Conception et réalisation d'une machine parallèle de codage d'images à bas débit sur un réseau de transputers

Mactar Seck

Université de Metz — U.E.R. Mathématique Informatique et Mécanique

Laboratoire de Mécatronique Industrielle - I.S.G.M.P.

THESE

CONCEPTION ET REALISATION D'UNE MACHINE PARALLELE DE CODAGE D'IMAGES A BAS DEBIT SUR UN RESEAU DE TRANSPUTERS

Date de soutenance: 10 Décembre 1993
pour obtenir le titre de
DOCTEUR DE L'UNIVERSITE DE METZ
Spécialité : ELECTRONIQUE

par

Mactar SECK

Composition du jury:

Président	: M. A. TOSSER-ROUSSEY	ENI- Metz
Rapporteurs	: M. L. LEGRAND	Université de Dijon
	: M. E. YVROUD	CNRS-ENSEM-INPL-Nancy
Examinateurs	: M. C. LHERMITTE	Supélec - Metz
	: M. A. PRUSKI	Université de Metz
	: M. F. ROYER	Université de Metz
Invités	: M. P. BOURCET	Chef du Département T.S. TDF-C2R
	: M. S. VIALLE	Supélec-Metz

TDF
Groupe France Telecom

● Centre d'études en Radiodiffusion
et Radiocommunications

C2R/TS/MS/212/93

Les dispositifs technologiques et les cartes prototypes présentées dans ce mémoire sont cités dans deux demandes de brevets d'inventions, dont les intitulés sont:

- " **Un Système de traitement d'images et son application à la compression-décompression des ces images ou à un dispositif d'autofocalisation pour caméra vidéo**"
N° de dépôt 931201 au nom de TéléDiffusion de France.

- " **Un quantificateur vectoriel intégré bit série**" (Dépôt en cours).

TéléDiffusion de France
TDF-C2R : 1, rue Marconi - Technopôle Metz 2000 - 57070 Metz - Tél. (33) 87 20 75 00 - Fax (33) 87 76 33 63
Siège social : 10, rue d'Oradour-sur-Glane - 75732 Paris Cedex 15
Société Anonyme au Capital de 1 043 478 200 F - RCS Paris B 342 404 399

REMERCIEMENTS

Le travail présenté dans ce mémoire résulte de la collaboration entre le laboratoire de Mécatronique Industrielle de l'ENIM (Ecole Nationale d'Ingénieurs de Metz) et TDF-C2R Groupe France Télécom (Centre d'études en Radiodiffusion et Radiocommunications de TéléDiffusion de France).

J'exprime ma plus profonde gratitude à Monsieur A. TOSSER-ROUSSEY, Professeur à l'ENIM, pour la confiance qu'il m'a accordée en m'accueillant dans son laboratoire et pour son soutien constant au cours de mes travaux.

Sincères remerciements à Monsieur Arnaud LAPREVOTE responsable du laboratoire d'Electronique Numérique et Réseaux de Neurones TDF-C2R, pour son encadrement industriel , pour les nombreux conseils qu'il m'a prodigués tout au long de ce travail et pour la grande disponibilité dont il a fait preuve à mon égard.

Je suis très reconnaissant envers Messieurs L. LEGRAND, Maître de Conférence à l'Université de Dijon et E. YVROUD, Directeur de Recherche CNRS , Professeur à l'ENSEM (Ecole Nationale Supérieur d'Electricité et de Mécanique) de Nancy pour avoir accepté d'être rapporteurs de thèse et pour l'intérêt qu'ils ont porté à ce travail et les nombreuses suggestions données lors de nos différents entretiens.

Je tiens à remercier Messieurs C. LHERMITE, Professeur, Chef du service Informatique à l'ESE (Ecole Supérieure d'Electricité) de Metz, S. VIALLE, Ingénieur Enseignant à l'ESE, A. PRUSKI, professeur des Universités et F. ROYER, Maître de Conférence à l'Université de Metz, pour l'attention qu'ils ont bien voulu porter à ce travail et pour l'honneur qu'ils me font de participer à ce jury.

Je tiens à remercier Monsieur Patrice BOURCET, Chef du Département Traitement du Signal TDF-C2R, pour son efficacité doublée de bonne humeur et pour l'environnement matériel et scientifique de qualité mis à ma disposition pour réaliser ce travail.

