitué que d'un seul processus, le temps d'accès en écriture ou en lecture est de 40 ns.

Dans le modèle complet de SYMPATI2, les paramètres propres aux processeurs et à l'unité de commande sont définis dans un fichier de configuration initialisant différents signaux du modèle.

Ensuite, une fois le modèle de SYMPATI2 élaboré et vérifié grâce à des simulations, le réseau intelligent ouvert a été à son tour modélisé. Le réseau étant composé de cellules, autant que de processeurs, un modèle de la cellule a été élaboré en premier, puis plusieurs de ces modèles ont été assemblés entre eux de manière structurelle pour former le réseau. Chacune de ces cellules est connectée à un processeur, par les bus d'adresses et de données (figure III.3.2). Un décodage d'adresses est nécessaire afin de déterminer si le processeur effectue un accès mémoire ou bien un accès réseau.

Bien évidement, à ce stade de l'étude, seules les fonctionnalités de la cellule du réseau étaient connues, non sa structure interne si bien que la modélisation s'est effectuée au niveau algorithme. Nous avons repris les fonctionnalités exigées suite à l'étude algorithmique du chapitre précédent pour établir ce que l'on appellera le "premier jet" du modèle du réseau intelligent. Notons, que ce réseau est commandé par une unité de commande centralisée, unité qui par la suite, sera fondue avec l'unité de commande des processeurs de SYMPATI2.

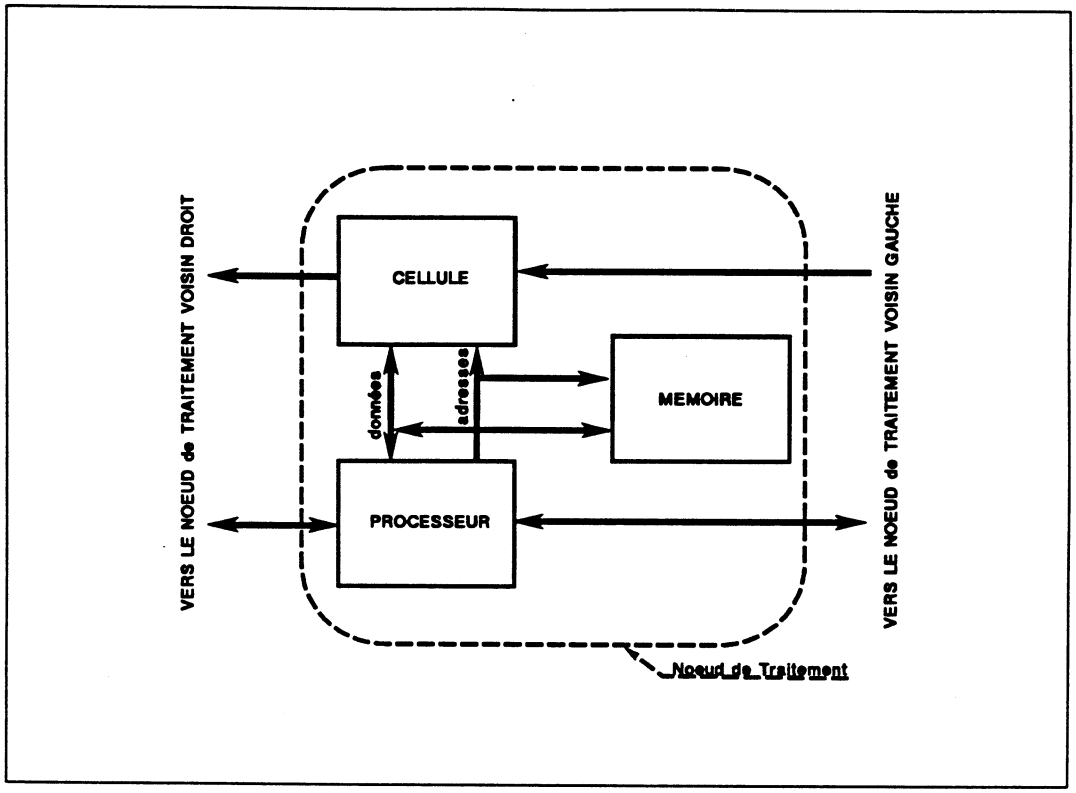

***Figure III.3.2** : Principe de l'interconnexion entre le processeur, sa mémoire, et sa cellule. L'ensemble forme alors un noeud de traitement de SYMPATIX. Toutes les cellules sont connectées entre elles pour former le réseau intelligent ouvert.*

Arrivés à ce stade, il nous fallait faire cohabiter le modèle de SYMPATI2 et ce premier jet du modèle du réseau intelligent. Comment intégrer, en 4LP, des instructions d'envoi de données dans le réseau? Ces écritures s'effectuent au niveau des processeurs grâce au mode d'adressage indexé (ou mode tabulaire). Pour ce mode, le processeur compose le champ adresse du message à envoyer sur le réseau par affectation de son registre d'index (il positionne les registres LIG et COLO lesquels représentent respectivement les bits de poids fort et de poids faible du registre d'index). Donc cette écriture pourrait, simplement, correspondre à une opération d'écriture en mode tabulaire, mais il est nécessaire d'indiquer à l'unité de commande qu'une opération d'écriture dans le réseau va être effectuée, ceci afin d'autoriser ou non une écriture selon l'état du réseau. Ainsi chaque instruction d'écriture réseau, codée grâce à un accès tabulaire, est précédée d'une instruction non exécutée par les processeurs mais interprétée par l'unité de commande afin qu'elle commande les décalages. Un programme 4LP dans lequel une opération d'écriture réseau est effectuée, se présente de la manière suivante:

```
PROG ENV_RES;

INIT;

LEXT;

        LINT

        /* instructions classiques

        S0.SP  = FCT(2) S0.SP,L; /* instruction non exécutée sur les processeurs,
                        /* mais interprétée par l'UC
        TAB    = S6.SP; /* bits de poids faibles du message
        TAB    = S7.SP; /* bits de poids fort du message

        /* instructions classiques

LEND;
STOP;
```

Dans le paragraphe suivant, nous reviendrons sur la modélisation du réseau intelligent car elle a évolué au cours des différentes simulations effectuées. Après ce premier contact avec le VHDL, notons quelques difficultés rencontrées. Tout d'abord, la première fut rencontrée lors de l'élaboration du processus asynchrone du modèle du processeur élémentaire modélisant le réseau d'interconnexion interne au processeur. En effet ce réseau n'est composé physiquement que de logique combinatoire, il était dont préférable de rendre ce processus asynchrone c'est-à-dire sensible sur toutes ses entrées, et n'ayant pas d'horloge de séquencement. Typiquement, ce type de processus conduit à générer de la logique combinatoire. Un processus sophistiqué a donc été écrit, et sa mise au point fut difficile. Pour la modélisation de l'unité de commande, l'impossibilité d'effectuer des entrées/sorties sur fichiers (TEXTIO) nous a pénalisé et contraint à développer du logiciel dépendant de notre environnement ce qui rend notre modèle plus délicat à exporter. Cette limite qui existait sur la première version de notre simulateur a disparu dans les versions suivantes, si bien qu'il peut être envisagé de réécrire le modèle de l'unité de commande, modèle qui deviendrait alors plus facilement exportable.

III.3.2 : Elaboration du modèle de SYMPATIX.

Le paragraphe précédent présente l'écriture du modèle de base de SYMPATIX. Le modèle VHDL de SYMPATI2, processeurs élémentaires, mémoires et unité de commande dédiée aux processeurs n'évoluera plus au cours de cette étude. Par contre, il faut, à ce stade, valider SYMPATIX, et par là même le réseau intelligent, sur des algorithmes de traitement

d'images de moyen niveau. Cette phase nous à conduit à augmenter, et à modifier les fonctionnalités du premier jet du modèle du réseau intelligent ouvert.

Le premier jet du modèle de la cellule du réseau intelligent ouvert comprend un processus d'écriture réseau (processus ecriture_reseau), un processus de décalage et de récupération des messages (processus recup_data), un processus de lecture des messages reçus (processus lecture_fifo) et un ensemble de processus effectuant l'interfaçage avec l'unité de commande. Dans un premier temps, il n'a été envisagé de récupérer les messages que dans une mémoire locale à la cellule, mémoire de type FIFO. Les fonctionnalités de ce premier modèle du réseau intelligent ouvert sont les suivantes :

- le RIO est interfaçable avec les processeurs élémentaires, il est sollicité en écriture pour un ensemble d'adresses définies;

- Les messages écrits dans le réseau sont de type point à point, le numéro du processeur destinataire est alors précisé dans le champ adresse du message ou de type diffusion, c'est alors une adresse particulière qui doit être générée;

- La mémorisation des messages ne se fait que dans des mémoires locales à chaque cellule;

- Une fois le réseau vide, une indication est envoyée vers l'extérieur, le réseau peut alors supporter une nouvelle écriture réseau. De même, lorsque qu'au moins une mémoire locale de type FIFO d'une cellule est pleine, un signal est généré à l'extérieur afin d'interdire d'autres écritures, et afin d'envoyer une séquence de lecture des FIFO.

Une fois que ce modèle rudimentaire de SYMPATIX fut terminé, nous nous sommes lancés directement dans la simulation d'algorithmes dans le but de compléter le modèle et de l'améliorer. Les améliorations apportées sont de deux types, elles interviennent soit au niveau des fonctionnalités du système, soit au niveau de la structure du système. Ces modifications sont présentées ci-dessous.

La première limitation apportée intervient sur la structure. En effet, il est fréquent, en traitement d'images moyen niveau, de travailler sur des coordonnées de points plus que sur des niveaux de gris si bien qu'il s'avère nécessaire d'avoir un champ de données capable d'accepter le couple de coordonnées (x,y). Considérant le processeur actuel dont le bus de

données externe n'est que de 8 bits, il est préférable de limiter les tailles image à 256×256 points, les coordonnées (x,y) d'un point sont alors contenues sur 16 bits, donc extériorisées par deux écritures. De façon plus générale, le champ de données doit être d'une largeur égale au double de celle du bus de données des processeurs. Pour générer un message, si le champ données est complet, alors 2 écritures réseau sont nécessaires. De même, lors de la récupération, 2 lectures, une pour x et l'autre pour y sont nécessaires également.

L'inconvénient de la méthode précédente est qu'il est obligatoire de calculer dans le programme x et y. Or, lors d'un balayage de l'image en mode hélicoïdal, ces valeurs peuvent directement être calculées à partir de données envoyées par l'unité de commande aux processeurs. En effet pour les opérations de traitement d'images de bas niveau, l'image est balayée par les processeurs, et c'est l'unité de commande qui assure ce balayage en envoyant aux processeurs les coordonnées du segment des points à traiter, appelé segment courant. Il est donc intéressant de calculer automatiquement, à partir de ces données envoyées par l'UC, les coordonnées (x,y) du point courant pour le processeur associé. Toutefois, afin de limiter les entrées/sorties de la cellule, ces coordonnées sont calculées à partir de l'adresse générée par le processeur.

Ensuite, lors de l'exécution de longs programmes, nous nous vîmes confrontés à un problème intervenant dès qu'une des FIFOs se trouve pleine. Quand cette situation intervient, il est alors nécessaire de stopper le programme en cours d'exécution et de vider les FIFOs. Il est intéressant, lors de cette seconde opération de ne pas effectuer un simple transfert des données reçues par le réseau sur la FIFO vers la mémoire processeur, mais de traiter directement les données contenues dans les FIFOs. Pour cela, un sous programme est exécuté dès qu'une FIFO se trouve pleine, le contexte du programme principal doit être sauvegardé. Les registres de balayage, propres à l'unité de commandes, sont sauvegardée dans cette unité tandis que les registres et les drapeaux des processeurs doivent être sauvegardés par les processeurs. Une première solution aurait été d'effectuer cette sauvegarde dans des registres de la cellule, mais cette solution présente des problèmes pour la sauvegarde des registres LIG et COLO utilisés pour adresser la cellule. Une seconde solution est de demander à l'unité de commande de dérouler un programme de sauvegarde en mode hélicoïdal dans une imagette. C'est cette seconde solution qui a été retenue car elle permet une sauvegarde exhaustive en 27 instructions, soit 3.5 µs pour la version actuelle, mais monopolise une image de travail. De même, la restitution du contexte s'effectue une fois les FIFOs vides en 34 instructions soit en 4 µs. Ainsi, avec cette nouvelle fonctionnalité, deux programmes sont à charger lors de l'initialisation du système, l'un travaillant sur les images, l'autre sur les listes de points stockées dans les FIFOs des cellules.

Pour certaines applications, le stockage des points reçus dans une FIFO n'est pas très intéressant, notamment pour les applications de transformation géométrique ou de génération graphique, opérations sortant du domaine d'application visé à l'origine. Au cours de ces opérations, un processeur calcule les coordonnées d'un point de l'image résultat, si bien qu'il est intéressant de ne pas passer par une FIFO en réception, mais d'écrire directement la donnée transmise dans la mémoire du processeur associé, on parle alors d'écriture DMA (Direct Memory Acces). Dans ce cas, le champ adresse du message contient l'adresse de stockage dans la mémoire. Une variante de ce mode est de rendre la cellule plus intelligente, à savoir plutôt que de demander au processeur de créer l'adresse hélicoïdale de stockage, celle-ci peut être directement calculée dans la cellule, de même que le numéro du processeur destinataire, le processeur se trouve alors libéré de cette servitude.

Enfin, la possibilité d'effectuer la diffusion des données de manière électrique est possible. Les cycles de transfert sont plus longs que la période de décalage car il faut intégrer le temps de la propagation électrique sur le bus commun. Ce temps est proportionnel au nombre de processeurs, et ici nous avons considéré, en grossière approximation, que le temps de diffusion d'une donnée est égale à deux fois la période d'horloge de décalage du réseau. Ce mode n'est intéressant, par rapport à la solution par décalages successifs, que si moins de la moitié des processeurs ont une donnée à diffuser. Dans ce cas, les cellules sont court-circuitées et les cellules ayant une donnée à émettre le font chacune leur tour grâce à un jeton leur permettant d'accéder au bus alors commun, toutes les cellules se positionnant en mode réception. Notons que la solution de diffusion par décalages permet de mixer et l'envoi point à point et la diffusion de données.

III.3.3 : Le modèle retenu.

Suite aux différentes simulations effectuées, afin de concevoir le modèle du réseau intelligent ouvert par une approche incrémentale, nous avons obtenu le modèle de la cellule qui, pour le domaine d'application visé, nous semble le mieux adapté. Nous reprenons ici les caractéristiques de ce modèle final qui sera utilisé pour la conception descendante du circuit électronique correspondant à la cellule du réseau intelligent ouvert.

Tout d'abord, ce circuit est présenté sous forme d'une boîte noire qui correspond à l'entité du modèle VHDL de la cellule. Des signaux d'entrée, de sortie et de commande sont connectés à cette entité (figure III.3.3). En entrée, il y a les commandes issues de l'unité de contrôle (ucin), les adresses issues du processeur associé (adrpein) et le message entrant dans la cellule (min). En sortie, un ensemble de données retournent vers l'unité de contrôle (dcout)

et un message est envoyé à la cellule voisine (mout). Enfin en entrées/sorties, il n'y a que le bus de données du processeur associé (datape).

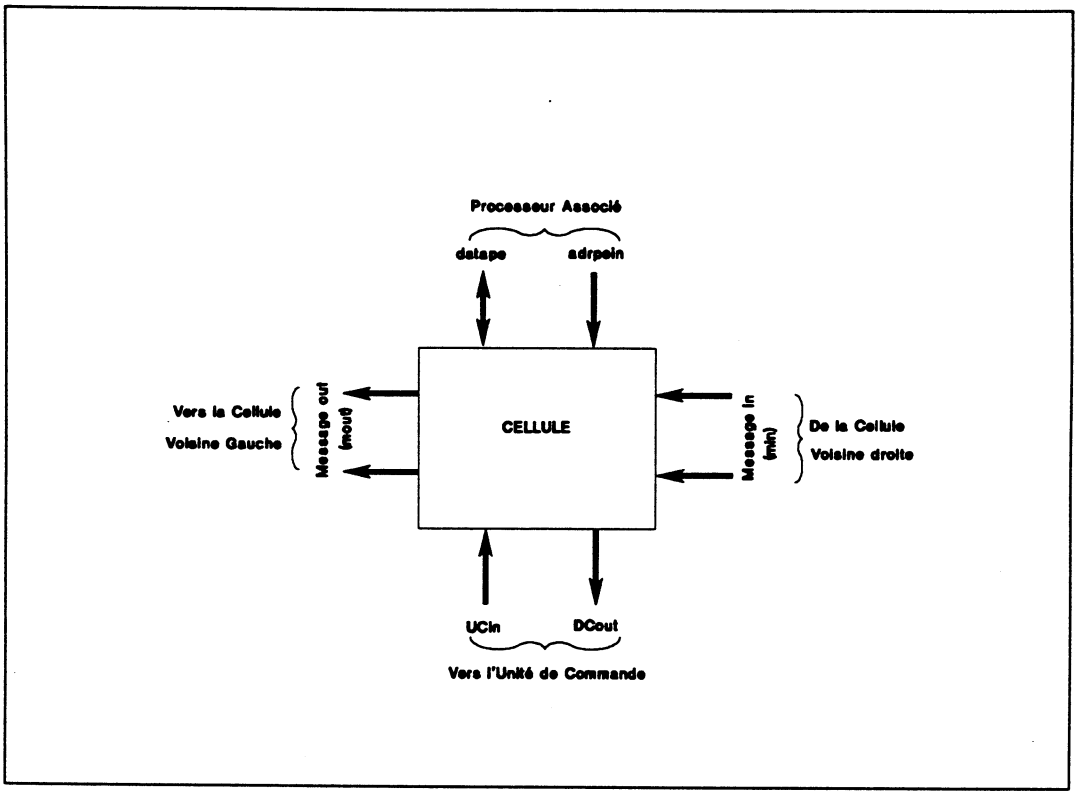

Figure III.3.3 : L'entité de la cellule du réseau intelligent ouvert.

Les fonctionnalités remplies par l'unité de commande sont très simples. Cette dernière génère et envoie aux cellules les signaux contenus dans "ucin" en tenant compte du retour de l'état du réseau contenu dans le "dcout". L'unité de commande n'autorise une écriture dans le réseau par les processeurs que quand le réseau est vide. De même elle indique aux cellules si l'échange effectué fait appel à des stockages en mode d'accès DMA. Le programmeur doit spécifier dans son algorithme le type de service demandé au réseau grâce à de nouvelles instructions mettant en oeuvre le mode d'adressage indexé. Ces nouvelles instructions sont appelées, FRES, DRES, BRES et SRES et correspondent aux possibilités d'interconnexion du réseau. Décrivons ces possibilités :

- FRES : mode point à point ou diffusion de message avec le positionnement du numéro du processeur destinataire sur les 8 bits de poids faible de adrpein et stockage des données dans une FIFO au niveau de chaque cellule;

• DRES : mode point à point avec stockage des données en mode DMA dans les mémoires associées aux processeurs. Cette possibilité admet 2 variantes exclusives :

• Solution A : le processeur associé à la cellule calcule le numéro du processeur destinataire du message, ainsi la cellule récupère ce numéro sur les bits de poids faibles du bus adresse du processeur. Les adresses DMA, calculées par le processeur associé, sont, quant à elles, envoyés via ce bus de données, et 4 écritures sont alors nécessaires pour construire le message complet;

• Solution B : le processeur émetteur du message positionne l'adresse du point mémoire à envoyer, c'est alors la cellule qui détermine le numéro du processeur destinataire et l'adresse de stockage DMA dans la mémoire associée à ce processeur. Seulement 2 écritures sont nécessaires pour former un message contenant une donnée sur 16 bits, et surtout les processeurs sont libérés de ces tâches, ingrates, gourmandes, et répétitives, de calcul du processeur destinataire et de l'adresse de stockage dans la mémoire de ce processeur;

• BRES : mode de diffusion de messages par propagation électrique;

• SRES : mode point à point en synchrone, tous les processeurs envoient une donnée à une distance figée dans la micro-instruction. Dans ce cas, aucun champ adresse n'est généré;

Suivant l'un de ces modes, les cellules doivent réaliser les opérations de construction des messages, de décodage de messages, de stockage des messages dans les FIFOs ou dans les mémoires, de court-circuitage... De plus, elles doivent générer des signaux vers l'unité de contrôle (FIFO pleine par exemple), restituer les valeurs contenues dans les FIFOs lors des opérations de lectures de celle-ci et bien entendu, à chaque pas de décalage, elles propagent le message vers la cellule voisine.

Notons également qu'il est possible aux processeurs de récupérer automatiquement les coordonnées du point courant dans la cellule ceci afin d'éviter d'effectuer ces calculs de x et de y dans le programme, ce qui monopolise des registres et ralentit le processeur. La récupération de x et y par les processeurs s'effectue par des lectures à des adresses

particulières. A terme, cette possibilité doit directement être intégrée dans le processeur. Une écriture à une adresse particulière charge alors automatiquement x et y dans le champ données du message. De même qu'il est possible aux processeurs de récupérer x et y, il leur est possible de récupérer le compteur de FIFO indiquant le nombre de valeurs stockées dans la FIFO.

Ainsi, l'accès au réseau est identifié grâce à un décodage d'adresses afin de détecter s'il s'agit, d'une écriture pour un envoi point à point, d'une écriture pour une globalisation, d'une lecture de la FIFO, de x, de y, du compteur, ou encore d'un chargement de x et y dans le champ données du message. Pour nos simulations, les adresses suivantes sont retenues :

Opération effectuée	ADRESSES 15 14 ... 9	8	7 ... 0		COUT maxi en INSTRUCTIONS
accès mémoire	0	X	X	X	3 (8 bits)
accès réseau stockage FIFO (FRES)	1	X	1	Numéro du processeur destinataire	4 (16 bits)
accès réseau en DMA (solution A) (DRES)	1	X	0	Numéro du processeur destinataire	6
lecture FIFO	FFFF				4 (16 bits)
lecture X	FFFE				3 (8 bits)
lecture Y	FFFD				3 (8 bits)
lecture compteur	FFFC				3 (8 bits)
X Y dans champ données (write)	FFFE				3 (Plus 1)
écriture avec DMA (solution B) (DRES)	Adresse physique du point mémoire cible				2 (16 bits)
diffusion par propagation électrique (BRES)	X (quelconque)				2 (16 bits)
Décalages synchrones (SRES)	X (quelconque)				2 (16 bits)

***Tableau III.3.1** : Adresses d'accès aux différentes possibilités du réseau intelligent ouvert.*

Les coûts en instructions présentés donnent le nombre maximal d'instructions nécessaires à l'élaboration du message et à son écriture dans le réseau. De façon générale, il est nécessaire de positionner le registre LIG, le registre COLO puis d'effectuer les écritures dans le réseau. Pour la dernière proposition (écriture DMA solution B), seules deux instructions sont nécessaires car LIG et COLO sont positionnés lors du calcul du point image d'une opération de manipulation graphique par exemple, calcul effectué indépendamment du réseau.

Voyons maintenant la partie la plus critique du réseau, à savoir la vitesse des décalages. La vitessse idéale (non bloquante) est dépendante du nombre de processeurs (M) et du nombre de micro-instructions insérées entre chaque écriture réseau (Nmi). Cette dépendance s'exprime de la façon suivante :

$$Td <= \frac{Nmi}{M} \times Tcycle \qquad \text{(III.3.3.1);}$$

avec Tcycle le temps du cycle machine de SYMPATI2 et Td la période de l'horloge de décalage. La valeur de Td correspond au cas où il est nécessaire d'effectuer M décalages après chaque écriture dans le réseau. Précisons ici, que dès que le réseau est vide, une nouvelle écriture réseau est possible ce qui peut arriver après seulement 1ou 2 décalages par exemple. Si bien que même si cette vitesse n'est pas respectée (suivant l'équation III.3.3.1), les processeurs ne sont pas forcément bloqués, ce blocage n'intervenant que lorsque le réseau n'est pas vide et que les processeurs doivent exécuter une nouvelle instruction d'écriture réseau. Pour nos simulations, nous sommes partis sur une période de décalage fixée a priori à 30 ns. Dans le chapitre suivant, l'équation (III.3.3.1) est reprise afin de déterminer les vitesses idéales de décalage.

Notons que lorsque la distribution des messages est accélérée par duplication des réseaux, une fonction supplémentaire est alors indispensable. Dans ce cas, le numéro du processeur destinataire (ND) contenu dans le message est comparé au numéro du processeur local (NL). Cette comparaison aiguille le message vers le réseau qui effectuera la distribution la plus rapide. Si la valeur absolue de (ND-NL) est supérieure au nombre de processeurs divisé par 2, alors le message est envoyé sur le réseau dont le sens de distribution des messages, s'effectue dans l'ordre décroissant des numéros de processeurs, sinon le message est envoyé dans le réseau interconnectant les processeurs en ordre croissant (figure III.3.4).

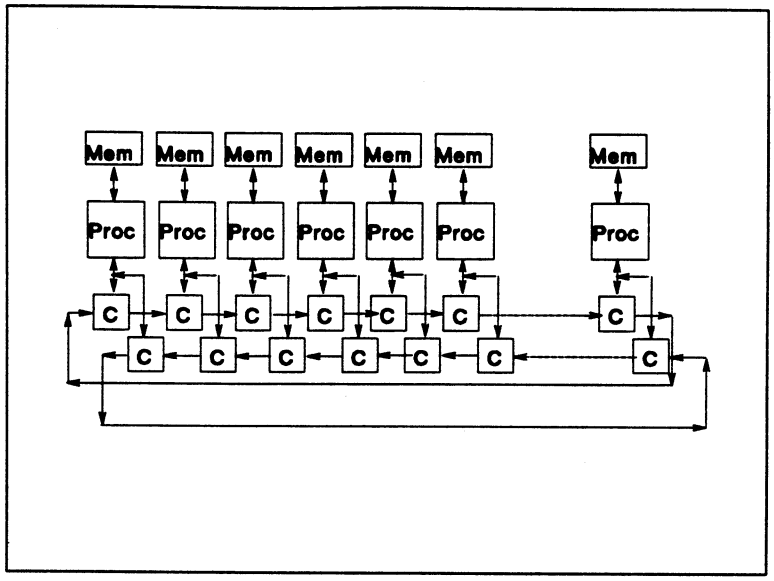

***Figure III.3.4** : Deux réseaux peuvent être utilisés pour accélérer les échanges. Dans chacun des réseaux, les messages circulent en sens opposé, et c'est un module contenu dans les cellules qui effectue l'aiguillage du message dans le bon processeur afin qu'il arrive à son destinataire le plus rapidement possible.*

Enfin pour éviter des blocages du réseau lors d'accès trop fréquents aux cellules par rapport à la fréquence de décalage, une solution serait d'utiliser une mémoire tampon, type FIFO, à l'émission. Les processeurs viendraient alors écrire directement dans la FIFO d'émission, celle-ci ce déchargeant au fur et à mesure que des "cases" disponibles se présenteraient à elle. Cette solution, qui pour certaines applications est intéressante, n'a pas été retenue pour les raisons suivantes :

- si une donnée, à échanger entre deux processeurs, est codée sur plus de bits que ne peut en contenir le champ donnée du message (16 pour notre cas), l'échange s'effectuera donc en plusieurs envois réseau. Sans FIFO d'émission, la cohérence temporelle et spatiale (en relatif) des messages est conservée, le processeur peut ainsi reconstituer ce type de message. Par contre, l'utilisation de FIFOs à l'émission vient détruire cette cohérence, les messages ne peuvent être reconstruits;

- Cette solution interdit de mélanger dans un même programme des instructions d'envoi réseau différentes telles FRES et BRES par exemple, afin de garder la cohérence entre les envois FRES et les envois BRES. En effet, si cette cohérence n'est pas respectée, le processeur destinataire ne peut effectuer la différence entre une donnée de type "point à point" et une donnée diffusée;

- le coût matériel d'une telle solution est conséquent car la taille, en largeur, des FIFOs est celle des messages.

Notons néanmoins que, pour des structures à grand nombre de processeurs, cette solution pourrait être envisagée, à la condition d'avoir la possibilité de ne pas utiliser les FIFOs en émission pour exécuter des algorithmes, où la cohérence spatio-temporelle est exigée.

Doit-on intégrer toutes ces fonctions dans le modèle final à synthétiser? Cela dépend de l'application et du coût matériel de chacune de ces fonctionnalités. De même, la taille des FIFOs de réception de messages a été fixée pour nos simulations à 256 mots, mais cette taille peut évoluer dans la suite de même que la largeur des bus. Les résultats de synthèse VHDL présentés dans le chapitre suivant aideront le concepteur à choisir la meilleure option pour son application en fonction des performances demandées et de la surface électronique mise à disposition.

III.4 : CONCLUSION.

Ce chapitre démontre la nécessité de la simulation et présente les intérêts de la conception de systèmes par une approche descendante. Dans cet objectif, un environnement d'évaluation d'architectures de traitement d'images a été développé, il permet aux concepteurs, de matériel et de logiciel, de travailler en parallèle. Le logiciel est développé en un langage de haut niveau et est directement testé sur la structure, alors que cette structure matérielle est définie tant au niveau comportement qu'au niveau structurel en un langage de description de matériel, le VHDL. Grâce à cette approche, les conséquences des modifications d'architectures sont mesurées sur des applications et en nombre de portes nécessaires à leur réalisation.

Cette expérience, originale, de conception électronique descendante de systèmes parallèles par simulation VHDL, nous permet de présenter quelques remarques :

le VHDL permet de simuler facilement des structures parallèles, mais plus la structure comporte de composants, plus le temps de simulation est long;

le temps de prise en main du langage, et d'écriture du modèle VHDL de SYMPAT2 fut de 6 mois. Il est important de noter que les premiers développements se sont effectués sur un simulateur en "version bêta";

au niveau de l'écriture des modèles, il est nécessaire de réduire au maximum le nombre de signaux afin d'accélérer les modèles, de garder une écriture cohérente d'un processus à l'autre, de travailler avec un maximum de signaux de type énuméré, d'utiliser des paquetages standards (par exemple la logique à 9 niveaux) et de générer les retards fonctionnels avec une horloge particulière.

Grâce à cette approche, le modèle comportemental de SYMPATIX a été élaboré par une approche de conception incrémentale. L'apport de chacune des modifications envisagées a ainsi été mesuré sur un ensemble d'algorithmes, leur coût électronique a pu également être évalué, ce qui nous a conduit à retenir l'architecture présentant le meilleur coût/performances pour le domaine d'application visé.

Avec cet outil, le modèle VHDL de notre calculateur parallèle dédié à la vision a été élaboré, testé, et validé, directement par des algorithmes de traitement d'images. Les performances du modèle issu de cette étude, à savoir de SYMPATIX, sont présentées dans le chapitre suivant de même que le coût électronique des différentes fonctionnalités proposées.

Chapitre IV

Résultats.

Ce chapitre présente les performances de SYMPATIX, le système multiprocesseur dédié à l'imagerie et détaillé dans les chapitres précédents de cette thèse. Deux types de résultats sont détaillés ici, ceux des simulations les plus représentatives effectuées sur le modèle VHDL de SYMPATIX donnant le temps d'exécution d'algorithmes de vision, et ceux concernant la synthèse VHDL à savoir le coût électronique de chacune des modifications envisagées. Sont présentés également quelques résultats au niveau du temps de simulation du modèle sur différents simulateurs afin de guider le lecteur désirant utiliser ce type d'approche de conception.

IV.1 : LE DOMAINE D'APPLICATION DE SYMPATIX.

L'architecture de SYMPATIX est née d'une étude algorithmique dont le but était d'évaluer les limites du calculateur SYMPATI2 pour les opérations de traitement d'images de moyen niveau. SYMPATIX est constitué de SYMPATI2 et du réseau intelligent ouvert si bien qu'en ce qui concerne les opérations de traitement d'images, l'architecture proposée supporte les opérations effectuées par SYMPATI2, en accélère certaines, notamment celles de traitement d'images de moyen niveau, de génération graphique et étend le domaine d'application de ce type d'architecture (figure IV.1.1).

Sont présentées dans un premier paragraphe les performances obtenues pour les opérations de traitement d'images de moyen niveau, puis un second paragraphe donne les possibilités et quelques résultats d'opérations de manipulation d'images (ou opérations graphiques).

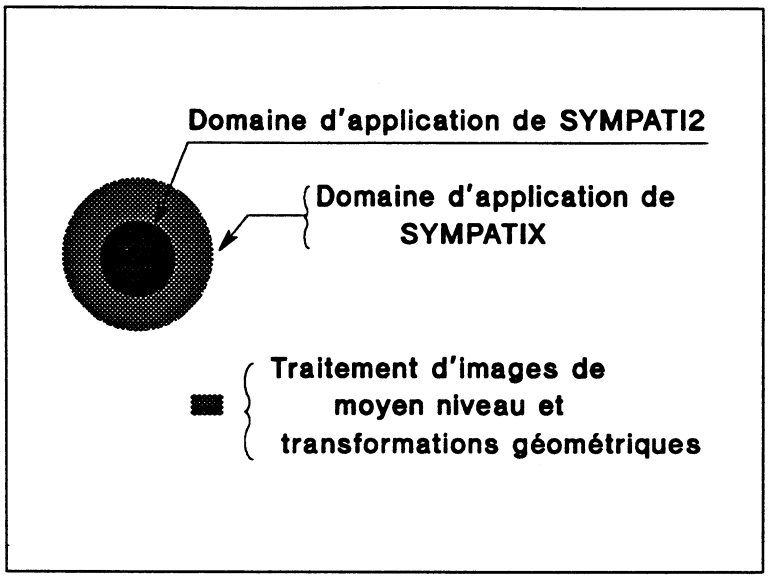

***Figure IV.1.1** : Comparaison des domaines d'application de SYMPATI2 et de SYMPATIX.*

IV.1.1 : SYMPATIX et les opérations de traitement d'images.

Le réseau intelligent ouvert offre, aux processeurs de SYMPATIX, la possibilité d'effectuer entre eux, des échanges asynchrones. Ce réseau n'est donc sollicité que lors des opérations réclamant ce type de communication. Ainsi les algorithmes travaillant au niveau iconique ne profitent pas du réseau intelligent ouvert si bien que SYMPATIX effectue ces opérations avec la même rapidité que SYMPATI2. Notons néanmoins que le réseau apporte la possibilité d'acquérir et de restituer plus rapidement qu'actuellement les images, de même il permet de redistribuer les images sous une autre forme et accélère les opérations mettant en oeuvre un grand voisinage, tel un masque de convolution 15×15 grâce à son mode de décalage synchrone (SRES).

En ce qui concerne les opérations de traitement d'images de moyen niveau, différents algorithmes ont été simulés sur le modèle VHDL de SYMPATIX et donnent de bons résultats, présentés dans les 3 paragraphes suivants, selon que ces algorithmes travaillent sur des points, des contours ou des régions.

IV.1.1.a : Les algorithmes travaillant sur des points.

Ces algorithmes étant relativement simples et rarement utilisés, nous n'en présentons ici qu'un seul, isolant les points caractéristiques d'une image et les répartissant sur les processeurs adéquats. Ce type d'algorithme est rarement utilisé seul, mais il intervient fréquemment dans

des opérations plus complexes. Une configuration simple a été étudiée et est présentée ici. Ce premier algorithme permet d'introduire la mise en oeuvre de SYMPATIX.

Le but de cette opération est de grouper en listes, les points correspondant à un même modèle dans le même processeur. Ainsi, un processeur peut récupérer, à titre indicatif, les points nuls, un autre les points blancs, un autre les points d'angle, etc. Les essais ont été effectués sur un algorithme détectant des points particuliers dans une image binaire. Les modèles de points à détecter sont les suivants :

$$P1 \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad P2 \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix} \quad P3 \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix} \quad P4 \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Voyons les opérations nécessaires pour détecter l'un de ces modèles. Il faut outre le point courant, récupérer les 8 voisins afin de déterminer si ces 9 points sont organisés suivant le modèle considéré. Si c'est le cas, le point, à savoir ses coordonnées sont envoyées sur le réseau vers le processeur adéquat. Par exemple, les points correspondant aux modèles P1, P2, P3 et P4 peuvent être envoyés respectivement sur les processeurs élémentaires 0, 1, 2 et 3. Pour la détection d'un modèle, l'organisation de l'algorithme pour un point de l'image est donnée dans la figure IV.1.2.

Nous travaillons en mode SIMD, tous les processeurs effectuent les mêmes instructions, et si au moment de l'envoi vers le réseau un processeur n'a pas de points à envoyer, alors il est masqué. L'envoi des points correspond à l'envoi des coordonnées (x,y) des points détectés.

L'envoi des coordonnées x et y de chaque point de contour s'effectue en deux écritures, le champ des données du message à envoyer sur le réseau est ainsi constitué des coordonnées (x,y) du point détecté, le champ adresse contient le numéro du processeur destinataire. Le nombre d'instruction (NI) mises en oeuvre dans cet algorithme est de :

$$NI = Nnh + k \times Ndp + Np \times Nwr \qquad \text{(IV.1.1.1)}$$

où Nnh représente le nombre d'instructions nécessaires à la gestion du mode non hélicoïdal, Ndp le nombre d'instructions nécessaires à la détection d'un modèle, Np le nombre de modèles, Nwr le nombre d'instructions nécessaires à l'écriture d'un message dans le réseau et k un facteur de proportionnalité (nombre réel positif).

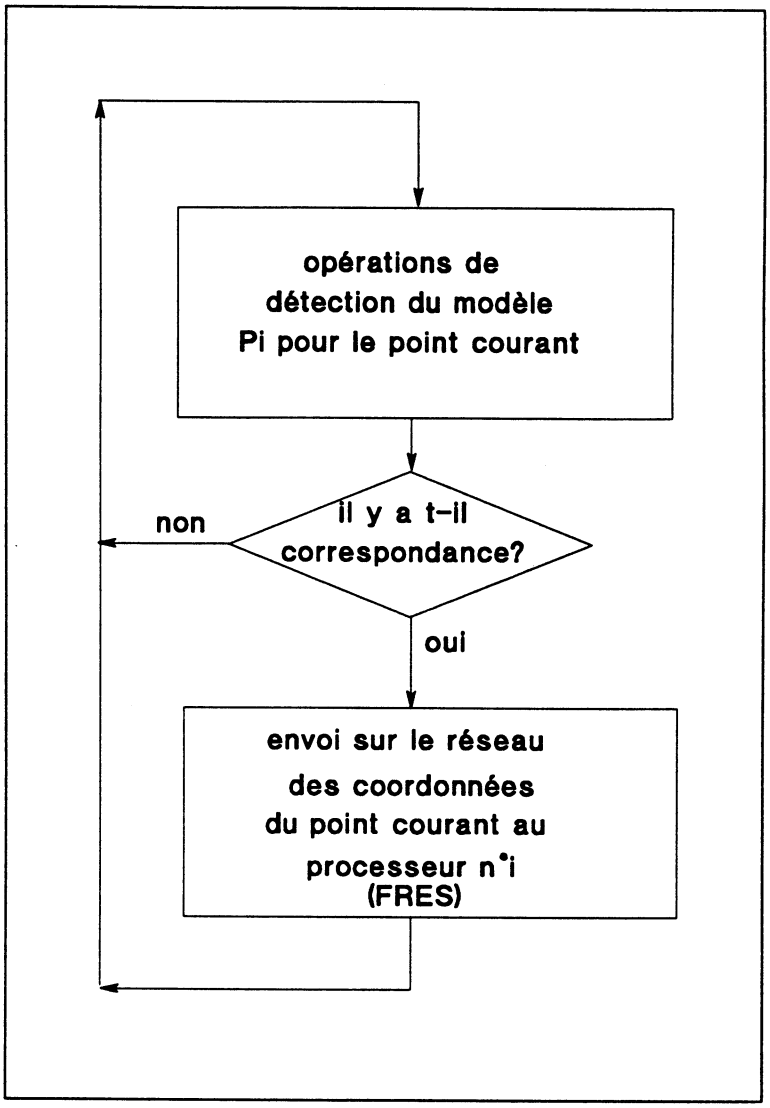

***Figure IV.1.2** : Organigramme de détection de points caractéristiques dans une image.*

Si le mode de calcul automatique dans la cellule de x et y n'est pas retenu, le coût de la gestion de ces deux valeurs est de 8 instructions 4LP (Nnh=8), pour le type de modèles utilisé Np vaut 21 et enfin pour composer le message à envoyer sur le réseau, il est nécessaire de positionner LIG (bits de poids forts du registre d'index) à une valeur particulière (afin de positionner les bits 15 et 8 du bus d'adresses à 1), puis COLO (bits de poids faibles du registre d'index) prend la valeur du numéro du processeur destinataire si bien que Nwr vaut 4.

```
Initialisation

LEXT;

        LINT;

        Gestion des coordonnées
        xy non hélicoïdales                          )Nnh

        Détection de l'un des modèles                ) Ndp

        Envoi sur réseau si correspondance:
        "LIG=cst /I0;
        COLO=numéro PE destinataire /I0;             Nwr
        FRES= x /I0;                                                  kxNdp
        FRES= y /I0;"                                                   +
                                                                      NpxNwr
        détection d'un autre modèle;

        envoi réseau si correspondance;

        etc...

LEND;

STOP;
```

Tableau IV.1.1 : Programme 4LP de détection de points caractéristiques.

IV.1.1.b : Les algorithmes travaillant sur des contours.

Deux algorithmes sont présentés ici, l'un en détail, il s'agit de la transformée de Hough, l'autre étant l'approximation polynomiale par les moindres carrés.

La transformée de Hough, décrite en annexe A.1.5 de cette thèse, est une opération gourmande en calculs, non pas au niveau précision mais au niveau de la quantité de points à traiter. Cette opération est fréquemment utilisée, si bien que les calculateurs dédiés à la vision présentent leurs performances pour cette opération, ce qui permet de les comparer entre eux. Le but de la transformation est, à partir de l'ensemble des points de contour d'une image, de générer une table d'accumulation (ρ,θ). Cette table est obtenue en effectuant l'opération suivante sur tous les points de contours :

$$\rho = x \times \cos(\theta) + y \times \sin(\theta) \qquad \text{(IV.1.1.2)}$$

avec θ décrivant l'intervalle 0 à 2π en prenant autant de valeurs qu'il y a de colonnes dans la table de Hough résultat. La parallélisation envisagée s'effectue au niveau de ces colonnes, un

processeur gère une (ou plusieurs) colonne, c'est-à-dire qu'il effectue l'opération (IV.1.1.2) pour un (ou plusieurs) θ sur tous les points de contour de l'image.

Ainsi, les points de contour doivent être distribués sur tous les processeurs, le réseau doit donc travailler en mode diffusion de données. Rappelons que le principe du réseau est de distribuer les messages sur les processeurs, alors que ces mêmes processeurs effectuent des opérations, de bas niveau généralement, sur d'autres points image. Ainsi, un algorithme de filtrage/seuillage effectue cette détection. Cet algorithme travaille au niveau des points de l'image, et envoie les coordonnées des points de contour sur le réseau (figure IV.1.3). Le réseau diffuse les coordonnées de ces points de contour (contenues dans des messages), pendant que les processeurs traitent d'autres points de l'image. Une fois tous les points de l'image traités, les processeurs effectuent la transformée de Hough sur les points de contour détectés lors du seuillage, et diffusés à tous les processeurs grâce au réseau.

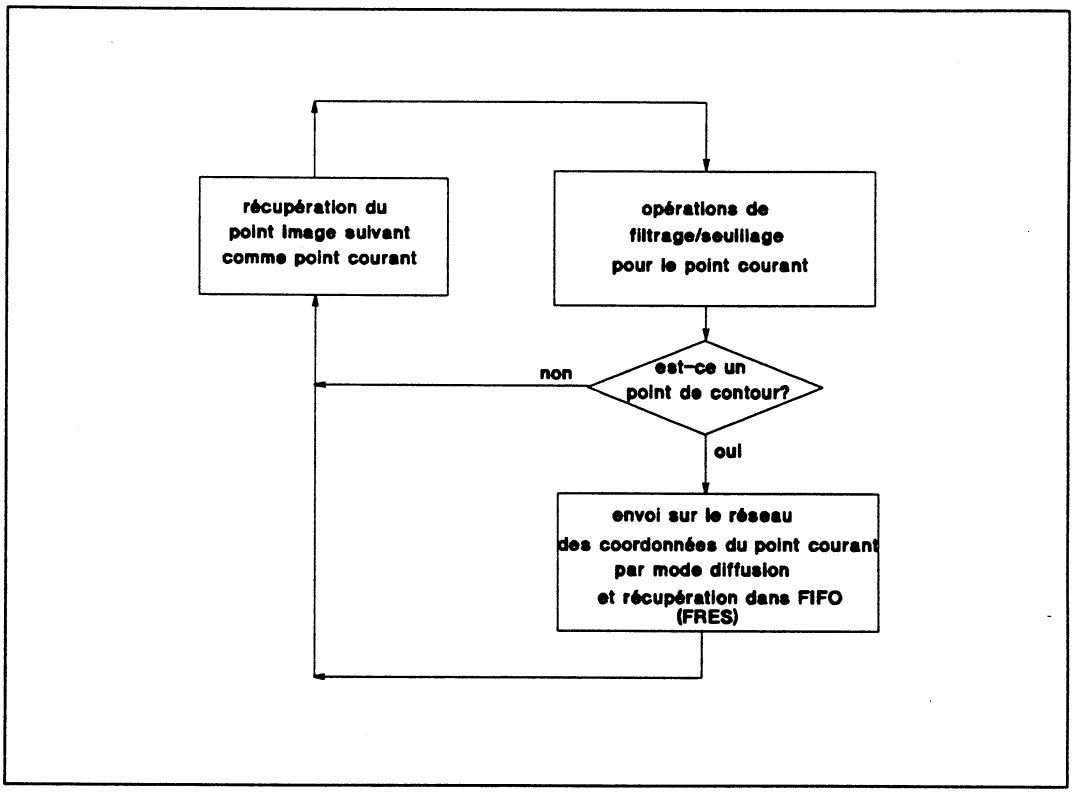

Figure IV.1.3 : Organigramme de l'opération de traitement d'images de bas niveau de filtrage/seuillage.

En ce qui concerne la configuration du réseau, la réception des données s'effectue sur des FIFOs, ces données sont les coordonnées des points de contour, et la transformée de Hough travaille sur ces points regroupés en liste dans les FIFOs. Dans notre simulation, le filtrage choisi est un filtre de Sobel qui est codé en 31 instructions 4LP. Le seuillage, réalisé

par rapport à une constante, s'effectue au travers de 4 instructions. Si la cellule ne dispose pas du module de calcul des valeurs x et y du point courant, 8 instructions sont alors nécessaires pour effectuer ce calcul en 4LP. Enfin, l'envoi réseau, lui, ne consomme que 2 instructions dans les 2 cas (avec ou sans le calcul de x y en interne de la cellule), LIG et COLO sont positionnés en début de programme une fois pour toutes. Entre les deux solutions, la différence est de 8 instructions, soit 20% de code en plus pour l'algorithme détaillé ici. Si l'image traitée est de 256×256 points et si la structure comprend 32 processeurs, alors le temps perdu à effectuer ces calculs de (x,y) est de 2 ms.