Que l'ensemble du personnel de TDF-C2R en particulier ceux du Département Traitement du Signal et le personnel féminin trouvent ici l'expression de mes remerciements chaleureux pour la collaboration efficace et pour l'ambiance amicale et sympathique qui a toujours régné entre nous.

A ma famille, à la famille SAKHO de Vandoeuvre-les-Nancy, à la communauté sénégalaise de Lorraine et à tous les anonymes qui par leurs conseils, leurs critiques pertinentes, leur soutien perpétuel ont contribué directement ou indirectement à la réalisation de ce travail, qu'il trouvent ici le témoignage de ma profonde considération.

Sommaire

Introduction

Chapitre I - Formalismes du Parallélisme

Chapitre II LE TRAITEMENT PARALLELE DE L'IMAGE

Chapitre III REALISATION MATERIELLE

Cette thèse est une contribution au traitement d'images en parallèle ou à la parallélisation d'algorithmes de traitement du signal. On attend de l'étude d'un point d'intersection entre le traitement d'images et le parallélisme qu'elle permette d'aboutir à des résultats efficaces et nouveaux, compte tenu des possibilités technologiques actuelles.

Objectifs

Le but de notre étude est de réaliser un prototype de machine de codage d'images fixes à faible coût, destinée à transmettre des images de basse qualité sur un canal à très faible débit. La partie matérielle réalisée ne concerne que le codage source. Une architecture parallèle est retenue.
Dans un premier temps, l'algorithme de codage mis au point à TDF-C2R, basé sur la TCD (Transformée en Cosinus Discréte) et la quantification vectorielle dans le domaine transformé, est à modifier dans le but d'augmenter le taux de compression et d'améliorer la qualité des images restituées. Un codage d'Huffmann sera utilisé. L'architecture parallèle est ensuite à programmer avec l'algorithme adéquat et le remplacement des parties logicielles critiques par des cartes spécialisées sera envisagé.

1 Nécessité d'une architecture parallèle

En traitement d'images, la quantité de données à manipuler et la complexité des calculs souvent répétitifs font appel à une puissance de calcul importante, notamment lorsque le temps réel est requis.

La taille d'une image noir et blanc de 512x512 pixels sur 256 niveaux de gris est de 256 Koctets. Il faut 32 secondes pour la transmettre sur un canal du Réseau Numérique à Intégration de Services. Pour les séquences animées la quantité d'informations est multipliée par le nombre d'images à acquérir par seconde.

Tout système de transmission d'images doit être en mesure d'acquérir, de traiter, de transmettre et de visualiser des séquences d'images numérisées en un temps limité. Les contraintes au temps réel varient avec le domaine d'application. Pour un service de télévision 25 images par seconde sont nécessaires, par contre pour le visiophone ou la télésurveillance, quelques images suffisent (entre 10 et 15 images par seconde).

Le traitement d'images est sans aucun doute un domaine privilégié pour l'étude et la mise en oeuvre du parallélisme. En effet les systèmes de transmission d'images nécessitent une puissance de calcul justifiant l'utilisation de machines parallèles.

Le parallélisme permet de concevoir des systèmes utilisant plusieurs processeurs spécialisés, travaillant de façon autonome mais pouvant échanger des informations avec les autres processeurs. L'architecture définit les liens entre les différents processeurs. Les architectures parallèles permettent d'accélérer le traitement. A cette grande vitesse d'exécution, vient s'ajouter un gain conceptuel car on peut résoudre certains problèmes qui étaient jusque là impossibles à mettre en oeuvre sur machine séquentielle.

2 Les possibilités technologiques actuelles

Les performances se sont améliorées au fil des années par:

- l'amélioration technologique (doublement du nombre des transistors sur une puce et amélioration de la fréquence d'horloge tous les 18 mois)[BUH-92],
- l'adoption de tampons mémoire très rapides à tous les niveaux dans l'unité centrale (mémoire cache interne), entre la mémoire et/ou les entrées/sorties et l'unité centrale. Ceci a permis d'augmenter le débit d'informations efficace entre l'unité centrale, les périphériques et la mémoire centrale,
- l'augmentation du nombre des fonctions de chaque composant (intégration des opérations en virgule flottante dans l'U.C., écriture directe des E/S dans la mémoire, intégration de fonctions graphiques complexes dans les composants vidéo, ...),
- des améliorations architecturales (dessin de circuits mémoires très denses, architectures RISC, pipe-lining, architectures vectorielles, ...).