Précisons maintenant, qu'elle doit être la vitesse des décalages dans le réseau nécessaire, si l'opération de diffusion doit s'effectuer par décalages. Afin d'éviter tout blocage au niveau du réseau, il est nécessaire d'effectuer la distribution des messages entre 2 écritures réseau. Nous avons vu dans le chapitre précédent, paragraphe III.3.3, que la vitesse (Td) des décalages est données par l'équation III.3.3.1 :

$$Td <= \frac{Nmi}{M} \times Tcycle$$

Ici, Nmi, le nombre de mico-instructions entre chaque écriture réseau, est de 35 (31+4) au pire cas. Ce pire cas correspond au minimum d'instructions insérées entre 2 écritures réseau, à savoir quand le calcul de x,y s'effectue dans la cellule. Pour un cycle processeur (Tcycle), de 120 ns, nous obtenons les résultats donnés dans le tableau IV.1.2.

Nombre M de processeurs	Td max pour 1 réseau	Td max pour 2 réseaux
32	131 ns	262 ns
64	65 ns	131 ns
128	32 ns	65 ns
256	16 ns	32 ns
512	8 ns	16 ns
1024	4 ns	8 ns

***TableauIV.1.2** : Vitesse de décalage requises pour l'opération de filtrage/seuillage et diffusion par décalages (FRES).*

Pour cette opération, une FIFO en émission, insérée dans chacune des cellules, peut être utilisée car il n'est pas nécessaire de récupérer les messages dans l'ordre de leur émission. Le

rôle de ces FIFOs est alors de charger le réseau au maximum, si bien que la limite des vitesses de décalage peut être repoussée au prix de cette électronique supplémentaire. Pour cette application, cette solution est à envisager pour des structures à plus de 128 processeurs élémentaires et un seul réseau intelligent, car en deçà, les vitesses requises sont a priori réalisables.

Si les vitesses de décalage présentées ci-dessus sont réalisées, ou si une FIFO d'émission est mise en oeuvre (dans ce cas, le nombre de points de contour de l'image doit être très faible par rapport au nombre de points de l'image), le temps requis pour effectuer l'opération de filtrage/seuillage et envoi au réseau est le suivant :

Nombre M de processeurs	image de taille 128×128	Image de taille 256×256
32	2.5 ms	10 ms
64	1.3 ms	5 ms
128	0.7 ms	2.5 ms
256	0.7 ms	1.3 ms

***TableauIV.1.3** : Temps d'exécution de l'opération de filtrage/seuillage et diffusion des points de contours sur les processeurs.*

Une fois cette opération de filtrage/seuillage et envoi au réseau terminée, les points de contour se retrouvent dans les différentes FIFOs sous forme de listes, et les processeurs viennent récupérer ces points afin d'effectuer leur transformée de Hough partielle. Chaque processeur construit une ou plusieurs colonnes de la table de Hough. Il travaille sur un ou plusieurs θ, et à ce niveau, il n'y a aucun échange entre les processeurs.

Le temps d'exécution de cette opération dépend des 4 paramètres suivants :

- la taille des mémoires tampon (FIFOs). Quand les FIFOs des cellules se trouvent pleines, il est nécessaire de les décharger grâce à un programme d'interruption, qui est la transformée de Hough elle même. Plus les FIFOs sont petites, plus souvent il est nécessaire de faire appel à ce programme d'interruption. Avant d'exécuter la transformée de Hough, il est nécessaire de sauvegarder le contexte du programme de seuillage/filtrage et de le restituer en fin d'opération (figure IV.1.4);

- du nombre de points de contour détectés, donc du nombre de points à traiter;
- du nombre de directions de la Transformée de Hough;
- et bien entendu du nombre de processeurs.

Posons Np le nombre de points de contour et Tf la taille des FIFOs, on a alors :

$$Np = K \times Tf + F \qquad \text{(IV.1.1.3)}$$

avec F < Tf.

Le temps d'exécution de la transformée de Hough (Tth) suivant M directions, M étant le nombre de processeurs de la structure, est alors égale à :

$$Tth = (Tsc+Tith+Trc)\times K + Ttth\times K\times Tf + Tith + Ttth\times F), \quad \text{(IV.1.1.4)}$$

avec Tsc le temps nécessaire pour sauvegarder le contexte, Trc le temps nécessaire pour restituer le contexte, Tith le temps d'initialisation de la transformée de Hough et Ttth le temps de traitement d'un point de contour.

Pour notre structure quelle que soit l'application, Tsc et Trc restent les mêmes et consomment respectivement 28 et 34 instructions 4LP soit 3.34 µs et 4.1 µs. Par contre Tith et Ttth sont propres à notre application, et Tith représente 13 instructions 4LP soit 1.6 µs et Ttth 21 instructions 4LP soit 2.5 µs. Ainsi le temps de traitement de la transformée de Hough s'exprime également de la façon suivante :

$$Tth = [(75 + 21\times Tf)\times K + 21\times F + 13]\times 120\times 10^{-9}\ s. \quad \text{(IV.1.1.5)}$$

Si le nombre de directions (Nd) de la transformée de Hough est supérieur au nombre de processeurs, alors Tth devient :

$Tth_{Nd>M} = Tth_{Nd=M}\times([Nd(div.ent)M]+1)$, (I.V.1.16) où (div.ent) représente la division entière.

Le tableau IV.1.4 présente les résultats obtenus pour des simulations de la transformée de hough suivant 256 directions, effectués avec des tailles de FIFOs de 256 octets, pour des images de 128×128 et de 256×256 points ayant 5%, 10% ou 20% de points de contour.

taille	taux	32 PE	64 PE	128 PE	256 PE
128×128	05%	17ms	8ms	4ms	4ms
	10%	33ms	17ms	8ms	8ms
	20%	67ms	33ms	17ms	17ms
256×256	05%	67ms	33ms	17ms	8ms
	10%	134ms	67ms	33ms	17ms
	20%	264ms	134ms	67ms	33ms

***TableauIV.1.4** : Temps d'exécution de l'opération de transformée de Hough suivant 256 directions, sur des images de tailles différentes ayant 5, 10 ou 20 % de points de contour.*

On peut reprocher à ces résultats d'être tributaires de l'image par le nombre de points de contours détectés. En effet, le temps maximum est celui correspondant à une image blanche où la totalité des points (100%) sont à traiter. Mais il est important de noter, pour éviter ce problème, qu'il est préférable d'effectuer un seuillage adaptatif ne retenant que x% de points de contour. Notons que ce type de seuillage risque d'apporter des contraintes plus strictes en ce qui concerne la vitesse de décalage du réseau. En effet, il ne serait pas possible d'effectuer un algorithme de filtrage/seuillage, mais un algorithme de filtrage puis un autre de seuillage adaptatif. Il serait alors préférable, pour les structures comportant de nombreux processeurs d'effectuer la diffusion de données par propagation électrique afin d'éviter des blocages du réseau. Dans ce cas, le message est transmis simultanément à tous les processeurs.

Ainsi, cette solution de propagation par diffusion électrique (BRES) est a envisager, si le nombre de processeurs de la structure est important, et si le nombre de processeurs ayant une donnée à émettre est inférieur à la moitié du nombre total de processeurs. En effet, cette seconde hypothèse induit que le temps de distribution des points devient plus rapide, par rapport à la solution à décalages successifs. Or si l'on considère que l'ensemble des points de contour est réparti uniformément sur l'image, on trouve les résultats du tableau IV.1.5 en ce qui concerne le temps de distribution des messages après une opération d'écriture dans le réseau (avec un seul réseau).

Une telle solution, permet de n'avoir que quelques instructions entre chaque écriture réseau, à la condition que les points de contour soient répartis uniformément sur l'image. Pour s'affranchir de cette contrainte, une FIFO à l'émission permet de repousser également ces limites.

% de points de contour	32 PEs	64 PEs	128 PEs	256 PEs
5%	FRES: 32 Td BRES: 1.6 Td	FRES: 64 Td BRES: 3.2 Td	FRES: 128 Td BRES: 6.4 Td	FRES: 256 Td BRES: 12.8 Td
10%	FRES: 32Td BRES: 3.2 Td	FRES: 64 Td BRES: 6.4 Td	FRES: 128 Td BRES: 12.8 Td	FRES: 256 Td BRES: 25.6 Td
20%	FRES: 32Td BRES: 6.4 Td	FRES: 64 Td BRES: 12.8 Td	FRES: 128 Td BRES: 25.6 Td	FRES: 256 Td BRES: 51.2 Td

***TableauIV.1.5** : Temps nécessaire pour effectuer la distribution des données en mode diffusion. FRES correspond au mode de diffusion par décalage, BRES par propagation électrique. Td représente la période de l'horloge de décalages.*

Pour l'opération d'approximation polynomiale par les moindres carrés, il est également indispensable de récupérer les coordonnées des points courants, c'est sur ces coordonnées que l'approximation est faite. Le réseau est ici utilisé en mode de transfert point à point afin de distribuer ces coordonnées sur les processeurs, cette distribution s'effectue, par exemple, par paquets de 10 points de contour connexes, sur un même processeur. La récupération des données se fait sur les FIFO des cellules. On retrouve alors le même principe d'algorithme que celui utilisé pour détecter les points caractéristiques d'une image. Une fois cette distribution effectuée, les processeurs effectuent l'approximation avec les 10 points stockés dans leur FIFO respective, ensuite les processeurs communiquent entre eux de manière synchrone pour comparer leurs résultats afin de regrouper les paquets de points, ou afin d'établir des points de cassure.

De manière générale, les opérations de cette classe, travaillant sur les contours, manipulent les coordonnées (x,y) des points de contour, si bien qu'il est intéressant que les cellules puissent calculer automatiquement les coordonnées des points courants. Le mode de distribution point à point est fréquemment utilisé, dès que le parallélisme est obtenu au niveau des contours, par exemple pour le codage de Freeman et les "Psi-curves"..

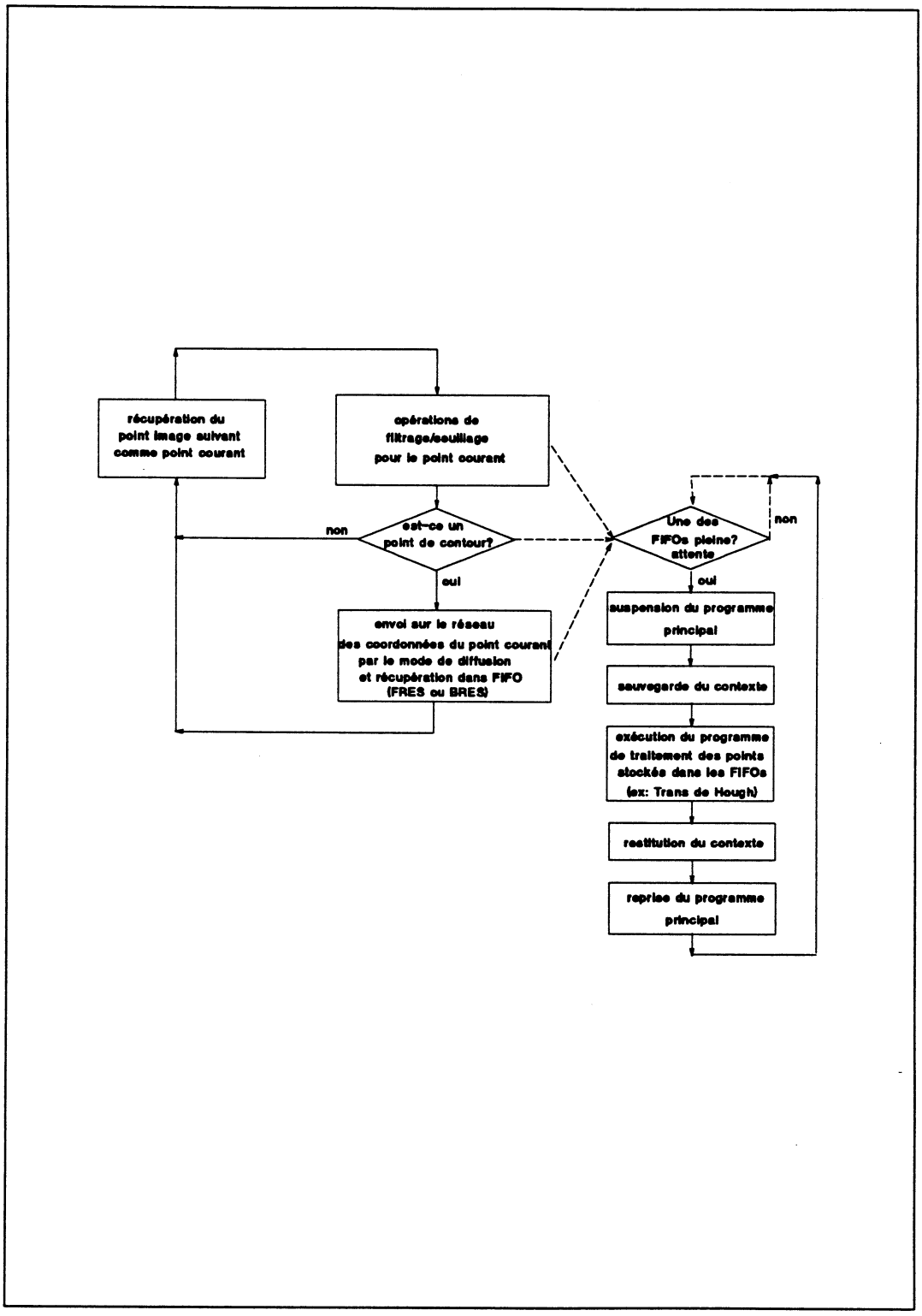

***Figure IV.1.4** : Organisation du couplage des opérations de filtrage/seuillage et de la transformée de Hough (programme de traitement des points stockés dans les FIFOs); quand une des FIFOs de réception de données se trouve pleine, une interruption est générée afin d'exécuter le programme destiné à vider les FIFOs.*

IV.1.1.c : Les algorithmes travaillant sur des régions.

Voyons dans cette section l'opération, appelée l'étiquetage de régions, qui consiste à identifier sur une image les régions connexes. Dans l' annexe A.1.6 de cette thèse, nous proposons d'effectuer cette opération en travaillant non pas sur les points de l'image, mais sur des ensembles de points connexes appelés "run length codes" ou segments. Une première opération est d'effectuer la détection de ces objets dans l'image de départ. Cette détection s'effectue par colonnes et les segments détectés sont distribués sur les processeurs de la manière suivante : les premiers segments de chacune des colonnes sont envoyés au premier processeur de la chaîne, les seconds au second processeur, etc. Le réseau intelligent ouvert intervient ici pour effectuer cette distribution sur les processeurs en mode point à point avec réception en FIFO (FRES). Les segments sont composés des coordonnées du point final (ou de départ) du segment, ainsi que de sa longueur si bien que, l'ensemble formant un mot codé sur plus de 16 bits, un segment est transmis par l'envoi de 2 messages sur le réseau. Il est alors obligatoire de reconstruire les messages à l'arrivée, l'utilisation d'une FIFO à l'émission est alors à proscrire afin de conserver la cohérence spatio-temporelle des différents messages.

Une fois cette opération de détection et de distribution effectuée, l'opération de recherche de connexités commence. Aussi, les processeurs élémentaires de SYMPATIX travaillent en mode associatif. Un segment (le segment courant) est présenté, par le processeur possédant ce segment, à tous les processeurs lesquels déterminent, en parallèle, s'ils possèdent un segment connexe au segment présenté. Le réseau intelligent ouvert est utilisé pour présenter le segment courant à tous les processeurs, le mode de diffusion par propagation électrique est alors mis en oeuvre (BRES), ensuite les processeurs répondent grâce au mode point à point (FRES). Le temps d'exécution de cet algorithme est proportionnel au nombre de segments à étiqueter.

L'algorithme de détection et d'envoi des segments connexes consomme 29 instructions, dont 8 sont utilisées pour calculer x et y et 4 pour construire les messages et adresser le réseau. Un segment est constitué de 24 bits, il est donc codé sur deux messages. Deux écritures réseau sont alors nécessaires, mais elles doivent être séparées par suffisamment d'instructions afin de ne pas bloquer le réseau. Il est important de noter, à cet égard, que si un processeur a effectué un envoi réseau à la ligne numéro l, il n'en effectuera pas à la ligne suivante (l+1) si bien que l'envoi de la seconde partie du message peut être effectué lors du traitement de la ligne suivante.

Une fois toutes les données en place, la recherche de connexités peut s'effectuer. Dans un premier temps, les processeurs dépilent toutes les valeurs des FIFOs et les rangent dans

l'ordre croissant des colonnes, il n'y a qu'un seul segment par colonne et par processeur, cette partie correspond au programme d'interruption à exécuter si une des FIFOs se trouve pleine. Ensuite, les segments sont pris un à un et leur connexité est déterminée en comparant les coordonnées du segment courant aux coordonnées des segments des colonnes voisines, distantes de moins un et plus un. Cette opération se décompose ainsi : envoi par le processeur contenant le segment courant de ses caractéristiques (P1), comparaison avec les segments de la colonne précédente et la colonne suivante (P2) et envoi des résultats, mise à jour des étiquettes (P3) et enfin détermination du nouveau segment courant (P4) (figure IV.1.5).

Le temps d'exécution de l'algorithme complet (Tt), de la détection des segments à la mise à la mise à jour des étiquettes, s'exprime de la façon suivante :

$$Tt = Tdrlc + Nrlc \times (Ttrans + Tpa) \qquad \text{(I.V.1.1.7) où}$$

$$Tpa = Tp1 + Tp2 + Tp3 + Tp4. \qquad \text{(I.V.1.1.8)}$$

Tdrlc est le temps nécessaire à la détection des "run lenght codes", ce temps est uniquement dépendant du nombre de processeurs et de la taille des images. Si la vitesse du réseau est suffisamment grande, Nrlc représente le nombre des segments détectés, Ttrans correspond au temps de transfert et de rangement des segments des FIFOs vers la mémoire et les Tpi représentent les temps nécessaires à l'exécution des différentes phases Pi présentées ci-dessus. Ttrans, Tp1, Tp2, Tp3 et Tp4 sont les temps d'exécution de respectivement 9, 3, 32, 2 et 10 micro-instructions 4LP. Pour une image de taille 256×256 points contenant 5000 segments, soit en moyenne 20 par colonne, ce qui peut être considéré comme une situation limite, le temps Tt vaut 41 ms pour une solution à 32 processeurs élémentaires. Pour des nombres de segments plus réalistes, toujours avec le même nombre de processeurs, les temps d'exécution de l'algorithme complet sont donnés dans le tableau IV.1.6.

Nombre de segments	1 segment par colonne (256)	2 segments par colonne (512)	5 segments par colonne (1280)	10 segments par colonne (2560)	15 segments par colonne (3840)
Temps total (Tt)	9 ms	11 ms	16 ms	24 ms	33 ms
% pour segments (1)	19 %	32 %	55 %	70 %	78 %

(1) : correspond à la proportion du temps de traitement des segments par rapport au temps de traitement total. Le temps de détection des segments est constant pour une structure donnée et une taille d'image donnée et vaut 7.1 ms (32 PEs, 256×256 points).

***TableauIV.1.6** : Temps nécessaire à l'étiquetage en régions d'une image.*

Les résultats pour cette opération d'étiquetage, sont améliorables suite aux remarques énoncées dans l'annexe A.1.6, et présentent l'avantage de n'être tributaires que du nombre de segments dans l'image et non de l'organisation de ceux-ci. Si le barreau n'est pas scindé en différentes parties suite aux remarques présentées dans l'annexe A.1.6, alors seul le temps de détection des segments est linéairement proportionnel avec le nombre de processeurs. A titre indicatif, l'image de la figure A.1.3, donnée en annexe, est étiquetée en 15 ms.

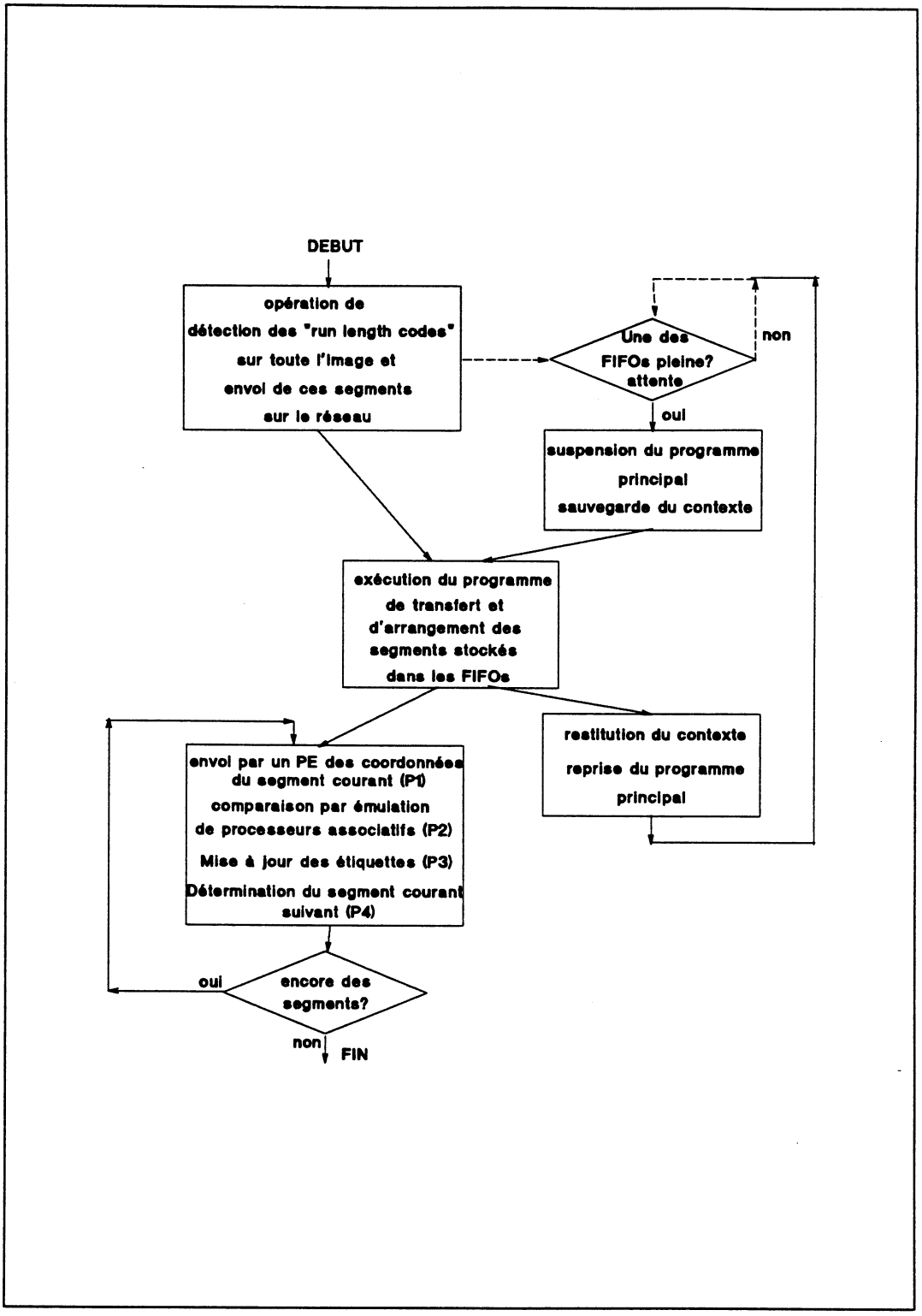

***Figure IV.15** : Organigramme le l'algorithme d'étiquetage en régions. Pour cette opération, les processeurs de SYMPATIX émulent une structure multiprocesseur associative.*

L'algorithme de diffusion/fusion peut également tirer avantageusement parti des possibilités du réseau intelligent ouvert, tant au niveau de l'organisation des données dans les processeurs que de la précision des opérations à effectuer. Pour la réorganisation, les processeurs peuvent exécuter un programme redistribuant les points de l'image sous forme de bandes sur les processeurs à partir d'une distribution hélicoïdale. Pour cette opération, le mode de stockage par DMA est utilisé, mais c'est aux processeurs d'effectuer l'opération de calcul du processeur destinataire et de l'adresse de stockage. Pour chaque point de l'image, le nombre d'instructions nécessaires à la réorganisation des points est de :

$$NI = Ncf + Nwr \qquad \text{(IV.1.1.9)}$$

où Ncf représente le nombre d'instructions utilisées pour calculer les coordonnées du processeur destinataire ainsi que de l'adresse DMA, et Nwr représente le nombre d'instructions pour envoyer un message dans le réseau, ici Nwr vaut 5. Cette formule est valable pour tous changements de format, et pour celui qui nous concerne, Ncf vaut 3. En effet, ces échanges étant statiques, il n'est en fait pas nécessaire, aux processeurs, de calculer le numéro du processeur destinataire, ni l'adresse de stockage DMA, ils sont stockés dans des images de "constantes" et donc directement récupérées et utilisées par les processeurs. Ainsi NI représente 8 instructions, et ce nombre correspond à de nombreux changements de formats. Pour une image de 256×256 points, cette opération est effectuée en 2 ms sur une structure à 32 processeurs. Toutefois, suivant la vitesse de décalage retenue, et suivant le nombre de processeurs, il devient nécessaire d'introduire, à partir d'une certaine limite donnée par l'équation III.3.3.1, des instructions supplémentaires entre chaque écriture réseau, instructions pouvant, par exemple, correspondre à un filtrage.

En ce qui concerne la précision des calculs, SYMPATIX peut recevoir un module de calcul spécialisé dans la chaîne formée par le réseau. Ce module peut effectivement effectuer des calculs de grande précision sur des données envoyées par les processeurs. Ce processeur spécialisé doit être interfacé avec le réseau et doit reconstituer les messages à l'arrivée, d'où l'importance de conserver la cohérence entre ces messages (figure IV.1.6). Si les processeurs décident de solliciter ce coprocesseur pour effectuer une addition en flottant de deux nombres codés sur 32 bits, 4 messages doivent être envoyés sur le réseau en mode point à point à une adresse particulière, celle du processeur spécialisé, ensuite les processeurs ne peuvent disposer des résultats aussitôt ces écritures effectuées, ils doivent attendre que le calculateur spécialisé leur renvoie le résultat. Par contre lors de cette attente, ils peuvent travailler sur d'autres données. Notons qu'exceptionnellement, les messages envoyés aux processeurs doivent être accompagnés du numéro de ce processeur afin que le module spécialisé sache à

qui il doit restituer le résultat. Pour effectuer cette restitution, le processeur spécialisé attend que le réseau soit vide, ensuite il injecte les résultats dans le réseau. Cette solution demande du matériel particulier à développer.

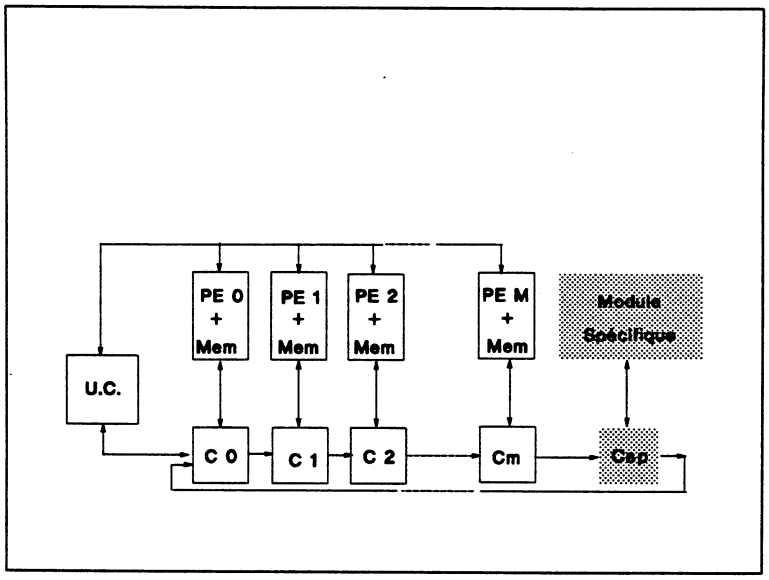

***Figure IV.1.6** : Exemple d'utilisation d'un processeur spécialisé dans SYMPATIX. Une cellule supplémentaire (CSp) est utilisée afin d'interfacer le processeur spécialisé (module spécifique) au réseau intelligent ouvert.*

IV.1.2 : Les opérations graphiques supportées par SYMPATIX.

Il s'avère que, outre les opérations de traitement d'images de bas et moyen niveau, SYMPATIX est bien adapté pour d'autres opérations de vision. Soit ces opérations travaillent sur des images uniquement et exigent des redistributions aléatoires de points sur les processeurs, c'est le cas pour les opérations de manipulation d'image, soit ces opérations transforment des listes de données en images, ce qui est l'inverse des opérations de traitement d'images de moyen niveau où des images sont transformées en listes de points (figure IV.1.7). Commençons par ces opérations de génération graphique.

IV.1.2.a : SYMPATIX et la génération graphique.

Des travaux de recherche effectués dans notre laboratoire sur le domaine de la parallélisation des algorithmes de génération graphique, ont permis d'évaluer SYMPATIX, par rapport à SYMPATI2 pour ces opérations. De premiers résultats issus de [LET91] donnent les performances de la structure pour l'opération de tracé de segment. Pour des structures à 32 processeurs élémentaires, SYMPATIX s'avère 6 fois plus rapide que SYMPATI2 grâce au réseau intelligent ouvert. Ainsi la structure trace 1.2 million de vecteurs

de 10 points et de direction quelconque par seconde. Pour cette opération, c'est le mode DMA (instruction DRES) qui est utilisé. Pour ce mode, le calcul, interne à la cellule, du numéro du processeur destinataire, et de l'adresse de stockage interne, à partir des coordonnées du point cible est utilisé. Le programme calcule les coordonnées (x',y') du point à atteindre et ensuite sollicite le réseau pour effectuer le stockage de la donnée (niveau de gris) dans le processeur adéquat. Le réseau est actuellement en cours d'évaluation pour les autres opérations de la génération graphique, à savoir le remplissage de figures etc...

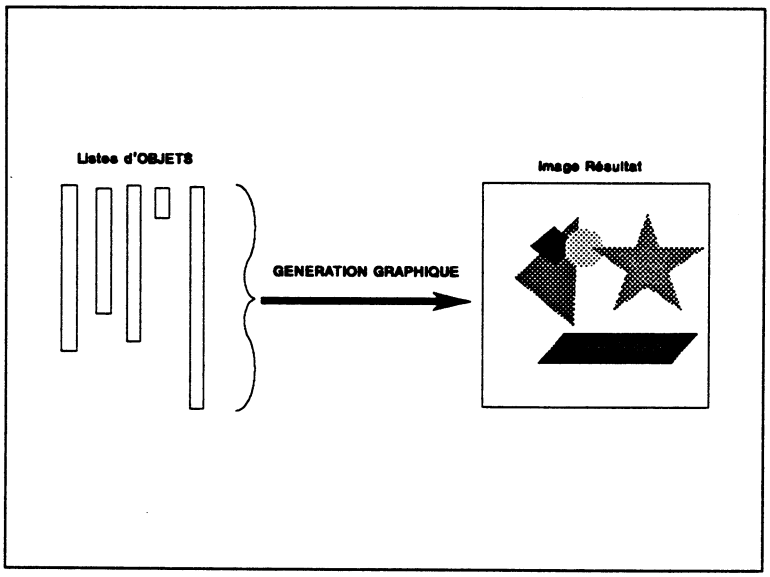

***Figure IV.1.7** : Principe de la génération graphique. Des images sont fabriquées à partir de listes de données.*

IV.1.2.b : SYMPATIX et la manipulation d'images.

Le but du travail de cette thèse était d'étendre les capacités de SYMPATI2 au traitement d'images de moyen niveau, non à ce type d'opérations de manipulation d'images. Toutefois, il nous a semblé intéressant d'évaluer la structure pour ce type d'opérations gourmandes en calculs et réclamant des échanges aléatoires entre processeurs.

Le principe de ce type d'opérations est le suivant. Il s'agit à partir de coordonnées d'un point de l'image (x,y), de déterminer les coordonnées du point image (x',y') et de transférer la valeur en niveau de gris du point (x,y) au point (x',y'). La correspondance entre les 2 couples de coordonnées s'obtient par l'équation suivante :

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} \qquad \text{(IV.1.2.1)}$$

Ce sont les coefficients de la matrice qui fixent le type de déformation. Pour une opération de rotation d'image, ces coefficients prennent les valeurs suivantes, A=cosθ, B=-sinθ, C=sinθ et D=cosθ. Afin d'éviter tous les problèmes d'artefacts, ce n'est pas cette transformée directe qui est utilisée, mais la transformée inverse. A partir des coordonnées des points du plan résultat, l'ensemble des antécédents est calculé, le niveau de gris retenu est alors une moyenne pondérée des niveaux de gris des points de l'ensemble des antécédents. Malheureusement, le réseau intelligent ouvert confère à SYMPATIX des possibilités d'écriture aléatoire à distance (RAW : Random Acces Write), non des lectures aléatoires à distance si bien qu'il est plus aisé d'exécuter l'opération IV.1.2.1 dans le sens direct que dans le sens inverse. Nous présentons en premier, les résultats obtenus pour effectuer cette opération en sens direct, ensuite nous en verrons les limites.

L'opération de transformation choisie est la rotation, elle permet de visualiser les performances de la structure pour une opération, où des échanges aléatoires interviennent sans déformation du format d'arrivée par rapport au format d'entrée. Pour tous les points de l'image, les processeurs effectuent la transformation du couple (x,y) en un nouveau couple de coordonnées (x',y'), puis associent à ce nouveau point le niveau de gris (Ng) du point (x,y). Le réseau est utilisé pour aller stocker Ng dans le point image (x',y'), à savoir dans la mémoire du processeur contenant ce point. Le mode DMA est utilisé pour effectuer cette opération (DRES), l'option permettant de calculer le numéro du processeur destinataire ainsi que l'adresse de stockage DMA est retenue. L'algorithme (A1) de transformation directe du couple (x,y) en (x',y') est composé des 3 phases suivantes :

- calcul des coordonnées du point courant (x,y), il en coûte 8 instructions si ce calcul est effectué en interne, par contre si la cellule est dotée de cette capacité de calculer automatiquement ces coordonnées, 6 instructions sont tout de même nécessaires pour récupérer x et y à l'intérieur du processeur;

- à partir de (x,y), x' et y' sont calculés. 8 instructions codent cette opération qui est effectuée en précision entière;

- le niveau de gris du point (x,y) est envoyé, grâce au réseau, à l'adresse (x',y'). Si ce niveau de gris est codé sur 256 niveaux, une seule écriture réseau est nécessaire.

Le total d'instructions 4LP est de 17, ce qui donne les temps d'exécution listés dans le tableau IV.1.7, obtenus sur une image de 256×256 points :

Nombre processeurs	32	64	128	256
Temps d'exécution de A1	4.2 ms	2.1 ms	1.1 ms	0.5 ms
Vitesse de décalage minimale du réseau	64 ns	32 ns	16 ns	8 ns

Tableau IV.1.7 : Temps d'exécution de l'opération de rotation sur SYMPATIX en considérant les vitesses de décalage comme réalisées. Ces vitesses sont les vitesses limites de saturation du réseau.

Pour cette opération, à chaque boucle interne tous les processeurs effectuent un envoi réseau si bien que le réseau est fortement chargé. Notons que dans la pratique, ces contraintes sur la vitesse des décalages sont repoussables, car généralement avant d'effectuer des décalages, des opérations de bas niveau sont effectués sur les images, si bien que des instructions supplémentaires sont intégrables entres deux écritures réseau. De plus l'utilisation d'un double réseau, double les vitesses limite annoncées dans le tableau IV.1.7.

Qu'en est-il de la qualité des images obtenues? Elle n'est pas très bonne comme vous pouvez en juger sur la figure IV.1.9 où SYMPATIX à fait subir à l'image de "Lenna" une rotation de 30° (algorithme A1), il y a de nombreux trous dans l'image. Pour les boucher, nous avons utilisé un filtre, qui remplace tous les trous par la moyenne en niveau de gris des 8 voisins ce qui donne des résultats corrects. (figure IV.1.10).

Ainsi cette transformation directe est utilisable, pour les opérations générant peu de trous dus à l'arrondi effectué sur les calculs, c'est-à-dire pour les opérations transformant une image en une autre de même taille, ou mieux de taille plus petite. Toutefois, il est envisageable d'effectuer des transformées inverse dans le but de déterminer les coordonnées des point d'origine. Cet ensemble de coordonnées est facile à obtenir sur SYMPATI (et obtenu rapidement), par contre à la suite de cette opération, il est nécessaire d'effectuer des opérations de lectures aléatoires à distance (RAR : Random access Read), opération délicate pour SYMPATIX car le réseau ne supporte pas de tels échanges. En effet, une telle possibilité ne peut être obtenue sur une structure SIMD car les opérations de RAR nécessitent une dé-synchronisation des processeurs entres eux. Ceci se comprend, si par exemple, tous les processeurs effectuent une lecture à distance, en même temps, de données contenues dans le même processeur. Ainsi, cette configuration serait réalisable si les lectures s'effectuaient en mode DMA sans l'intervention du processeur lu. Cette réalisation pourrait s'envisager s'il s'avère nécessaire de développer un calculateur dédié à ce type d'opération.

***Figure IV.1.8** : Image de départ.*

***Figure IV.1.9** : Image résultat après rotation. Ce type de transformation directe crée des trous (points non affectés) dans l'image résultat..*

***Figure IV.1.10** : image finale après rotation et filtrage. Le filtrage à pour rôle de "boucher" les trous créés lors de la transformation directe.*

Ce qui a été présenté dans le paragraphe IV.1.1.c en ce qui concerne l'utilisation d'un processeur spécialisé (flottant) inséré dans la chaîne, peut être repris afin d'effectuer automatiquement et précisément toute opération de transformée géométrique (IV.1.2.1). Dans ce cas, les processeurs soumettent à la cellule dédiée le calcul de cette transformation, puis récupère le résultat précis sur (x',y').

IV.1.3 : Conclusion.

Ce paragraphe présente les résultats obtenus pour quelques opérations, de traitement d'images et graphiques. Ces résultats permettent d'apprécier concrètement les possibilités du réseau, ses contraintes ainsi que ses limites. On y voit ainsi comment mettre en ouvre des opérations d'envoi de messages sur le réseau en 4LP, comment récupérer des données arrivant sur un processeur, comment envoyer des coordonnées de points (x,y), ou des informations plus longues contenues sur plus d'un message, comment intégrer un module de traitement spécialisé dans la chaîne formée par les différentes cellules du réseau, et quel mode du réseau utiliser, suivant, le nombre de processeurs de la structure, la vitesse des décalages et la fréquence des envois dans le réseau.

Il est important de noter que dès que la vitesse des décalages des messages dans le réseau n'est pas assez importante, il y a blocage des processeurs et perte de la linéarité des performances. Dans ce cas, il peut s'avérer intéressant d'utiliser une FIFOs à l'émission pour certaines applications afin de diminuer le temps d'exécution de certains algorithmes (figure IV.1.11). Mais cette solution détruit la cohérence spatio-temporelle des messages qui est nécessaire pour certains algorithmes. La limite à partir de laquelle le réseau sature, est déterminée par le nombre d'instructions entre 2 écritures réseau (Nmi), le nombre de processeurs (M), la vitesse des décalages sur le réseau (Td) et le temps de cycle des processeurs (Tc). En mode point à point ou diffusion par décalages, d'après l'équation III.3.3.1, Il y a risque de saturation du réseau quand on a :

$$\frac{Nmi \times Tc}{M \times Td} > 1$$

Si cette règle n'est pas respectée, il y a un blocage certain pour les opérations de diffusion de données, et pour les opérations de transfert point à point, il y a blocage si M décalages sont nécessaires, sinon le point limite est repoussé et dépend du nombre de décalages maximum effectués.

Les FIFOs émettrices présentent effectivement des avantages pour les algorithmes ne nécessitant pas de cohérence entre les messages, pour lesquels, en moyenne, peu de processeurs effectuent l'opération d'écriture de messages. Cette possibilité s'obtient au prix d'une électronique supplémentaire conséquente. Notamment pour les solutions à beaucoup de processeurs (M>128), cette FIFO peut devenir une nécessité compte tenu des vitesses de décalages (Td) exigées.

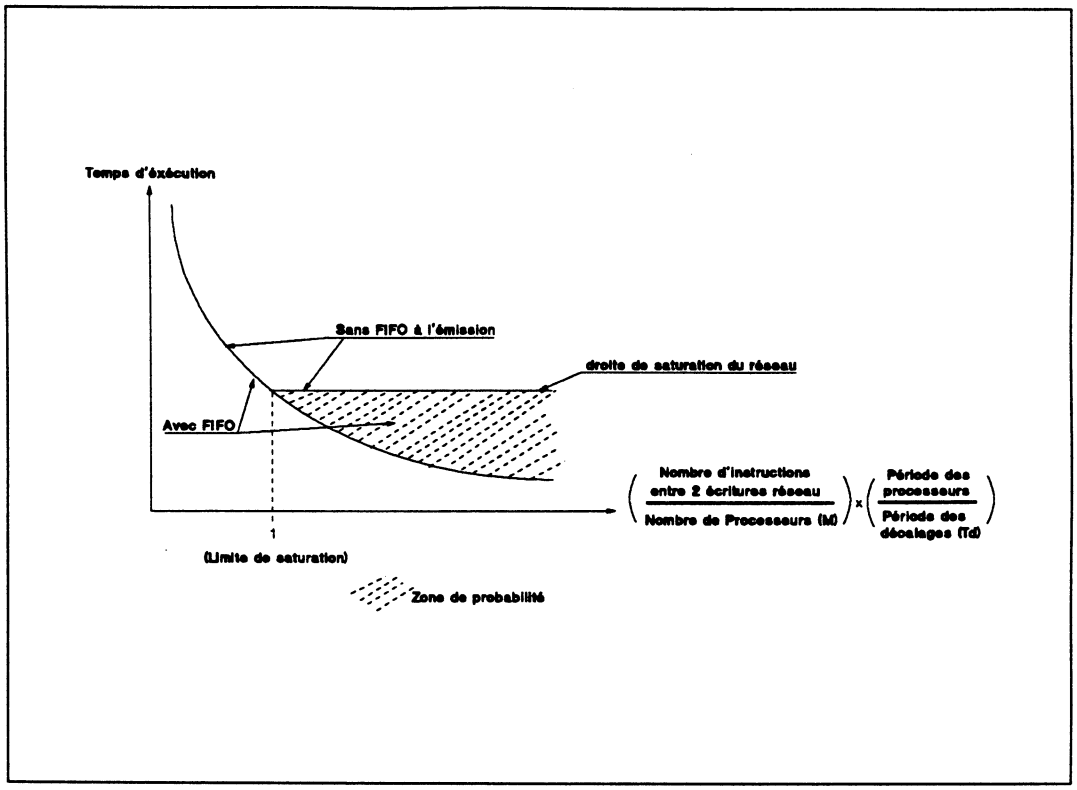

***Figure IV.1.11** : Comparaison des temps d'exécution d'algorithmes avec et sans FIFO à l'émission. Pour cet exemple, on s'est placé dans un cas limite, où pour chaque envoi de message dans le réseau, il y a M décalages à effectuer.*

Nous pouvons, pour l'opération de la transformée de Hough, comparer nos résultats avec ceux obtenus sur d'autres calculateurs. Par rapport au calculateur NCube [RAN90], nous sommes environ 40 fois plus rapide pour une même configuration (64 processeurs, image 256×256 et 10% de points de contours). Le calculateur CAAPP [WEE89], matrice de 512×512 processeurs, effectue cette transformation en 27 ms quel que soit le nombre de points de contour, SYMPATIX atteint cette performance doté de 256 processeurs sur des images comportant 5% de points de contour. Enfin, par rapport à une solution câblée à base d'un L64250 [LSI92], nous sommes déjà 13 fois plus rapides avec un SYMPATIX doté de 32 processeurs sur des images comprenant 20% de points de contour.

Il est maintenant nécessaire d'effectuer de la recherche algorithmique par rapport à cette structure, afin de présenter d'autres performances de SYMPATIX. La figure IV.1.12 donne le domaine dans lequel ces développements alrorithmiques peuvent s'effectuer. Ces évaluations doivent-elles s'effectuer sur le modèle VHDL ou un autre modèle? Nous répondons à cette question dans le paragraphe suivant.

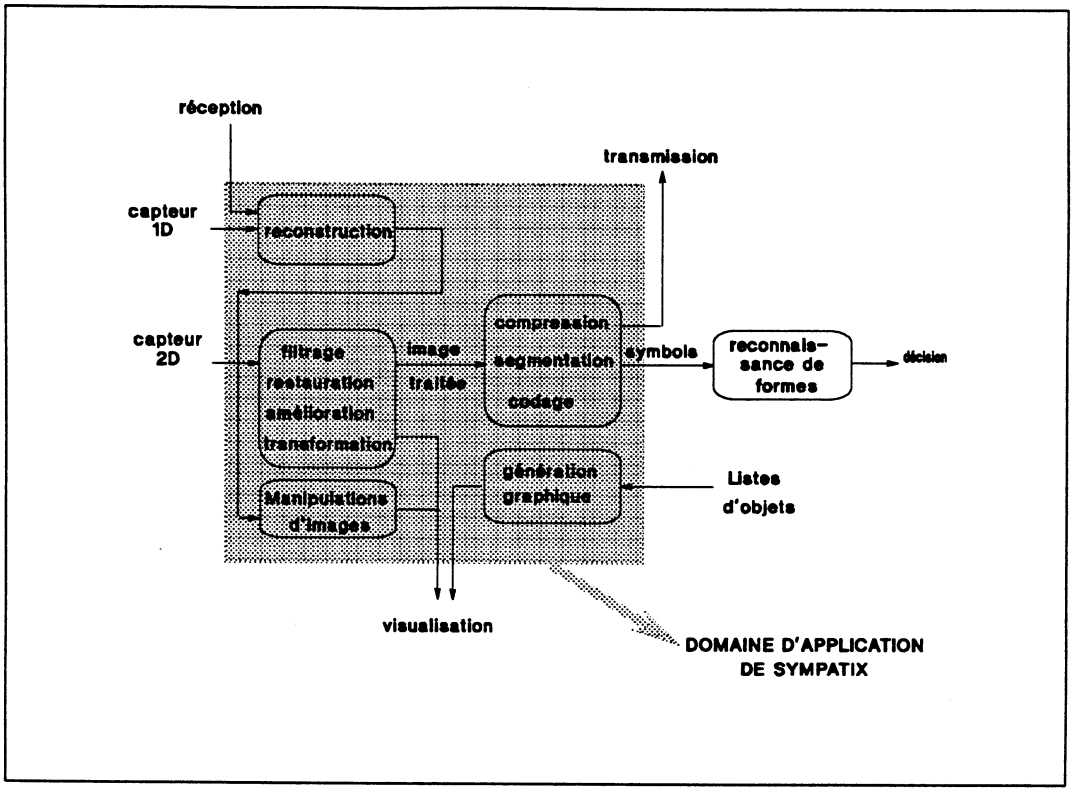

***Figure IV.1.12** : Domaine d'application du calculateur SYMPATIX présenté dans cette thèse.*

IV.2 : VHDL : SIMULATIONS ET SYNTHESE.

Voyons dans ce paragraphe les avantages, ainsi que les limites de la conception VHDL descendante afin d'en extraire une méthodologie réutilisable sur d'autres projets. Ainsi les performances de différents simulateurs sont présentées dans une première section, ensuite les contraintes liées à la synthèse et enfin les résultats de synthèse obtenus pour notre modèle de la cellule du réseau intelligent ouvert.

IV.2.1 : La simulation VHDL de haut niveau : une réalité?

Quelles sont les performances et les limites des simulateurs de modèles décrit en VHDL, jusqu'à quelle complexité peut-on pousser la simulation de systèmes? Pour répondre à ces questions, nous présentons ici les résultats obtenus, en temps de simulation sur différents produits, pour notre modèle comportemental de SYMPATI2.

Le modèle VHDL de SYMPATI2 est composé de 65 composants décrits en VHDL comportemental, 32 processeurs élémentaires, 32 mémoires RAM et une unité de commande. L'ensemble représente 14 000 lignes de VHDL, 320 processus fortement couplés 10 à 10 et 8500 signaux. Le benchmark consiste à exécuter une opération de filtrage par un sobel d'une image de 32×32 points codés sur 256 niveaux de gris. Le nombre d'événements est environ de 700 000 et le temps réel sur le calculateur SYMPATI2 pour effectuer ce benchmark est de 0.15 ms [COL92]. Quatre simulateurs ont été testés, le modèle "d'origine" a été élaboré sur le premier simulateur (A), ensuite il a été exporté sur les 3 autres simulateurs VHDL. Sur les quatre, trois simulent du VHDL compilé, un, le B, de l'interprété. En ce qui concerne la portabilité, le portage de A vers B s'est effectué sans problèmes, celui de A vers C a posé un petit problème d'affectation de signal et celui de A vers D fut complexe car D ne supportait pas, à l'époque, les déclarations de variables de type tableau. Il est important de noter que le modèle n'a pas été amélioré afin de le rendre plus rapide, il a été pris tel qu'il a été conçu. Les résultats présentés dans le tableau IV.2.1, ont été obtenus sur une station de travail créditée d'environ 18 MIPS VAX et possédant 64 Moctets de mémoire vive.

SIMULATEURS	A	B	C	D
Temps de simulation	15 mn	8 mn	11 mn	14 mn (1)
Rapport avec le calculateur (SYMPATI2)	$6\ 10^6$	$3.2\ 10^6$	$4.4\ 10^6$	$5.6\ 10^6$

(1) : Ce résultat est donné pour une simulation hors environnement graphique, le simulateur ne pouvant exécuter le benchmark dans son environnement graphique.

***TableauIV.2.1** : Comparaison des temps de simulation obtenus sur différents simulateurs logiques de modèles VHDL.*

Par rapport à un simulateur uniquement logiciel, les simulations VHDL sont 1000 fois plus lentes (pour le simulateur A). Une différence importante existe au niveau de la fonctionnalité entre le simulateur A et les autres simulateurs. A accepte différents types de modèles en entrée ce qui n'est pas le cas de B, C et D qui n'acceptent que du VHDL.

Certes les temps de simulations sont importants dans le choix d'un simulateur, mais au vu de ces résultats qui sont sensiblement les mêmes, il est nécessaire de tenir compte des ouvertures du simulateur (A est plus ouvert que les autres selon la remarque précédente), et de "l'ergonomie" du simulateur, à savoir ses capacités de débogage, son mode de visualisation des résultats etc (pour ces critères, actuellement, C nous semble le meilleur).

A la suite de ces résultats, il est important de noter que les simulations sur de grandes images ne peuvent être exécutées tous les jours, si bien qu'il est délicat d'effectuer de la recherche algorithmique sur un modèle VHDL. Celui-ci ne doit être exploité que pour la validation d'algorithmes sur l'architecture modélisée. La simulation VHDL est très gourmande en capacité mémoire si bien qu'il est préférable de disposer d'un calculateur ayant une grande taille de mémoire vive. Notons qu'en ce qui concerne la simulation d'un modèle structurel décrit en VHDL au niveau portes, les premiers résultats présentés dans [HUE91] sont mauvais, en effet, suivant les simulateurs, il y a blocage à partir d'un certain nombre de portes.

IV.2.2 : La synthèse VHDL : les contraintes.

La description en VHDL, ou de manière générale en un langage de description de matériel, est utilisée pour modéliser des systèmes électroniques. Cette modélisation est reprise pour simuler un composant existant, pour effectuer des simulations au niveau système ou bien elle est reprise pour synthétiser un composant à fabriquer. Si un standard VHDL existe pour la simulation (IEEE-1076), il n'en existe pas encore pour la synthèse, et aucun synthétiseur n'accepte IEEE-1076 comme entrée de synthèse. Alors, pour chaque outil, il est nécessaire de développer un modèle synthétisable, on ne profite plus de la qualité de standard, on devient dépendant des plates-formes.

Avant de présenter la synthèse du modèle de la cellule du réseau intelligent ouvert, il est très important de noter que les synthétiseurs VHDL ne sont pas des outils "presse-bouton", il est nécessaire de guider fortement l'outil et de lui présenter un modèle VHDL digeste. Le concepteur doit nécessairement bien connaître et l'outil et les règles qui régissent la fabrication d'un circuit. Suite à une première expérience de synthèse d'un module décrit en VHDL sur 2 outils différents, on constate que plus l'outil est contraignant, plus le temps de développement est long, mais meilleur est le résultat.

Tout d'abord, où se place l'outil de synthèse VHDL dans la chaîne de la conception descendante? Actuellement, les outils demandent en entrée une description au niveau transfert de registres (RTL), ou bien au niveau intermédiaire entre les niveaux transfert de registres et algorithmique, si bien qu'il est nécessaire d'effectuer, manuellement, le découpage en blocs ("partitionning") du modèle comportemental VHDL (figure IV.2.1). La vérification des 2 descriptions s'effectue par simulation ou preuve formelle. Néanmoins, certains outils génèrent automatiquement du VHDL synthétisable (au niveau RTL), à partir de descriptions non VHDL (machine d'état par exemple).

Une fois les modèles RTL obtenus, une première phase de synthèse intervient, elle génère une "netlist", hiérarchique ou non, indépendante de la technologie. Ensuite, un outil d'optimisation transforme ce schéma en une description électrique du circuit, description, elle, tributaire de la technologie cible.

Les fonctions VHDL qui ne sont pas supportées par notre outil de synthèse VHDL, comme par beaucoup d'autres outils, sont les suivantes :

- les fonctions n'ayant aucun rapport avec le matériel, tels les accès aux fichiers, les assertions et la généricité;

- les fonctions arithmétiques trop complexes telles la division et l'exponentielle;

- les fonctions impliquant un développement de matériel non connu dans sa quantité, telles les boucles;

- les fonctions synthétisables uniquement par connaissance de la technologie cible, telles les contraintes temporelles (after, transport);

- et enfin les fonctions nécessitant plusieurs couches de logique séquentielle, par exemple, il n'est pas possible d'utiliser plusieurs wait dans un même processus.

Par contre, les types énumérés, la déclaration de variables.., permettant un haut niveau d'abstraction, sont, elles, synthétisables.

De manière plus concrète, si nous reprenons le modèle de notre module de calcul présenté dans le chapitre précédent, les instructions marquées (☛) ne sont pas synthétisables :

```
ENTITY modèle_simple IS

TYPE fonction IS (add,sous,mul,div);
PORT      ( a,b: IN integer:=0;
            f: IN fonction:= add;
            s: OUT integer);

ARCHITECTURE compt OF modèle_simple IS
BEGIN
principal: PROCESS
VARIABLE result_intern : integer :=0;
CASE f IS
            WHEN add => result_intern:= a + b;
            WHEN sous=> result_intern:= a - b;
            WHEN mul => result_intern:= a × b;
            ☞WHEN div => result_intern:= a / b;
            ☞WHEN others ASSERT false
                                    REPORT "ERREUR fonction non définie dans
                                     architecture compt de modèle_simple
                                                SEVERITY ERROR;
            result_intern:=0;
END CASE;
☞s<= result_intern after 1 s;
WAIT ON a,b,f;
END PROCESS principal;
END compt;
```

Il est alors nécessaire de reprendre le modèle et de contourner ces instructions non synthétisables. Dans cet exemple, la plus contraignante est la troisième, celle due à la clause "after". Soit cette clause est un retard fonctionnel, auquel cas ce retard doit être obtenu grâce à une horloge rapide, soit c'est un retard du à la technologie, il doit donc être, dans ce cas, révisé après la synthèse. Pour cette seconde hypothèse, le concepteur doit revoir ce temps en fonction du résultat de la synthèse, il mesurera alors ce temps précisément sur le circuit. Pour les retards fonctionnels, le modèle compte X fronts d'une horloge rapide, avant de générer la sortie. Ce type de construction est complètement synthétisable. Sans horloge supplémentaire avec une correction finale manuelle, le modèle synthétisable pourrait être le suivant (si l'on excepte les contraintes d'initialisation) :

```
ENTITY modèle_simple IS

TYPE fonction IS (add,sous,mul,div);
PORT      ( a,b: IN integer:=0;
            f: IN fonction:= add;
            s: OUT integer);

ARCHITECTURE compt OF modèle_simple IS
BEGIN
principal: PROCESS
VARIABLE result_intern : integer :=0;
CASE f IS
            WHEN add => result_intern:= a + b;
            WHEN sous=> result_intern:= a - b;
            WHEN mul => result_intern:= a × b;
            WHEN div => result_intern:= fonction_division_synthétisable (a,b);
            WHEN result_int:=0;
END CASE;
s<= result_intern;
WAIT ON a,b,f;
END PROCESS principal;
END compt;
```

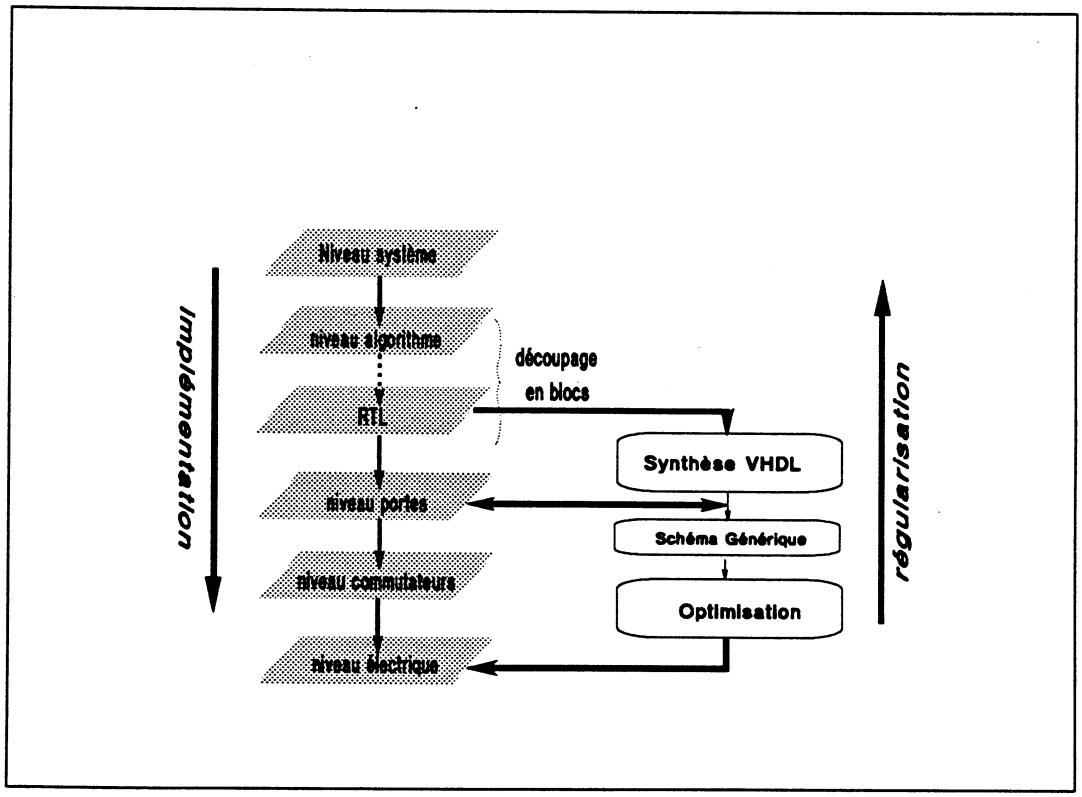

Figure IV.2.1 : Niveau d'intervention de notre outil de synthèse VHDL.

IV.2.3 : Les résultats de synthèse.

Ce paragraphe présente les résultats obtenus, au niveau de la synthèse VHDL, de la cellule du réseau intelligent ouvert. Ces résultats permettent de donner la complexité électronique des différentes fonctionnalités envisagées pour le réseau. Nous sommes partis du modèle VHDL comportemental issu de la phase de mise au point de notre réseau, modèle complètement déconnecté de la réalisation physique, donc de la synthèse. Quelques conseils de modélisation de haut niveau en vue de la synthèse sont donnés suite à notre expérience.

Tout d'abord, d'un modèle d'origine s'articule autour de 5 processus fondamentaux, 2 réservés à la gestion de la FIFO, les 3 autres gérant, soit l'écriture dans le réseau (P1), soit la récupération des messages (P2), soit le transfert des données (P3). Evidement les 2 processus gérant la FIFO n'ont pas été synthétisés, de tel circuits existent sur le marché. Donc, la synthèse de P1, P2 et P3 a été effectuée, les résultats suivant en ont été retirés.

Tout d'abord, il a été nécessaire d'ajouter une horloge rapide au circuit, en vue de générer les retards fonctionnels. Ensuite, P1 et P2 ont été éclatés en plusieurs processus afin de construire des structures synthétisables. Aussi, P1 représente 8 processus, et P2 deux processus. Notons que cette transformation n'aboutit pas à un modèle au niveau transfert de registres, mais à un niveau intermédiaire entre le niveau algorithmique et le niveau transfert de registres. Le modèle retenu pour effectuer cette synthèse, comprend les caractéristiques suivantes : l'emission du message par les processeurs, peut être du type point à point ou diffusion. La réception s'effectue, soit au niveau des FIFOs, soit en écriture directe dans la mémoire du processeur destinataire (mode DMA). La cellule ne calcule pas automatiquement l'adresse de stockage, mais c'est au processeur émetteur de calculer, le numéro du processeur destinataire de son message. Les résultats obtenus par notre synthétiseur, sans optimisation particulières, sont les suivants.

La réalisation du processus P1, nécessite l'utilisation des composants suivants :

- 1 icrémenteur 6 bits;
- 1 comparateur 6 bits;
- 8 bascules D;
- 65 latches D.

Le processus P2 fait appel, pour sa réalisation, à :

- 97 bascules D;
- 1 additionneur 16 bits.

Enfin P3 est réalisé au travers de 46 bascules D.

Le tableau IV.2.2 donne le coût en nombre de portes "standard cell" des trois processus avant et après optimisation. L'optimisation s'effectue par rapport à une technologie particulière, nous avons choisi, à titre purement illustratif, la technologie LSI 10K.

Technologie	P1	P2	P3	Total
Générique (sans optimisation)	484	747	338	1383
LSI 10K	194	338	184	712

***TableauIV.2.2** : Coût électronique, en portes logiques, des trois principaux processus. Le coût donné en "générique" représente le résultat issu du synthétiseur VHDL (Autologic_VHDL), le résultat donné en LSI 10K est celui obtenu après optimisation (Autologic).*

Ces premiers résultats de synthèse VHDL donne une idée de l'importance, au niveau électronique, d'une cellule du réseau intelligent ouvert. Aussi, associé à une FIFO de 256 mots de 16 bits, la cellule comporte, approximativement, 1400 portes logique. Cette cellule représente **25%** de l'électronique d'un processeur élémentaire actuel.

IV.2.3 : Conclusion.

Suite à cette expérience en conception descendante à base de VHDL, on peut retenir les grands principes suivants :

- ne pas compter, notamment pour les architectures dédiées au traitement d'images, sur le modèle VHDL de l'architecture pour effectuer des recherches algorithmiques, il faut se restreindre à la validation sur la structure d'algorithmes éprouvés, vu les temps de simulation obtenus;

• bien connaître son outil de synthèse VHDL! Ce logiciel d'aide à la conception n'est pas un produit miraculeux de type "presse bouton";

• ne jamais se déconnecter de la synthèse. Dès qu'une nouvelle idée d'architecture, formulée dans un processus non synthétisable, s'avère efficace et se trouve validée, transformer ce processus, intrus à la synthèse, en processus synthétisables;

• si l'outil le permet, travailler avec des types énuméré, de façon générale, simplifier au maximum l'écriture des modèles;

• travailler en logique synchrone, utiliser une horloge très rapide, par rapport à la fréquence extérieure du circuit, afin de générer des retards fonctionnels synchrones précis. Cette approche permet ainsi de créer des retards indépendants de la technologie. Par contre, les retards liés à la technologie ne sont connus qu'après l'implantation du circuit;

• ... que les électroniciens se rassurent, ce type de conception nécessite ("encore") des compétences électroniques en conception logique, ce n'est juste que le métier qui évolue.

IV.3 : EVOLUTIONS.

Maintenant, il est possible de passer à la réalisation physique du composant. Quelles options choisir? Quelle technologie utiliser? C'est à l'utilisateur de trancher, selon qu'il décide d'avoir un calculateur SIMD "general purpose", ou dédié, suivant le nombre de processeurs et le type d'applications envisagées. Le choix d'une vitesse de décalage entraîne de nombreuses conséquences qui sont présentées dans une première section. Ensuite, nous proposons quelques évolutions à apporter au processeur afin qu'il s'interface plus facilement et plus économiquement au réseau intelligent ouvert. Nous suggérons également quelques axes de recherche à envisager suite à cette thèse, et enfin nous concluons cette partie par une critique de notre méthode de travail, au niveau conception descendante, critique qui nous a conduit à proposer une méthode optimale de conception descendante.

IV.3.1 : La vitesse de décalage : ses conséquences.

On a vu que la vitesse limite des décalages est à déterminer par l'équation III.3.3.1 :

$$Td = \frac{Nmi}{M} \times Tc;$$

Nmi dépend de l'application, M des moyens, Tc actuellement vaut 120 ns et enfin Td dépend de la technologie utilisée pour réaliser le réseau. Td dépend également de la solution choisie pour intégrer le réseau. En effet, si les cellules sont intégrées avec les processeurs dans un même boîtier, la vitesse Td ne sera pas la même que si chaque cellule est intégrée seule dans son propre boîtier. Pour une technologie CMOS on peut atteindre un Td de 20 ns, pour de l'AsGa Td passe en deçà de 10 ns.

La réalisation du réseau intelligent ouvert peut même s'effectuer en technologie optique, mais cette solution exige l'utilisation de composants optiques pas encore disponibles sur le marché (comparateurs optiques, cellules de mémorisation optique etc). En effet, si l'interconnexion entre les cellules est réalisée par des liaisons optiques, il est nécessaire d'utiliser ce type de composants afin de garder le bénéfice de ces liaisons rapides. Toutefois, il peut être envisagé d'effectuer en optique, uniquement l'interconnexion entre les cellules, afin de bénéficier des avantages de ce type de connexions, à savoir l'immunité au bruit et la rapidité.

Le choix de cette vitesse de décalage influe directement sur les conditions de saturation de ce réseau, et suivant le nombre de processeurs, ce choix détermine le nombre d'instructions à insérer entre chaque écriture, à savoir Nmi. A titre illustratif, pour un Td de 20 ns et un Tc de 120 ns, ce nombre évolue linéairement de la manière suivante :

	32 PEs	64 PEs	128 PEs	256 PEs
Nmi	6	11	22	43

***TableauIV.3.1** : Nombre de micro-instructions nécessaires entre chaque écriture réseau afin d'éviter tout éventuel blocage du calculateur.*

Comme nous l'avons vu précédemment, ce nombre diminue si des FIFOs émettrices sont utilisées ou si un double réseau est mis en oeuvre.

Cette vitesse détermine la bande passante de SYMPATIX en entrées/sorties. Cette bande passante en méga octets par seconde vaut :

$$BP = BP_{SYMPATI2} + 2\times(1/Td), \qquad (IV.3.1.1)$$

avec $BP_{SYMPATI2}$, la bande passante de SYMPATI2 qui vaut 13.5 Moctets/s. Le mode de stockage DMA est utilisé pour acquérir les images. Pour un Td de 30 ns, cette bande passante s'élève à 80 Moctets/s. Mais attention, le réseau transfert des données sur 16 bits et une seule adresse de stockage est jointe à ces 16 bits. Cette bande passante représente la vitesse de transfert entre la mémoire image, le(s) processeur(s) spécialisé(s) et le processeur de haut niveau.

IV.3.2 : Améliorations du processeur.

Côté processeur, il serait souhaitable de disposer de certaines fonctionnalités afin d'améliorer les performances de la structure. Après sa refonte, le processeur élémentaire de SYMPATIX doit :

- pouvoir écrire directement dans le registre d'index, plutôt que de passer par LIG et COLO;

- disposer d'un mode d'adressage indexé plus puissant permettant des adressages directs, indirects...;

- disposer, à l'intérieur du processeur, du module de calcul de x et y et donc de ces registres, ce module serait alors moins coûteux au niveau électronique que ce qu'il représente actuellement dans la cellule;

- disposer d'un ensemble de registres de sauvegarde de contexte ou d'une pile afin de pouvoir exécuter plus facilement, et plus rapidement des sous-programmes;

- éventuellement, intégrer la cellule avec le processeur élémentaire ce qui éviterait de "dupliquer" sur un même noeud de traitement le module de calcul d'adresses hélicoïdales nécessaire pour le DRES.

De plus, si la taille des images à traiter devient grande, il est nécessaire d'augmenter la largeur des chemins de données, des registres et des entrées/sorties de type données afin de générer les coordonnées d'un point image en 2 écritures, l'une pour les x, l'autre pour les y.

IV.3.3 : Travaux à envisager.

Il est démontré qu'une approche de conception descendante permet d'effectuer des conceptions matérielles et logicielles concurrentes. En ce qui concerne le développement de nouveaux algorithmes sur notre structure, s'il ne s'agit que de portages d'algorithmes en 4LP, le modèle VHDL peut être utilisé comme support de simulation, par contre pour ce qui est de la recherche algorithmique, celle-ci ne peut s'effectuer avec ce modèle VHDL, il est nécessaire de développer un nouveau simulateur uniquement logiciel, basé sur ce modèle, afin d'effectuer plus rapidement ces recherches. Toutefois, une fois un nouvel algorithme développé, celui-ci devra être testé sur le modèle VHDL afin de prévenir les erreurs éventuelles commises lors de l'écriture de ce simulateur logiciel.

Des études peuvent être également menées en ce qui concerne la tolérance aux pannes de ce type de calculateur. En effet, le coût de la redondance, nécessaire pour rendre le système tolérant aux pannes, est le suivant :

$$Cpt = \frac{\text{matériel supplémentaire}}{\text{matériel initial}}; \qquad \text{(IV.3.3.1).}$$

Pour un circuit électronique classique, si cette tolérance est obtenue par redondance et vote majoritaire, Cpt vaut 200%. Par contre, sur SYMPATIX, pour obtenir un tel système, il est nécessaire de disposer de 3 unités de commande, mais que de (M+R) noeuds processeur (processeur+mémoire+cellule). Ainsi ce coût est donné par :

$$Cpt(SYMPATIX) = \frac{(2 \times Vuc + R \times Vpe)}{(Vuc + M \times Vpe)} \qquad \text{(IV.3.3.2).}$$

avec Vpe et Vuc les volumes électroniques, respectivement du processeur élémentaire de SYMPATI2 et de l'unité de commande.

On a R < M, voire R << M, si bien que, en fonction du nombre de processeur et si l'on considère en grossière approximation que Vuc=6×Vpe, ce coût évolura suivant la courbe présentée en figure IV.3.1.

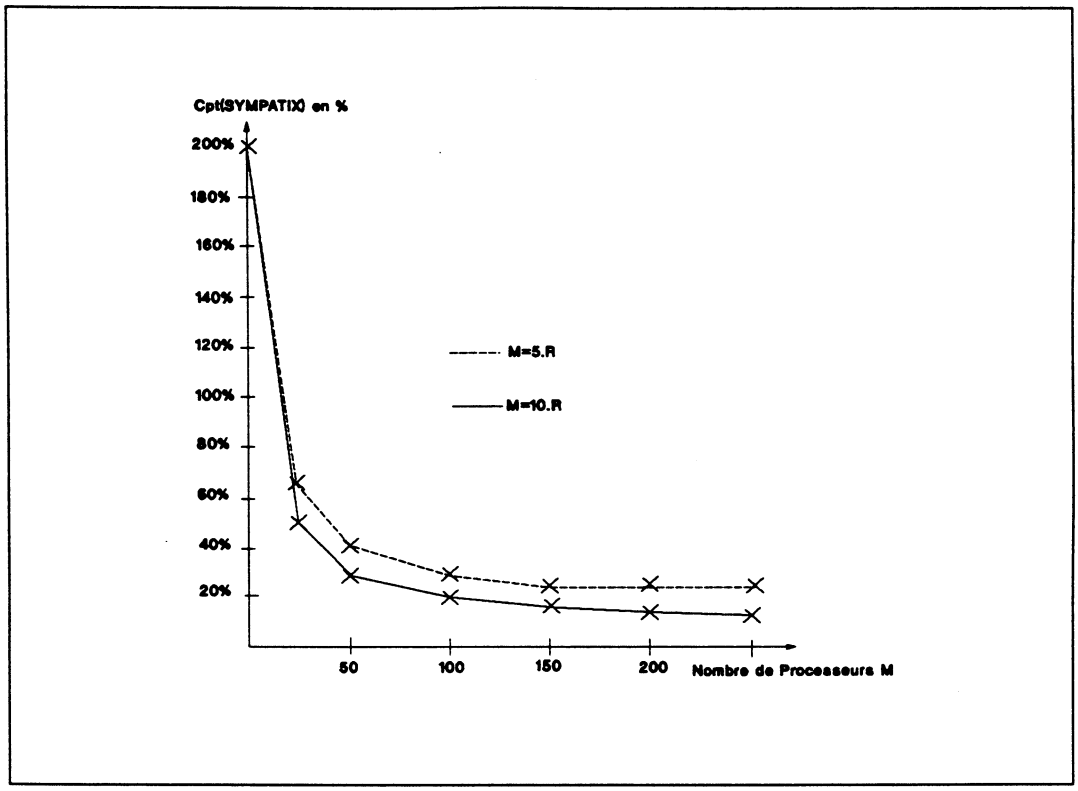

***Figure IV.3.1** : évolution du coût électronique supplémentaire pour rendre SYMPATIX tolérant aux pannes.*

Ainsi, plus le nombre de processeurs est important, plus ce coût est faible. Le vote majoritaire s'effectue entre les processeurs de la manière suivante. Le réseau intelligent ouvert véhicule des vecteurs de test de type donnée sur tous les processeurs, la même donnée à tous si bien que le mode de diffusion est utilisé, ensuite les processeurs effectuent des opérations sur ces données, puis comparent 2 à 2 leurs résultats. S'ils sont identiques, c'est bon, sinon les résultats sont envoyés vers le hôte qui détecte le (ou les) processeur(s) défaillant(s) et lance une phase de reconfiguration de la structure en isolant le(s) processeur(s) défaillant(s), puisant dans les processeurs de "rechange" au nombre de R pour le(s) remplacer. Ce type d'approche permet une tolérance aux pannes fonctionnelles.

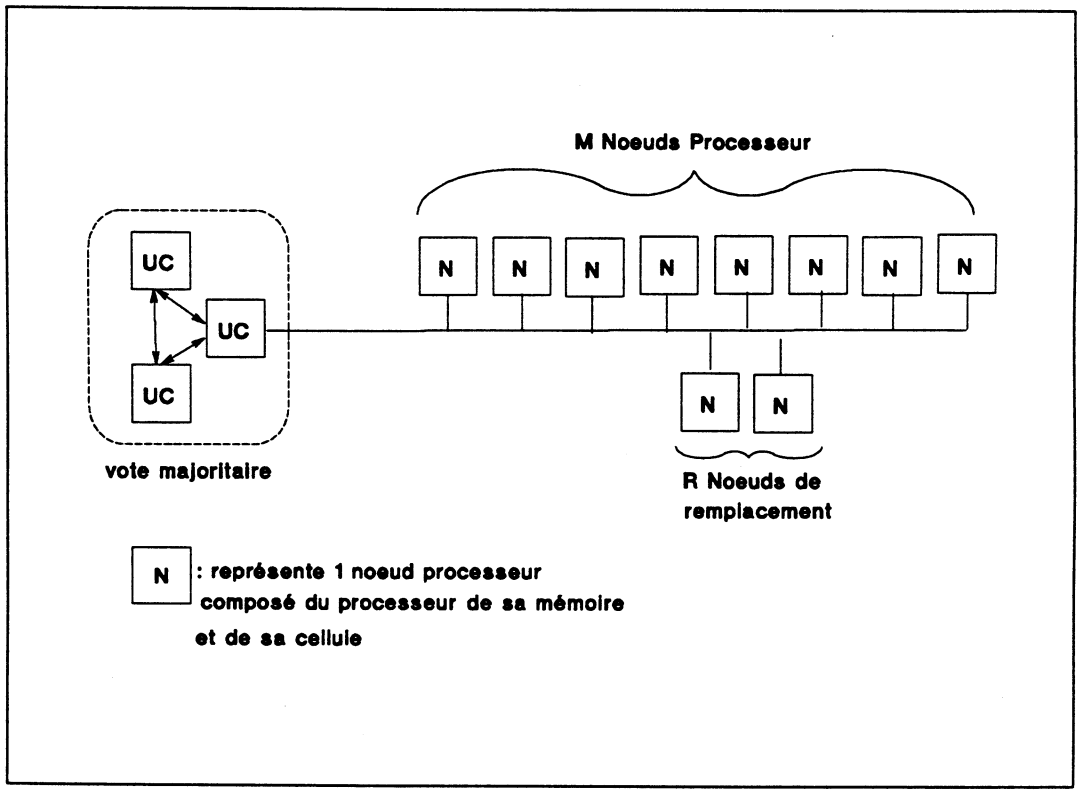

Figure IV.3.2 : Schéma de principe de la structure de SYMPATIX tolérante aux pannes. Il n'y a que M noeuds processeur qui travaillent, mais dès qu'un de ces noeuds n'est plus opérationnel, il est remplacé par l'un des R Noeuds processeur.

Enfin, dans le but de disposer d'une structure de traitement d'images couvrant tous les niveaux, il est nécessaire d'étudier l'interfaçage de SYMPATIX avec un processeur de type "general purpose", MIMD par exemple. Cet interfaçage est facilité par le réseau intelligent ouvert qui véhicule à grande vitesse les données vers, ou de ce processeur. De même, si la structure doit supporter des opérations graphiques, l'utilisation du réseau en mode de lecture à distance doit être envisagée, ce qui pourrait conduire à réaliser une cellule plus complexe donc un réseau plus complet.

IV.3.4 : "... et si c'était à refaire".

En ce qui concerne la méthodologie de conception descendante, nous proposons la chronologie suivante :

- 1 : écriture et validation d'un modèle comportemental du système à réaliser, puis découpage en blocs de ce modèles, et transformation de chacun des processus comportementaux en processus synthétisables (niveau RTL, voire

algorithmique). Le concepteur ayant conçu le processus de haut niveau, doit concevoir les processus synthétisables, ou doit fortement aider lors de cette opération;

• 2 : Une fois le modèle de niveau RTL obtenu, et par la même le découpage en blocs du système, la conception des circuits peut alors commencer de même que les développements logiciel. Pour cette phase, c'est ce modèle RTL qui doit servir de référence, non le modèle comportemental de haut niveau. Tous ces développements s'effectuent en parallèle, et des modifications interviennent dès que l'implantation physique des composants est réalisée;

• 3 : Dès que les circuits sont fabriqués, l'assemblage et le test du système peuvent débuter. Les séquences de test auront été élaborées lors de la phase 2. Tout doute sera soulevé en faisant appel au modèle de référence établi à l'issue de la première phase.

Il est donc indispensable, lors de la conception du premier modèle de garder un pied dans la synthèse afin d'éviter d'écrire des fonctions non synthétisables. Pour cela, il est nécessaire de connaître l'outil de synthèse qui sera utilisé, connaissances que nous n'avions pas lors de notre expérience. Si on s'est fait plaisir en écrivant de merveilleux processus lors du travail d'élaboration du modèle, on s'en est "mordu" les doigts lors de la synthèse.

Par contre, l'approche utilisée pour définir le réseau intelligent s'est avérée très fructueuse car elle préserve des erreurs ou des oublis de conception au niveau fonctionnel, tant par rapport à l'application, que par rapport à la structure du système.

IV.4 : CONCLUSION.

Ce chapitre présente les performances du calculateur SYMPATIX pour le domaine d'application visé, à savoir le traitement d'images, et pour les applications graphiques. Puis sont présentés des résultats sur la simulation VHDL et la synthèse VHDL, et enfin le dernier paragraphe donne des perspectives de travaux à effectuer et dresse une méthodologie sur la conception électronique descendante.

Pour ce qui est des performances de la structure, il est constaté que le réseau intelligent ouvert apporte, au concept SYMPATI, plus de libertés en ce qui concerne les échanges entre processeurs, mais que la qualité des échanges est fonction du nombre de processeurs et de la réalisation physique du réseau. Plus la technologie d'implantation de ce réseau est rapide, plus le nombre de processeurs peut être important, si l'on désire conserver la linéarité au niveau des performances.

Il ressort de ce chapitre, qu'il n'est pas conseillé d'effectuer de la recherche algorithmique sur le modèle VHDL. Dans ce cas, il est préférable de construire un simulateur uniquement fonctionnel.

Pour ce qui est de la réalisation électronique, il est relativement aisé de disposer du coût électronique des modifications envisagées. Il est ainsi démontré qu'une cellule représente environ 1/4 du processeur actuel. Suite à ce travail, dans la méthodologie de conception électronique proposée, il est indiqué que le concepteur ne doit à aucun moment faire abstraction des contraintes de synthèses de son outil, ceci afin d'obtenir des modèles synthétisables et synthétisés conforme à la spécification.

Conclusion.

L'utilisation des techniques du parallélisme devient une nécessité pour les architectes de calculateurs dédiés à la vision. Cette approche est, en effet, actuellement la seule issue permettant de répondre aux besoins grandissants des applications industrielles du traitement d'images. Des différents types de calculateurs parallèles, le SIMD s'adapte le mieux aux opérations de traitements d'images iconiques ou de bas niveau. En revanche, pour les opérations de traitement d'images de moyen niveau, opérations utilisées pour extraire les entités représentatives d'une image, ces calculateurs perdent énormément en efficacité.

Afin de mettre en évidence les types de problèmes rencontrés lors de cette phase du traitement d'images, une étude algorithmique a été effectuée. Elle montre les limites des ces calculateurs et en mesure leur importance. Il est ainsi démontré que le problème majeur vient des capacités d'interconnexion entre processeurs. En effet cette classe d'opérations exige des échanges aléatoires, ou asynchrones, entre les éléments de calculs, échanges difficilement supportés par les calculateurs SIMD. A la suite de cette étude, un réseau d'intercommunication, le Réseau Intelligent Ouvert, a été proposé. Il est dédié aux calculateurs SIMD, et son interfaçage avec SYMPATI2 a donné naissance à SYMPATIX, le nouveau calculateur proposé dans cette thèse.

Afin d'élaborer et de valider SYMPATIX, un modèle comportemental du système a été écrit dans le but d'effectuer la conception électronique descendante du système. Par une telle approche, le comportement d'un circuit est validé dans son environnement, pour ensuite être réalisé à partir de sa description comportementale. Aussi, par cette approche, SYMPATIX a pu être élaboré, évalué puis validé directement sur des opérations de traitement d'images. Le meilleur compromis coût électronique/performances de la structure de SYMPATIX est alors retenu.

Les performances de SYMPATIX obtenues sont très encourageantes. En effet, la structure supporte le domaine d'application visé, à savoir le traitement d'images de moyen niveau. De manière plus générale, les performances obtenus sur cette nouvelle architecture

démontrent que le réseau intelligent ouvert étend le concept SIMD. Il offre aux processeurs d'une architecture synchrone la possibilité d'effectuer des échanges asynchrones (ou aléatoires). Outre ces nouvelles fonctionnalités, le réseau intelligent ouvre le système au monde extérieur permettant ainsi à la structure de s'interfaçer facilement et rapidement avec un calculateur de haut niveau ou avec des processeurs spécialisés. Il devient ainsi aisé de concevoir un système à base de SYMPATIX supportant toutes les opérations de traitement d'images.

L'approche dite de conception descendante a été éprouvée pour élaborer SYMPATIX. Si elle est effectivement très efficace, sa mise en oeuvre doit être rigoureuse. Pour faciliter son utilisation, à la suite de notre expérience, une méthodologie de conception descendante est déraillée dans cette thèse. Elle offre, à l'architecte de systèmes électroniques, la possibilité de mesurer très rapidement les conséquences de ses choix architecturaux.

Suite à ces travaux, la réalisation du réseau intelligent ouvert peut débuter, de même les développements d'applications peuvent se poursuivre, soit sur le modèle comportemental VHDL de la structure, soit sur un simulateur logiciel de SYMPATIX. Les temps de simulation obtenus sur ce second simulateur seront bien plus rapides que ceux donnés par la simulation VHDL. Enfin, les calculateurs parallèles de type SIMD sont, parmi les structures multiprocesseurs, les plus tolérantes aux pannes logicielles. Aussi, il serait intéressant de développer les fonctions nécessaires pour permettre la reconfiguration de SYMPATIX en cas de panne de l'un de ces éléments de calcul. Avantageusement, le réseau intelligent ouvert augmente les capacités de la structure pour ces opérations.

Références.

REFERENCES DU CHAPITRE 1.

[ANN87] Annaratone M., Arnould E., Gross T., Kung H.T., Lam M., Menzilcioglu O., Webb J.A. *"The Warp Computer : Architecture, Implementation, and Performance"*, Trans on Computer, Vol. C 36, No 12, Décembre 1987, pp 1523-1537;

[BAL82] Ballard D.A., Brown C.M."Computer Vision", ed : Prentice Hall, 1982;

[BHU91] Bhuyan L.N, Yang Q., Agrawal D.P. "Performanrce of Multiprocessor Interconnection Networks", Computer, Vol 24, No 2, Février. 1991;

[CHA90] V. Chaudhary, J.K. Aggarwal, "Parallelism in Computer Vision : a Review", dans [KUM90], pp 271-309

[COL88] T. Collette, "Etude d'un Processeur d'Images Embarqué à Architecture Parallèle pour Robot Mobile", mémoire DEA d'électronique, Université de Clermont Ferrand2, Octobre 1988;

[COR91] Corcoran E. "Les Superordinateurs", Pour La Science, No 161, Mars 1991;

[CRO91] Crow J.D. "Optical Interconnects Speed Interprocessor Nets", Circuits and Device, Vol 7, No 2, Mars 1991;

[CYP89] R. Cypher, J.L.C Sanz, "SIMD Architectures and Algorithms for Image Processeing and Computer Vision", IEEE Transaction on Acoustics, Speech, and Signal Processing, Vol 37, No 12, décembre 1989, pp 2158-2174;

[DER92] J.P. Dérutin, B. Besserer, T. Tixier, "A Parallel Vision Machine : TRANSVISION", Proceeding CAMP'91, pp 241-251, Paris, Décembre 1991;

[DIN89] Dinning A. "A survey of Synchronization Methods for Parallel Computers", Computer, Vol 22, No 7, Juillet 1989 ;

[DUC90] Ducan R."A Survey of Parallel Computer Archiectures", Computer, Vol 23, No 2, Février 1990, pp 5-16;

[DUF86] M.J.B. Duff, "Intermediate-Level Image Processing", Ed : Academics Press, 1986;

[FEN81] T. Feng, "A Survey of Interconnection Networks", Computer, Vol 14, No 12, Décembre 1981, pp 12-27;

[FOR87] Fortes J.A.B., Wah B.W. "Systolic Arrays-From Concept to implementation", Computer, Vol 20, No 7, Juin 1987, pp 12-17;

[FOU86] T.J. Fountain, "Array Architectures for Iconic and Symbolic Image Processing",

[FOU87] Foulser D.E., Schreiber R. "The Saxphix Matrix1 : A General Purpose Systolic Computer", Computer, Vol 20, No 7, Juin 1987, pp 35-43;

[GON87] R.C. Gonzalez, P. Wintz, "Digital Image Processing", Ed : Addisson Wesley, 1987;

[HOC88] R.W. Hockney, C.R. Jesshope, "Parallel Computer 2", Ed : Adam Hilger, 1988;

[HWA84] Hwang K., Briggs F.A."Computer Architecture and Parallel Procesing", éditions : International Studient Edition, 1984;

[HWA87] Hwang K."Advanced Parallel Processing with Supercomputer Architectures", Proceedings of the IEEE, Vol 75, No 10, Octobre 1987;

[JAI89] Jain A.K. "Fundamentals of Digital Image Processing", ed : Prentice Hall, 1989;

[KUE86] J.T. Kuehn, H.J. Siegel, "Multifunction Processing with PASM", pp 209-229 dans [DUF86];

[KUM90] V. Kumar, P.S. Gospalakrishnan, L.N. Kanal "Parallel Algorithms for Machine Intelligence Vision", ed : Spinger Verlag, 1990;

[KUN82] Kung H.T. "Why Systolic Architectures?", Computer, Vol 15, No 1, Janvier 1982, pp 37-46;

[KUN87] Kung H.T., Lo S.C., Hwang J.N. "Wavefront Array Processors-Concept to Implementation", Computer, Vol 20, No 7, Juin 1987, pp 18-33;

[KUN88] Kung S.Y. "VLSI Array Processors", éditions : Prentice Hall, 1988;

[LOU91] Louri A. "Three Dimentional Optical Architecture and Data-Parallel Algorithms for Massively Parallel Computing", Micro, Vol 11, No 2, Avril 1991;

[PAR92] N. Paris, "Une synthèse des langages C pour le parallélisme des données", 4ième recontre du parallélisme, Villeneuve d'Ascq, 18-20 Mars 1992, pp 104-107;

[PIR90] A. Pirson, "Conception et simulation dArchitectures Parallèles et Distribuées pour le Traitement d'Images", Thèse de l'Institut National Polytechnique de Grenoble, le 4 Mai 1990;

[SAN88] J.L.C. Sanz, "Advances in Machine Vision", ed : Spinger-Verlag, 1988;

[SHU88] D. B. Shu, G. Nash, C. Weems, "Image Understanding Architecture and Applications", dans [SAN88], pp 297-355;

[SKI88] Skillicorn D.B. "A Taxonomy for Computer Architectures", Computer, Vol 21, No 11, Novembre 1988;

[SRI86] Srini V.P. "An Architectural Comparison of Dataflow Systems", Computer, Vol 19, No 3, Mars 1986;

[UNG58] Unger SH "A computer Oriented Toward Spacial Problems", Proc. IRE, Vol 46, pp 1744-1750, 1958;

REFERENCES DU CHAPITRE 2.

[ADM88] P. Adam, "Manuel d'utilisation du langage assembleur de la machine SYMPATI2", Note Interne CEA N° SIR/88.34;

[ALG90] Algotronix, "CHS2*4 User Manual", Algotronix Ltd, Scotland 1990;

[BAL81] D.A. Ballard, "Strip Trees : a hierarchical Representation for Curves", Communication of the AMC, Vol 24, No 5, Mai 1991;

[BAL82] : Ballard D.A., Brown C.M."Computer Vision", ed : Prentice Hall, 1982;

[BAS85] J.L. Basille, "Structures Parallèles et Traitement d'Images", Thèse d'Etat de l'Université Paul Sabatier de Toulouse, décembre 1985;

[BOU90] T. Bourré, "Vision par ordinateur et réseau de Tranputers. Contribution à l'étude d'une fonction de traitement d'images pour un système automatique de navigation", Thèse de docteur de l'Université de Paris 6, mars 1990;

[CEN92] Centralp, "System Openvision", documentation Centralp, SPE MTBII 078, Avril 1992;

[CEL89] M. Celenk et P. Lakshman, "Parallel Implementation of the Split and Merge Algorithm on Hypercube Processors for Object Detection and Recognition", SPIE vol 1095 Application of Artificial Intelligence VII, 1989;

[CHAa90] V. Chaudhary, J.K. Aggarwal, "Parallelism in Computer Vision : a Review", pp 271-310 dans [KUM90] chapitre I.

[CHAb90] F. Chantemarge, S. Nicolle, J.P. Dérutin, "Parallélisation of the Splitting Step of a Quadtree Algorithm", Proceeding de l'IASTED Lugono Suisse, juin 1990;

[CHO90] A.N. Choudhary, J.H. Patel, "Parallel Architectures and Parallel Algorithms for Integrated Vision Systems", ed : Kluwer Academic Publishers, 1990, pp 69-76;

[CLE90] P. Clermont, "Méthode de Programmation de Machine Parallèle Pyramidale. Application en Segmentation d'Images", Thèse de Doctorat de l'Université de Paris VII, janvier 1990;

[COLa91] T. Collette, H. Essafi, J. Kaiser, R. Schmit, "Le Traitement d'Images Moyen Niveau et SYMPATI : Proposition d'une Architecture en vue d'un parallélisme total", Rapport interne CEA/DTA/LETI, numéro DTA/LETI/DEIN/SIR/91.18;

[COLb91] T. Collette, H. Essafi, J. Kaiser, R. Schmit, "Architecture de système en tableau de processeurs à structure parallèle" brevet n°9109649, septembre 1991;

[COLa92] T. Collette, H. Essafi, K. Kaiser, D. Juvin, "SYMPATIX : a SIMD Computer Performing the Low and Intermediate levels of Image Processing", PARLE'92, Ed : Springer-Verlag, pp 147-161, juin 1992;

[COLb92] T. Collette, H. Essafi, K. Kaiser, D. Juvin, "Low and Intermediate Level Image Processing on SYMPATIX, a SIMD Parallel Computer", Proceeding du 11ième IAPR, la Haye, Septembre 1992;

[CYP90] R.E. Cypher, J.L.C. Sanz, L. Snyder, "Algorithms for Image Component Labelling on SIMD Mesh-Connected Computers", IEEE Transaction on Computers, Vol 39, No 2, février 1990, pp276-281;

[DER89] H.F. Derek, P.& R. Jayakumar, "Improvements ans Systolic Implementation of the Hough Transformation for Straight Line Detection", Pattern Recognition, Vol 22, No 6, pp 697-706, 1989;

[DIN90] I. Dinstein, G.D. Landau, "Parallel Algorithms for Contour Extraction and Coding on an EREW PRAM Computer", Pattern Recognition Letters 11, février 1990, pp 87-93;

[ESS88] H. Essafi, "Les processeurs ligne en traitement d'images", thèse de l'Université Paul Sabatier de Toulouse, 1988.

[ESS92] H. Essafi, M. Pic, D. Juvin, "K-Project/first Step : to Improve Data Manipulations and Representations on Parallel Computers", COMPAR92-VAPD V, Lyon 1-4 septembre 1992;

[GEM86] P. Gemmar, "Considerations on Parallel Solutions for Conventional Image Algorithms", pp 83-99 dans [DUF86] chapitre I;

[GON87] R.C. Gonzalez, P. Wintz, "Digital Image Processing", Ed : Addisson Wesley, 1987;

[GUE88] C. Guerra, S. Levialdi, "Computer Vision : Algorithms and Architectures", dans [SAN88] du capitre I;

[HAN91] G. Hantelle, J. kaiser, R. Schmit "Processeur de traitement d'images EPA4 : Spécifications techniques", Note CEA interne réf : IRDI/D.LETI/DEIN/91.36, 1991;

[HUM86] R. Hummel, "Connected Component Labelling in Image Processing with MIMD Architectures", pp 101-127 dans [DUF86] chapitre I;

[JAI89] Jain A.K. "Fundamentals of Digital Image Processing", ed : Prentice Hall, 1989;

[JON90] P.J. Jonker, E.R. Komen, R.P.W. Duin, "Architectures for Multidimentional Low and Intermediate Level Image Processing", MVA'90, IAPR Workshop on Machine Vision Applications, novembre 1990, pp 307-316;

[JUV83] D. Juvin, "Contribution à la reconnaissance automatique des images appliquée à la robotique", Thèse de Docteur-Ingénieur, Université de Paris Sud, Paris 1983;

[JUV88] D. Juvin, J.L. Basille, H. Essafi, J.Y. Latil, "Sympati2, a 1.5D Processor Array for Image Applications", Proceeding de l'EUSIPCO, Grenoble 1988;

[KOM90] E.R. Komen, "Low-level Image Processing Architectures : compared for some non linear recursive neighbourhood operations", PHD Thesis Delft, octobre 1990;

[LAB88] M. Labarère, J.P. Krief, B. Gimonet, "Le Filtrage et ses applications", editions : Cepadues-Edition, 1988, pp153-171;

[LIT89] J.J. Little, G.E. Blelloch, T.A. Cass, "Algorithmic Techniques for Computer Vision on a Fine-Grained Parallel Computer", IEEE Transaction on Pattern Analysis and Machine Intelligente, Vol 11, No 3, mars 1989, pp 244-257;

[LSI92] LSI Logic, "Digital Signal Processeing (DSP) Databook", Septembre 1991, pp 124-134;

[MAR89] S. Marshall, "Review of Shape Coding Techniques", Image and Vision Computing, Vol 7, No 4, novembre 1989, pp 281-294;

[MAR90] A. Marglit, A. Rosenfeld, "Reducing the Expected Computational Cost of Template Matching Using Run Length Representation", Pattern Recognition Letter 11, avril 1990, pp 255-265;

[OUS87] A. El Oussoul, "Etude et Realisation d'un Reseau Local à Insertion de Registres : Spécification d'un Protocole Déterministe de Niveau 1 et 2", Thèse de Docteur de l'Université de Clermont II, Juin 1987;

[PAG88] I. Page, "Parallel Architectures and Computer Vision", Ed : Oxford Science Publication, 1988;

[PLE90] Plessey Semiconductors, "ERA 60100 Electrical Reconfigurable Array - ERA", Plessey, Avril 1990;

[PRES89] K. Preston, "The abingdon cross benchmark survey", Computer, vol 22, No 7, Juillet 1989.

[RAN90] S. Ranka, S. Sahni, "Computing Hough Transforms on Hypercube Multicomputers", The Journal of Supercomputing, 4, pp 169-190, 1990;

[SCH91] R. Schmit, "Définition et Réalisation de l'Unité de Commande d'un Processeur Ligne Cible Pour le Traitement d'Images", Thèse d'études doctorales, Université de Paris XI, Juin 1991;

[SLE88] A.C. Sleigh, C.J. Radford, G.J. Harp, "RSRE Experience Implementing Computer Vision Algorithms on Transputers, DAP and DIPOD Parallel Processors", pp 133-155 dans [PAG88];

[SYP89] R.E. Cypher, J.L.C. Sanz, "SIMD Architectures and Algorithms for Image Processing and Computer Vision", IEEE Transaction on Accoustics, Speech, and Signal Processing, Vol 37, No 12, décembre 1989, pp 2158-2174;

[XIL92] Xilinx, "The Programmable Gate array book", Xilinx, 1992;

[WEE89] C.C. Weems, S.P. Levitan, A.R. Hanson, E.M. Riseman, "The Image Understanding Architecture", International Journal of Computer Vision, 2, pp 251-282, 1989;

REFERENCES DU CHAPITRE 3.

[AGE92] "L'age du Silicium", Collection du Musée d'Histoire Naturelle de Paris, 1992;

[AIR90] R. Airiau, J.M. Bergé, V. Olive, J. Rouillard, "VHDL du Langage à la Modélisation", Editions Presses Universitaires Romandes, 1990;

[BOR89] D. Borione, J.L. Paillet, L. Pierre, H. Collavizza, "Modélisation fonctionnelle et preuve de circuits digitaux", TSI, Vol 8, n° 6, 1989;

[CAR92] D.E. Carter, B.S. Baker, "CE Concurrent Engineering. The Product Development Environment for the 1990s", éditions Addison Wesley, 1992;

[DAS87] S. Dasgupta, "Computer Architecture : a Modern Synthesis", Edition WILEY, 1987;

[HOL91] A. Hohl, "Incremental Design - Application of Software-Based Method for High-level Hardware Design with VHDL", Proceeding de l'EuroVHDL 91, pp 133-140, Stockholm septembre 1991;

[IEE88] "IEEE Standard VHDL Language Reference Manual Standard 1076-1987" New York : IEEE : 1988;

[LAT90] D. Latard, "Dossier Asic", note intrne CEA, référence DSYS/SETIA/90213/DL, Juin 1990;

[LIP89] R. Lipsett, C. Schaefer, C. Ussery, "VHDL : Hardware Description and Design", Edition KAP, 1989;

[RAM91] F.J. Rammig, "Approaching System Level Design", Proceeding de l'EuroVHDL 91, pp 6-23, Stockholm septembre 1991;

[ROU91] D. Rouquier, "VHDL : un langage pour la modélisation des circuits et des systèmes", L'écho des Recherches n° 143, pp 45-56, 1er trimestre 1991;

[SAV88] Y. Savaria, "Conception et spécification des circuits VLSI", édition de l'Ecole Polytechnique de Montréal, 1988.

REFERENCES DU CHAPITRE 4.

[COL92] T. Collette, J. Kaiser, R Schmit, "Modélisation VHDL et évaluation d'une architecture parallèle dédiée à la vision", Workshop AFCET Synthèse et Compilation d'architectures, Grasse, Janvier 1992;

[HUE91] M. Hueber, S. Lasserre, A. de Monteville, "VHDL Experiments on Performance", Proceeding de l'Euro-VHDL91, pp 282-292, Stockholm 1991;

[LET91] L Letellier, "De l'étude du Tracé de Segment à sa Parallélisation sur une Structure SIMD", Rapport intrne CEA, N° DTA/LETI/DEIN/SIR/91-19, avril 1991;

[LSI92] LSI Logic, "Digital Signal Processeing (DSP) Databook", Septembre 1991, pp 135-155;

[RAN90] S. Ranka, S. Sahni, "Computing Hough transforms on Hypercube Multicomputers", journal of supercomputing, 4, 169-190, éditions Kluwer Academic Publishers, Boston, 1990;

[WEE89] C.C. Weems, S.P. Levitan, A.R. Hanson, E.M. Riseman, "The Image Understanding Architecture", International Journal of Computer Vision, 2, pp 251-282, 1989;

ANNEXES.

Annexe A1 :

Opérations de Traitement d'images Moyen Niveau : Présentation des Opérations et de leur Parallélisation.

Cette annexe présente, et le principe, et la parallélisation sur une architecture synchrone(SYMPATI2), des algorithmes retenus afin d'évaluer notre structure pour les opérations de traitement moyen niveau. Le lecteur peut se reporter à [BAL82, GON87, JAI89, MAR89] pour de plus amples détails de ces algorithmes et à [GEM86] afin de prendre en compte tous les critères influents sur la parallélisation d'opérations. Sont présentés dans cette annexe, les algorithmes d'extraction de points caractéristiques, de codage de contours de Freeman, par psi-curves et par approximation polynomiale, de transformée de Hough, d'étiquetage de régions et de division/fusion.

A1.1 : Extraction de Points Caractéristiques.

L'algorithme présenté permet d'isoler et de stocker des points caractéristiques dans des tables différentes. Une table regroupe les points de l'image d'un même type, par exemple les points anguleux, les points d'un niveau de gris particulier... L'objectif est de distribuer les différentes tables sur les processeurs. Ainsi, l'algorithme se décompose en deux phases, une première dédiée à la recherche dans l'image des points caractéristique (P1). Les points caractéristiques de même nature sont regroupés dans une même table, ils sont donc envoyés sur le processeur gérant cette table. Ainsi les processeurs reçoivent les points associés à la table qu'ils gèrent lors de la seconde phase (P2).

Sur SYMPATI2, P1 s'exécute avec un parallélisme total en un temps dépendant et du nombre de types de points caractéristiques et de la complexité des types. A titre d'exemple s'il est nécessaire, pour la détection de 4 types différents, d'effectuer 4 convolutions 3×3, le temps d'exécution de P1 est de 20ms pour une solution à 32 processeurs sur une image de 256×256 points. Par contre, pour P2, le parallélisme de la structure est mal exploité car il est nécessaire de distribuer de manière asynchrone les différents points sur les processeurs adéquats. En effet, il n'est pas possible aux processeurs de SYMPATI2 d'effectuer entre eux, avec un total parallélisme, des échanges non réguliers.

Ainsi pour cette opération de traitement moyen niveau travaillant sur des points, des améliorations doivent être apportées au niveau du système d'intercommunication entres processeurs.

A1.2 : Codage de Freeman.

Le but de cet algorithme est de coder le ou les contours d'une image binaire. Chaque contour est codé de la manière suivante : les coordonnées du point de départ sont mémorisées,

ensuite tous les autres points de contours, pris successivement dans l'ordre du contour, sont examinés les uns après les autres et sont codés suivant leur direction par rapport au point précédent. Le codage se présente donc sous la forme d'une chaîne composée des coordonnées du point origine suivies d'autant de codes de direction qu'il y a de points dans le contour. Le codage le plus souvent utilisé code 8 directions possibles à partir d'un point : N, S, E, W, NE, NW, SE et SW (N : Nord, S : Sud, E : Est, W : ouest).

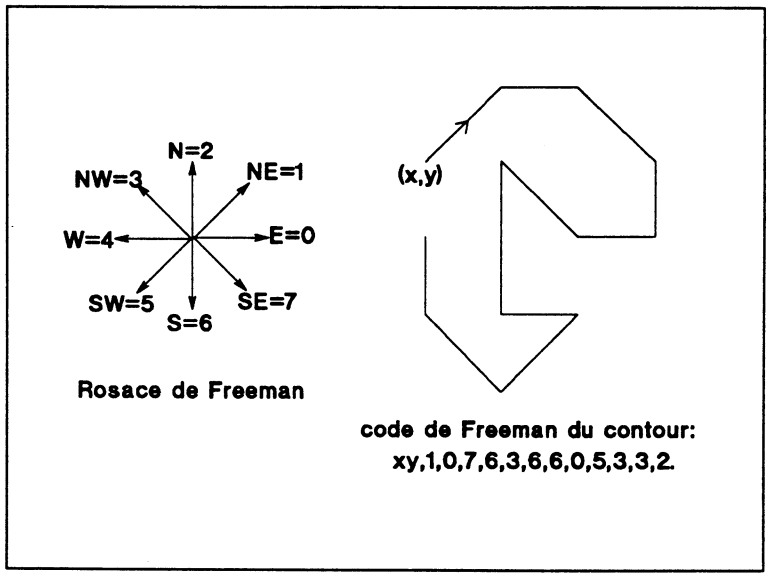

figure A.1.1 : Exemple de codage de Freeman suivant 8 directions.

L'intérêt d'un tel codage est qu'il fait abstraction de la position du contour dans l'image, si les coordonnées de départ sont ignorées, ce qui se prête très bien au "matching" de contours. De plus la dérivée de ce codage, facilement obtenue par modulo, est invariante en rotation. Mais cette méthode de codage est très sensible au bruit, une donnée erronée crée, sur toutes les autres données qui la succèdent, une erreur cumulative.

Comment paralléliser, sur une structure synchrone une telle opération? Nous proposons ici deux solutions. La première est de distribuer les processeurs sur l'image. Chacun des processeurs travaille sur une imagette. Ainsi l'algorithme est découpé en deux phases, la première (P1) transforme l'imagette binaire en une imagette contenant le codage. Pour cette première phase, tous les processeurs travaillent en parallèle sur leur imagette associée. Ensuite lors de la seconde phase (P2), les codes de Freeman sont extraits de l'image en séquentiel : un processeur commence à extraire les codes de l'un de ses contours jusqu'à ce qu'il arrive à une frontière. A ce moment, il passe la main au processeur avec lequel il partage la frontière, lequel continue à générer la liste de codes de Freeman du contour. Cette opération est à effectuer pour tous les contours. Dans cette solution le parallélisme n'existe

pas lors de la seconde phase, mais P2 peut s'avérer courte si le nombre de points de contours est faible. Au niveau de SYMPATI2, cette solution exige une autre répartition des points sur les processeurs, actuellement les images sont distribuées sur les processeurs alors qu'il est nécessaire de distribuer les processeurs sur l'image. De plus pour la seconde phase, les processeurs doivent être capables de se synchroniser entre eux, par passage de relais.

La seconde solution que nous proposons, consiste à paralléliser les traitements en distribuant les contours sur les différents processeurs. On parle alors de parallélisme de contours. Une fois les points d'un même contour distribués sur le même processeur, celui-ci compose le codage de ces points. Pendant ce temps les autres processeurs travaillent sur d'autres contours. Une première phase (P1) consiste donc à identifier les différents contours indépendants de l'image binaire, ensuite une seconde phase (P2) consiste à distribuer les points (coordonnées) des différents contours sur des processeurs différents. Enfin lors de la dernière phase (P3), les processeurs travaillent sur leur liste de points pour en extraire le code de Freeman du contour traité. Pour cette phase, il est nécessaire de disposer sur les processeurs et d'une UAL capable de travailler sur des coordonnées de points et d'un système d'adressage indexé. Cette méthode n'est totalement parallèle que si le nombre de contours est un multiple du nombre de processeurs, et que si les contours sont de même longueur, pour ainsi dire jamais. Pour effectuer la phase de distribution (P2), il est nécessaire de disposer d'un système d'intercommunication permettant aux processeurs d'effectuer des échanges asynchrones. Pour P3, la précision des processeurs de SYMPATI2 et le système d'adressage indexé sont bien adaptés.

A1.3 : Les "Psi-curves".

Cette opération de codage transforme un contour de points en une courbe s=f(Psi), s l'abscisse curviligne de chaque point et Psi l'angle de la tangente au contour par rapport à un axe donné. Cette opération est préféré à la signature polaire car elle est non équivoque.

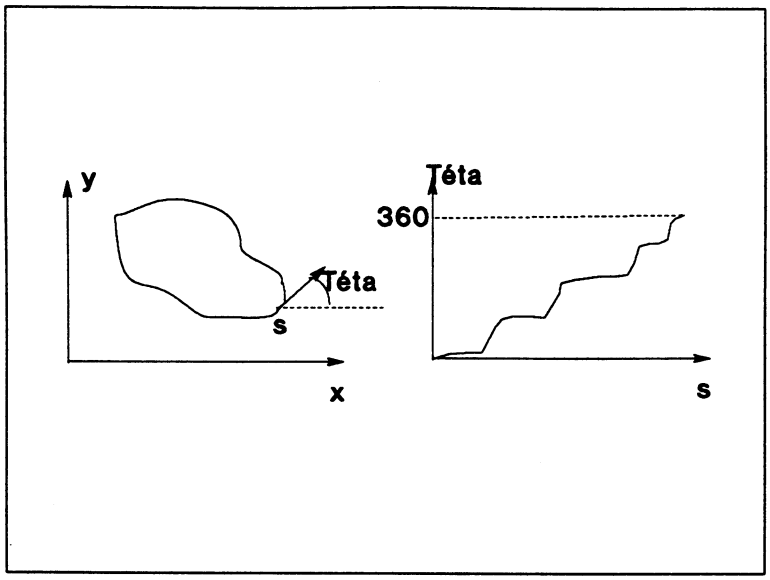

Figure A.1.2 : Transformation psi-curve Un contour fermé décrit tous les angles de 0 à 360 degrés.

La parallélisation d'une telle opération sur une structure SIMD peut être réalisée par distribution des contours sur les processeurs comme la seconde solution présentée pour paralléliser le codage de Freeman (c.f. A1.2). Ici également, le degré du parallélisme dépend du nombre de contours à transformer.

SYMPATI2 peut détecter les contours avec un parallélisme total mais ne peut distribuer efficacement ces contours sur les différents processeurs. Un système d'intercommunication plus évolué doit venir enrichir la structure actuelle si un tel algorithme doit être développé. Par contre la précision actuelle des processeurs s'avère suffisante [JUV83].

A1.4 : Approximation polynomiale par les moindres carrés.

L'objectif de ce codage est du trouver l'équation de la courbe qui approxime au mieux un ensemble de points consécutifs d'un contour. Deux approches mathématiques sont possibles, l'approche statique et l'approche dynamique [LAB88]. Dans l'approche statique, les coefficients (a_k) de la courbe y=a×x approximant k points sont donnés par :

$$ak = \frac{(\Sigma_{i=1}^{k} x_i y_i)}{(\Sigma_{i=1}^{k} x_i^2)}$$

Pour l'approche dynamique, (a_k) est déterminé de manière récursive par :

$$ak = a_{k-1} + K_k \quad (y_k \quad a_{k-1} \quad x_k);$$

$$Kk = \frac{P_{k-1}}{1+P_{k-1}\ x_k^2}$$

$$Pk = P_{k-1} - K_k \quad P_{k-1} \quad x_k \ .$$

L'approche dynamique est préférable à l'approche statique car l'évolution de l'erreur commise est contrôlable point à point ce qui permet de construire des points de contrôle (ou de cassure) judicieux. Toutefois, pour cette méthode, il est nécessaire d'initialiser la matrice de covariance P_0. Pour paralléliser cette opération dans sa version récursive deux méthodes sont proposées. Soit les points de contours, pris dans l'ordre du contour, sont chargés les uns après les autres dans les processeurs successifs, ensuite la structure travaille en SIMD systolique, ou bien les contours sont chargés dans des processeurs différents, les calculs s'effectuant alors en local (parallélisme de contours). Pour la première méthode, la phase critique sur SYMPATI2 est la distribution des points sur les processeurs. Par contre la structure actuelle permet de travailler en SIMD systolique où les processeurs effectuent les opérations en parallèle, en l'occurrence le calcul des a_k, k_k et p_k, ensuite ils envoient leurs résultats sur les processeurs voisins et une nouvelle phase de calcul peut alors s'effectuer pour l'indice (k+1). Pour cette solution, il est nécessaire d'avoir un contour avec un nombre de points conséquent et bien supérieur au nombre de processeurs afin de ne pas être handicapé par le temps de chargement du pipeline systolique. Actuellement, la distribution des points sur les processeurs ne peut s'effectuer correctement sur les processeurs, si bien qu'un système d'intercommunication plus évolué est nécessaire. Ce système d'intercommunication est également nécessaire pour la seconde méthode afin de distribuer correctement les contours sur les processeurs.

L'approche statique peut donner de bons résultats également si le nombre de points est faible. Ainsi, une solution astucieuse est de distribuer les points d'un contour par paquet de X (X faible, par exemple 10) consécutifs sur les processeurs. Chaque processeur effectue séparément l'approximation sur ses propres points, calcul de a_k. A la fin de ce calcul, chaque processeur compare son résultat avec celui de ses deux voisins gauches et droits, donc des deux segments connexes gauches et droits de X points. Si ces résultats sont semblables, par rapport à un seuil de tolérance, alors il y a fusion des 2X ou 3X points, sinon un point de cassure est établi. Pour cette approche également, un système d'intercommunication doit être proposé afin d'effectuer correctement la phase de distribution des X points sur les processeurs.

Pour toutes ces méthodes, deux autres problèmes sont à soulever. Le premier concerne la précision des processeurs. En effet, l'approximation par moindres carrés se fait sur des polynômes d'ordre choisi par l'utilisateur. Un ordre 1, approximation par des droites ne nécessite pas de flottant pour l'approche statique, voire même pour l'approche dynamique. Au delà de cet ordre, le flottant devient une nécessité mais ce cas de figure est assez rare en traitement d'images. La deuxième remarque concerne la phase pré-distribution des points sur les processeurs. Elle doit permettre soit d'identifier et de séparer les différents contours, soit de numéroter dans leur ordre tous les points d'un contour afin de les distribuer ensuite correctement sur les processeurs (un point par processeur pour l'approche dynamique, x points consécutifs par processeur pour l'approche statique). Dinstein et Landau proposent une méthode paralléle pour ce type de distribution dans [DIN90].

Notons que la méthode envisagée pour l'approche statique peut être utilisée pour l'algorithme d'approximation de contours par polylines. Dans cet algorithme, pour un ensemble de points, une droite est tracée entre les points extrêmes, ensuite la distance de tous les points par rapport à cette droite est calculée. Si une seule distance est supérieure à un seuil de tolérance précisé par l'utilisateur, alors le segment est cassé et l'opération réitérée. Ainsi chaque processeur peut recevoir X points d'un contour (X grand), et cette opération peut être effectuée en parallèle.

A1.5 : Le Transformée de Hough.

Le but de cet algorithme, fréquemment utilisé, est de détecter les courbes d'une image à partir de points de contours. Le type du contour à détecter doit être une courbe paramétrique, par exemple une ligne droite ou une courbe. L'intérêt de ce type de détection réside dans le fait que cette détection ignore les trous d'un contour et reste insensible au bruit.

Généralement, en traitement d'images, cette transformée détecte les lignes droites d'une image binarisée. Une droite est caractérisée par son équation $y=a\times x+b$. Pour tous les points de contour (x_i, y_i), on fait correspondre un ensemble de couple (a_j, b_j) déduit de $y_i = a_j \times x_i + b_j$ pour a_j compris de moins l'infini à plus l'infini. Tous ces couples obtenus sont alors stockés dans une table $a=f(b)$, initialisée à 0 au départ. A chaque couple (a_j, b_j) correspond une case qui est incrémentée à chaque occurrence de ce couple. A la fin de cette phase, la table présente des cases contenant un maximum local correspondant au couple (a_m, b_m) d'une droite représentative de l'image de contours.

Il est difficile de travailler avec la pente a qui varie de moins l'infini à plus l'infini si bien que deux solutions pallient à cet inconvénient. La première consiste à limiter de domaine de variation de a de -1 à + 1, remplir une première table avec de domaine, ensuite tourner l'image de 90°, les x deviennent les y et vice versa, et construire une deuxième table. L'autre solution très fréquemment utilisée consiste à travailler en coordonnées polaires. Une droite $y=a\times x+b$ est représentée par $\rho=x\times\cos\theta+y\times\sin\theta$. Dans ce cas, une simple passe est nécessaire pour θ variant de 0 à 2π. Les maxima (ρ,θ) obtenus en fin d'opération correspondent aux droites principales le l'image.

Qu'elle soit réalisée dans le plan cartésien ou polaire, cette opération est gourmande en calculs, car si θ prend 360 valeurs par exemple, l'équation $\rho=x\times\cos\theta+y\times\sin\theta$ est itérée 360 fois pour tous les points de contour. Ainsi il est nécessaire de paralléliser cette opération régulière. Principalement, deux méthodes existent, soit par parallélisme de données points images, dans ce cas chaque processeur calcule une table de Hough partielle, soit par parallélisme des paramètres et dans ce cas les processeurs travaillent sur des colonnes différentes de la table de Hough [CHO90].

Ainsi le parallélisation de cette opération s'effectue aisément sur une structure synchrone SIMD, car tous les processeurs effectuent la même opération sur des données différentes, mais le problème est de distribuer en début de calcul ces données (points frontière), sur les processeurs. Notons que les processeurs SIMD doivent être dotés d'un système d'adressage indexé leur permettant d'accéder à la case souhaitée de la table de Hough. Suivant la méthode utilisée, les points de contours détectés doivent être distribués sur tous les processeurs ou sur un seul processeur particulier. Mis à part ce problème d'intercommunication SYMPATI2 est adapté à cette transformation travaillant sur des droites. Notons que [BOU90], [DER89], [HUM86], [GUE88] et [RAN90] présentent la parallélisation de cette opération sur structures synchrones et asynchrones.

A1.6 : L'étiquetage de régions.

Le but de cet algorithme est d'identifier les différentes régions non connexes d'une image binaire. Tous les points doivent recevoir une étiquette correspondant au numéro de la région à laquelle ils appartiennent. Sur une structure séquentielle, tous les points d'une même région sont pris un à un et reçoivent le même numéro, ensuite une autre région est considérée, ainsi de suite. Une fois tous les points de région étiquetés, l'algorithme est terminé. Sur une structure séquentielle, la mise en oeuvre de cette fonction est aisée, par contre sa parallélisation sur une structure SIMD est complexe. Plusieurs solutions "parallèles" ont été

proposées, et s'appuient soit sur des algorithmes à traitements locaux (voisinage), soit sur des algorithmes à accès aléatoires (listes) [CYP90]. L'algorithme le plus fréquemment utilisé, à traitements locaux, s'appuie sur une série de balayages successifs suivant 4 directions sur l'image, nord-sud, est-ouest, sud-nord et ouest-est. Après une phase d'initialisation, ces balayages sont effectues sur l'image afin de propager les étiquettes. L'algorithme est terminé dès que les balayages ne modifient plus l'image. Le nombre de balayages nécessaires, à savoir le temps d'exécution, est tributaire des images, donc des données. Pour une image fortement concave, telle celle de la figure A.1.3, ce temps d'exécution est très long. Cette solution est actuellement utilisée sur SYMPATI2, la carte MATROX série et le calculateur CLIP4.

***Figure A.1.3** : Image de test pour l'opération d'étiquetage. La forte concavité de la figure exige un nombre de balayages importants pour les algorithmes locaux.*

Afin d'effectuer cette opération en un temps moins tributaire de la complexité des images, idéalement constant, d'autres approches sont envisageables. Citons le CAAPP (Content Adressable Array Parallel Processor) de l'université de Massachussetts, sur lequel cette opération s'effectue en un temps constant. Rappelons que le CAAPP est une matrice de processeurs où chaque point de l'image est associé à un processeur. Il y a alors autant de processeurs que de points d'image. Un réseau d'interconnexion appelé le "cotterie network" offre à chaque processeur la possibilité de se connecter avec ces quatre voisins. Les connections s'effectuent de façon électrique. Ainsi, pour l'opération d'étiquetage, chaque processeur scrute l'état de ces quatre voisins, et établit une liaison électrique avec ses voisins connexes. Une fois le réseau configuré à l'image de la connexité des points, la distribution des différentes étiquettes sur les région s'effectue par propagation électrique, donc en un temps constant et très rapide, 27 ms pour une image de 512×512 points [WEE89].

La méthode proposée sur SYMPATI2 actuellement ne nous convient pas, car son temps d'exécution est trop tributaire des données. A titre indicatif, l'étiquetage de la figure A.1.3 s'effectue en 53 passes soit 250 ms sur une configuration de SYMPATI2 avec 32 processeurs. Notons que cette opération sollicite pendant plusieurs secondes la carte Matrox série. De plus la puissance (entière 16 bits) des processeurs de SYMPATI2 est sous exploitée car chaque processeur travaille sur des nombres binaires, méthode bien adaptée aux processeurs "bit série" tel CLIP4. Ainsi au vu de la précision dont nous disposons, il est préférable de travailler sur des entités plus complexes intégrant des groupes de points connexes. Les "Run length Code" effectuent cette compression de points connexes [MAR90]. Ces entités codent les points connexes d'une même ligne ou d'une même colonne et sont composés des coordonnées de départ (ou de fin) du segment de points connexes ainsi que sa longueur (nombre de points connexes). Ensuite, les segments connexes reçoivent la même adresse, l'image complète se trouve ainsi étiquetée.

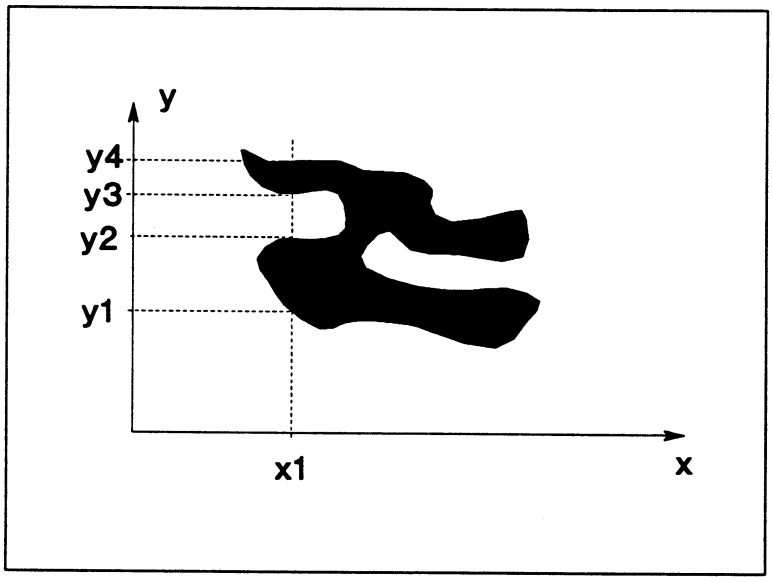

***Figure A.1.4** : Principe de détection et de formation des run length codes. Pour la colonne x1, deux segments sont détectés de longueur (y2-y1) et (y4-Y3).*

Dans un premier temps, il est nécessaire de détecter les "Run Length Code" dans une image binaire. L'algorithme écrit en 4LP, le langage de programmation de SYMPATI2, est exécuté en 5 ms pour une architecture à 32 processeurs sur une image de 256×256 points. Ce temps décroît linéairement avec le nombre de processeurs, et SYMPATI2 est parfaitement adapté à cette opération. Ensuite, comment exploiter ces segments?

Nous proposons ici deux solutions, l'une logicielle, l'autre matérielle. Pour la solution logicielle, nous proposons de traiter séquentielement les segments (run length) et d'utiliser la

structure parallèle pour effectuer les détections de segments connexes. Soit le nième segment d'un colonne j, S_{nj}, il ne peut être connexe aux autres segments de la même colonne, par principe de la génération des run length. Un segment connexe à S_{nj} ne peut être trouvé que sur les colonnes (j-1) ou (j+1). il est donc nécessaire de comparer entre eux tous les segments des colonnes précédentes et suivantes. On considère que les segments sont constitués par la longueur L et les coordonnées (X,Y) du point terminal du segment. Deux segments colonne S_{nj} et $S_{m(j-1)}$ sont connexes si et seulement si :

$$X(S_{m(j-1)}) - L(S_{m(j-1)}) <= X(S_{nj}) \qquad (1) \text{ et}$$

$$X(S_{m(j-1)}) => X(S_{nj}) - L(S_{nj}) \qquad (2)$$

sont vérifiées.

Ces deux opérations sont donc à répéter autant de fois qu'il y a de segments dans les colonnes (j-1) et (J+1). Ces donc à ce niveau que le parallélisme existe. Les opérations s'effectuent de la façon suivante. Un segment, appelé le segment courant, est pris en considération, il est présenté aux autres processeurs qui possèdent les k segments des colonnes précédentes et suivantes. Chacun des k processeurs effectue alors (1) et (2), les segments connexes sont ainsi déterminés en parallèle, reçoivent l'étiquette du segment courant et sont envoyés dans une liste d'attente de segment à traiter. Les segments de cette liste sont ainsi pris un à un et une fois la liste vide, une autre région, donc étiquette, est appliquée sur d'autres segments.

Au niveau de l'architecture, il est nécessaire de distribuer les l segments d'une colonne sur les l premiers processeurs de SYMPATI. Cette opération est effectuée pour toute les colonnes de l'image. Ensuite, la détermination des segments connexes est à effectuer, les caractéristiques du segment courant sont envoyés vers les k processeurs afin qu'ils puissent effectuer les calculs (1) et (2) sur leurs segments respectifs, appartenant soit à la colonne précédente, soit à la suivante. Les segments connexes sont étiquetés, puis envoyés dans une liste d'attente.

Pour effectuer les calculs (1) et (2), la précision actuelle des processeurs de SYMPATI2 est suffisante. Par contre, il est nécessaire de disposer d'un système d'intercommunication permettant d'effectuer tous les échanges présentés, à savoir la distribution des l segments obtenus à partir d'une colonne sur les l premiers processeurs et la globalisation d'un résultat pour la distribution, vers les k processeurs, des caractéristiques du segment courant. De plus

un des processeurs doit être capable de prendre le contrôle du bus afin de présenter son segment courant aux autres processeurs.

Le temps d'exécution de cet algorithme est proportionnel au nombre de segments (run length) détectés. Donc ce temps reste tributaire de l'image mais l'est bien moins que pour la solution à balayages successifs proposée actuellement. Par contre le nombre de processeurs au delà duquel le temps d'exécution reste constant est égale au nombre maximal de segments pour une colonne multiplié par 2. Une solution à 256 processeurs risque de présenter les mêmes performances qu'une solution à 32 processeurs! Pour paralléliser davantage cette opération sur une architecture possédant de nombreux processeurs, nous proposons de découper l'image de départ en imagettes non, ou faiblement, connexes, imagettes distribuées sur des groupes de processeurs et de tailles différentes. Pour effectuer le découpage, il est nécessaire de tracer la courbe du nombre de segments obtenus par colonne (nbS) en fonction du numéro des colonnes (numCol), nbS=f(numCol). Ensuite il est nécessaire de trouver les minima de cette courbe. Idéalement les minima passant par 0 permettent de découper l'image en imagettes complètement indépendantes.

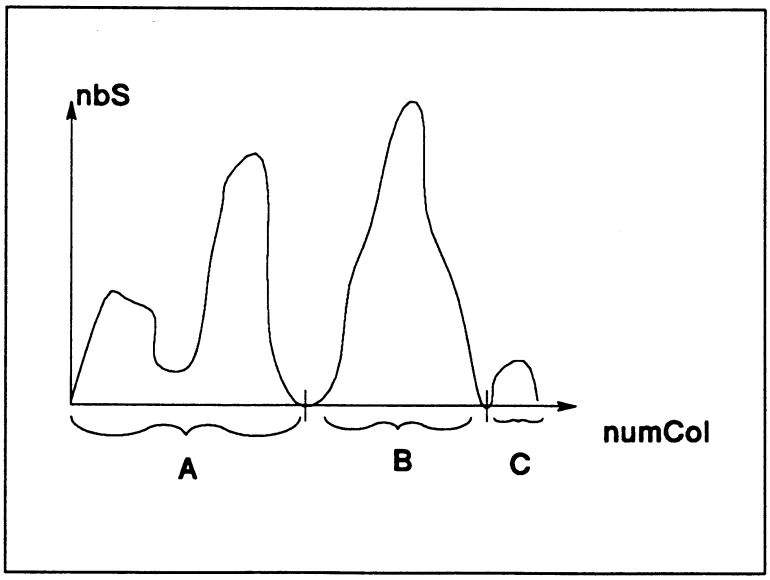

***Figure A.1.5** : Exemple de courbe nbS=f(numCol). Les zones A, B et C sont indépendantes et peuvent donc être étiquetées en parallèle.*

A chaque découpage, le barreau de processeurs doit être scindé virtuellement en 2 parties travaillant sur les imagettes. Le nombre de processeurs doit toutefois rester supérieur à deux fois le nombre de segments maximum (pour une colonne) obtenu pour l'imagette de travail, ce maximum est obtenu sur la courbe nbCol=f(numCol). Une fois ce découpage

obtenu, les segments sont distribués sur les processeurs, distribution tributaire du découpage, tous les processeurs travaillent sur leur imagette associée en mode SIMD. Si le découpage n'a pu se faire par des passages par zéro, une phase de mise à jour est nécessaire en fin d'exécution, phase minimisée car le découpage s'est fait en imagettes faiblement connexes. Plus les minima de découpage sont proches de zéro, plus cette phase est courte.

Notons enfin que pour obtenir le meilleur découpage possible, le programmeur peut essayer de travailler sur les deux courbes, nombre de segments de type colonne en fonction du numéro des colonnes et nombre de segments de type ligne en fonction du numéro des lignes, le meilleur découpage étant retenu.

La seconde solution pour effectuer cette opération d'étiquetage, basée sur une approche matérielle s'inspire de la méthode utilisée sur le coterie network du CAAPP. Une première phase exécutée sur SYMPATI2 permet, comme précédemment, de détecter les run length codes (ou les segments de points connexes) sur les colonnes. Ensuite tous les segments sont chargés sur un circuit reprogrammable. Ce circuit doit être composé d'une matrice de cellules reprogrammables. En fonction des segments reçus, les cellules s'interconnectent les unes aux autres s'il y a connexité. Chacune des cellules représente soit un segment, soit un point de l'image. Dans le premier cas le nombre de cellule est faible, égale au nombre de segments, mais l'interconnexion, à savoir le routage, peut être complexe, car un segment peut devoir être interconnecté à bien plus de 4 voisins. Pour la seconde solution, l'image est redessinée sur la matrice de cellules à partir des segments, l'interconnexion entre cellules s'établit simplement, le nombre d'interconnexions à établir par voisins n'est que de 4 au maximum, par contre il doit y avoir autant de cellules que de points dans l'image, et pour retourner vers les processeurs l'information doit être recompressée, sous forme de segments également, mais cette fois étiquetés. Notons qu'il est possible là également de travailler sur des imagettes si la matrice de cellules ne peut représenter directement l'image complète. Parmi les circuits reprogrammables du marché, les produits XILINX et PLESSEY [XIL92, PLE90], se prêtent bien à la première solution bien qu'il faille se méfier des possibilités de routage de ces circuits, les produits de Algotronix [ALG90] proposent des matrices de cellules très simples et permettant les connexions directes avec les voisins Nord, Sud, Est et Ouest et sont donc mieux adaptés à la seconde solution.

L'intérêt de ces deux méthodes est qu'elles offrent la possibilité d'effectuer cette opération d'étiquetage en un temps constant mais au prix d'un matériel plus lourd. Une telle solution n'est envisageable sur SYMPATI2 que si un système d'entrées/sorties des résultats obtenus sur les processeurs vient compléter la structure actuelle.

A1.7 : Division/Fusion.

Dans cette approche, il s'agit également de regrouper entre eux des points connexes mais ici le critère de connexité est défini par l'utilisateur et sur des images en niveaux de gris.

Notons qu'à l'origine, il n'existait que des algorithmes de division ou de fusion, et ce n'est que dans les années 70 que Robertson et Klinger ont proposé un mélange de ces 2 techniques ce qui réduit le temps d'exécution sans augmenter l'espace mémoire nécessaire.

Pour un critère défini dans une fonction "f", deux régions A et B sont considérées homogènes si f(A)-f(B) < T, avec T le seuil fixé par l'utilisateur, seuil à ne pas dépasser sur une région homogène. Une région R est composée d'un ensemble de points x_i, sur cette région une fonction d'homogénéité est définie telle que :

$$H(R) \begin{cases} \text{- Vraie si pour tout (xi,xj), on a f(xi)-f(xj) < T;} \\ \text{- Faux autrement.} \end{cases}$$

L'algorithme de fusion consiste, à partir d'une image complète, à identifier toutes les régions homogènes. Les différentes phases de cet algorithme sont les suivantes :

1 : Pour une région R, si H(R) est faux alors la diviser en 4 parties égales, puis reprendre ces 4 régions obtenues et calculer leur H(R), re-diviser si faux etc jusqu'à ce que l'on n'est que des régions homogènes. Puis pour chaque quarté de régions homogènes voisines de même taille R1, R2, R3 et R4, calculer H(R1UR2UR3UR4) et fusionner ce quarté si H est vraie.

2 : Ensuite pour 2 régions R1 et R2 non forcément de même taille calculer H(R1UR2) et les fusionner si H est vraie.

L'ensemble de ces divisions fusions est représenté sous la forme d'un arbre quaternaire appelé "Quad Tree". Les informations de connexité de régions et de parenté apparaissent sur cet arbre.

En ce qui concerne la mise en oeuvre d'un tel algorithme sur une structure parallèle, M. Celenk et P. Lakshman proposent dans [CEL89] la parallélisation de cette opération de division/fusion ou "split and merge" sur une architecture hyperbolique à 16 noeuds. L'image est découpée de façon régulière en imagettes distribuées sur les processeurs. Cette méthode convient pour des architectures faiblement parallèles, mais présente des problèmes pour les

architectures massivement parallèles où il y a autant de processeurs que points images. Pour les structures moyennement parallèles, tel SYMPATI2, ce type de découpage est envisageable et préférable au mode de distribution actuel (hélicoïdal). Chaque processeur travaille indépendamment sur son imagette, ensuite les sous-arbres obtenus sont regroupés et des fusions peuvent encore intervenir. Les mises en oeuvre de cette opération sur la structure pyramidale SPHINX et sur un réseau en "ferme de processeurs" sont présentées dans respectivement [CLE90] et [CHAb90].

En ce qui concerne la mise en oeuvre de cet algorithme sur SYMPATI2, il est nécessaire de disposer d'un mode de chargement différent permettant de stocker une imagette par processeur, imagettes rectangulaire ou carrées. Une fois cette distribution réalisée, les processeurs travaillent de façon autonome en mode SIMD sur leur imagette respectives. Une limite intervient à ce niveau, c'est la précision des processeurs de SYMPATI2. Pour certains critères (f), cette précision (16 bits) est suffisante, pour d'autres très complexes un autre format est nécessaire. Enfin, il faut regrouper tous les sous-arbres et fusionner les régions pouvant l'être. Un découpage de l'image en bandes est préférable à un découpage par carrés car dans ce premier cas, seules deux frontières sont à considérer, contre 4 pour la seconde solution. De plus 3 bandes voisines sont réparties sur 3 processeurs voisins donc les 2 frontière se gèrent très bien. En fin d'opération, l'arbre quaternaire complet doit être restitué, ce qui demande un système permettant de restituer rapidement des résultats de type non iconique. Pour gérer ces arbres, les processeurs doivent disposer d'un mode d'adressage indexé ce qui est le cas des processeurs de SYMPATI2.

Annexe A2 :

Détail des Entrées/Sorties de la Cellule.

Cette annexe présente en détails la structure du modèle VHDL de la cellule.

Tout d'abord, ce circuit est présenté sous forme d'une boîte noire qui correspond à l'entité du modèle VHDL de la cellule. Les entrées de cette entité sont les suivantes

- vqclk : horloge de cadencement des décalages;
- wri : signal d'écriture d'un message dans la cellule;
- oe : signal de lecture d'une donnée de la FIFO;
- ucin : signaux de contrôle venant de l'unité de commande (6bits);
- din : champ donnée du message issu de la cellule voisine (16 bits);
- adrin : champ adresse du message issu de la cellule voisine(16 bits);
- numpein : champ numéro processeur destinataire du message issu de la cellule voisine(8 bits);
- controlin : champ contrôle du message issu de la cellule voisine(4 bits);
- datapein : bus de données du processeur associé(8 bits);
- adrpein : bus adresse du processeur associé (16 bits);
- ma_cc_in : signal permettant le passage de priorité pour l'accès au réseau en mode par diffusion électrique. Signal géré par daisy-chain.

et les sorties :

- dcout : signaux de retours vers l'unité de commande en daisy chain vers la cellule voisine(3 bits);
- dout : champ données du message vers la cellule voisine (16 bits);

• adrout : champ adresse du message vers la cellule voisine(16 bits);

• numpeout : champ numéro processeur destinataire du message vers la cellule voisine(8 bits);

• controlout : champ contrôle du message vers la cellule voisine(4 bits);

• datapeout : bus de données du processeur associé(8 bits);

• adrmemout : bus adresse véhiculant l'adresse du DMA(16 bits);

• wrmem : signal d'écriture dans la mémoire associée en mode DMA;

• ma_cc_out : sortie du daisy_chain effectué avec le signal d'entrée signal ma_cc_in à l'intérieur d'une cellule.

Le total des entrées et des sorties est important, mais en réalisation finale, des bus entrées pourront être combinés avec des bus de sorties.

Les données envoyées par l'unité de commandes sur les cellules, sont les suivantes :

• DA : Décalage Autorisé - ucin(0) -;

• SC : l'unité de commande demande aux cellules de se court-circuiter afin de réaliser l'opération de diffusion de données par propagation électrique - ucin(1) -;

• vqclk : l'horloge de décalage est envoyée sur ce fil- ucin(2) -;

• heltab : l'unité de commande indique aux cellules, via ce fil, si les processeurs travaillent en mode d'adressage hélicoïdal ou tabulaire - ucin(3) -;

• DMA_OK : l'unité de commande indique aux cellules si elles peuvent effectuer une écriture en DMA - ucin(4) -;

• adres_exist : l'unité de commande indique aux cellules que le message qu'elles vont recevoir sera à stocker en DMA dans le processeur cible - ucin(5) -;

En retour, la cellule envoie vers l'unité de commande des informations concernant l'état de sa FIFO ainsi que la validité du message. Ces sorties sont de type collecteur ouvert et sont connectées ensembles sur le même bus, bus lu par l'unité de commande. Ces informations sont les suivantes :

• FRP : FIFO de réception pleine - dcout(0)-;

• FRV : FIFO réception vide - dcout(1)-;

• DL : Les données de l'ensemble des messages présent dans les cellules sont lues. La distribution est alors terminée - dcout(0)-.

Le modèle de la cellule est organisé autour de différents processus, 9 au total, dédiés à l'écriture de messages dans le réseau, à la récupération de données contenues dans les messages, à la gestion de la FIFO, du DMA etc. Le modèle comportementale de la cellule se présente de la manière suivante:

```
LIBRARY MGC_portable;
use MGC_portable.qsim_logic.all;
USE work.pe_pack.ALL;

ENTITY cellule IS
PORT   (wr,oe          :in    qsim_state        :='0';
        ucin           :in    datapath          :=a_zero;
        dcout          :out   numreg_bit;
        din            :in    donnee_16bits     :=a_zero16;
        dout           :out   donnee_16bits;
        adrin          :in    donnee_16bits     :=a_zero16;
        adrout         :out   donnee_16bits;
        numpein        :in    datapath          := a_zero;
        numpeout       :out   datapath;
        controlin      :in    numreg_bit        :=('0','0','0','0');
        controlout     :out   numreg_bit;
        datapeout      :out   datapath;
        datapein       :in    datapath          :=a_zero;
        adrpein        :in    donnee_16bits     :=a_zero16;
        adrmemout      :out   donnee_16bits;
        ma_cc_in       :in    qsim_state        :='0';
        ma_cc_out      :out   qsim_state;
        wrmem          :out   qsim_state);
END cellule;
```

```
ARCHITECTURE top_level OF cellule IS

---- déclaration des signaux internes à la cellule

BEGIN

---- affectations concurrentes

---- définition des processus

ecriture_reseau :process
---- processus d'écriture d'un message dans le réseau
end process ecriture_reseau;

recup_data :process
---- processus dédié à la récupération des messages destinés à la cellule associée.
end process recup_data;

lecture_fifo:process
---- processus dépilant mis en oeuvre lors de la lecture de la fifo.
end process lecture_fifo;

reso_mes :process
---- processus dédié à la résolution de signaux
end process reso_mes;

affec_cpt:process(inc,dec)
---- processus gérant le mécanisme de la FIFO
end process affec_cpt;

recup_x_y:process
---- processus calculant les coordonnées (x,y) du point courant.
end process recup_x_y;

daisy_chain: process
---- processus gérant le daisy-chain.
end process daisy_chain;

---- .... puis, un ensemble de processus de concaténation/déconcaténation de signaux.

END top_level;
```

L'ensemble de ce modèle représente 320 lignes de VHDL, et 50 signaux internes.

INPG département des études doctorales

AUTORISATION de SOUTENANCE

VU les dispositions de l'Arrêté du 23 novembre 1988 relatif aux Etudes doctorales

VU les rapports de présentation de

. Monsieur J. GALLICE , Professeur
. Monsieur C. LANDRAULT , Directeur Recherches C.N.R.S.

Monsieur COLLETTE 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 "

Fait à Grenoble, le 31 août 1992

Maurice RENAUD
Président
de l'Institut National Polytechnique
de Grenoble