3 Chronologie de notre étude

Nous allons donc nous intéresser dans ce travail à la conception et à la réalisation d'une machine parallèle à base de transputers destinée à la transmission d'images vidéo à bas débit.

Le travail sera divisé en trois chapitres comme suit.

Le premier chapitre est un exposé introductif sur le parallélisme. Il présente les différents aspects du parallélisme qui seront abordés tout au long des différents chapitres de la thèse, l'accent étant mis sur les architectures parallèles.

Dans un second chapitre, à travers un algorithme de codage d'images, nous allons montrer la nécessité du parallélisme en traitement d'images. Puis nous allons mettre au point plusieurs techniques de parallélisation en vue d'accélérer le temps de traitement des algorithmes de codage d'images. Nous proposons une technique de parallélisation par les données sur machine

MIMD (transputer). Ce processeur sera amplement décrit en annexe. Cette partie comporte plusieurs résultats expérimentaux.

Enfin dans le dernier chapitre, le plus long, nous allons nous intéresser notamment à la réalisation matérielle de la machine parallèle de codage pour valider notre approche logicielle. Des cartes prototypes spécialisées seront mises au point pour remplacer les parties logicielles critiques .

Puis nous conclurons sur la complexité intrinsèque du problème et l'évolution de la machine.

Chapitre I

FORMALISMES DU PARALLELISME

1 - Introduction

Le parallélisme permet de concevoir des systèmes utilisant plusieurs processeurs spécialisés, travaillant de façon autonome mais pouvant échanger des informations avec les autres processeurs. L'architecture définit les liens entre les différents processeurs.

Le parallélisme est devenu indispensable au développement de l'informatique, car les besoins en calcul numérique ne cessent de croître et de nombreux traitements deviennent de plus en plus complexes donc grands consommateurs en temps de calcul.

Les architectures parallèles permettent d'accélérer le traitement. A cette grande vitesse d'exécution, vient s'ajouter un gain conceptuel car on peut résoudre certains problèmes qui étaient jusque là impossibles à mettre en œuvre sur machines séquentielles.

Les systèmes parallèles sont arrivés à maturité à partir des années soixante dix. Leurs performances ne cessent de croître depuis. L'évolution des processeurs RISC a permis le développement rapide de machines parallèles très efficaces. Et actuellement on a tendance à utiliser des processeurs élémentaires de plus en plus puissants et des réseaux de connexions performants [CAT1-86].

2 - Mécanismes de parallélisation

Le parallélisme est caractérisé par l'emploi de plusieurs unités de traitement distinctes qui accomplissent en concurrence le même travail ou des travaux différents. Afin de disposer du maximum d'avantages du parallélisme, il faut tenir compte pour chaque application de la quantité de données qu'elle contient et de la manière dont les traitements sont organisés. Ainsi nous distinguons trois sortes de parallélisme:

- le parallélisme vrai,

- le pipelining,

- le faux parallélisme.

2.1 Le vrai parallélisme

Le parallélisme vrai ou parallélisme concurrent implique que les traitements soient indépendants les uns des autres pour pouvoir s'exécuter simultanément sur des unités distinctes. Les unités peuvent échanger des données et regrouper des résultats.

Une application est composée de tâches ou processus qui peuvent être exécutés de manière plus ou moins indépendante sur des ressources de calculs appelées processeurs.

Un processeur contient généralement une unité de calcul, une unité de contrôle et une mémoire qui contient les programmes et les données.
Une tâche se caractérise en général par :

- la réception des données d'entrée de ses prédécesseurs,
- le traitement de ces données,
- l'envoi des données traitées vers ses successeurs.

Par exemple considérons une machine à n processeurs P1, P2, ..., Pn et une application composée de n tâches T1, T2, ..., Tn. Deux cas sont possibles:

- ***Indépendance des tâches***(Fig. 1): si les tâches sont indépendantes, il suffit d'associer un processeur à chacune d'entre elles pour obtenir un gain en temps de traitement qui est linéaire. N tâches indépendantes s'exécuteront N fois plus vite sur N processeurs. Ce cas est idéal [CES-91].

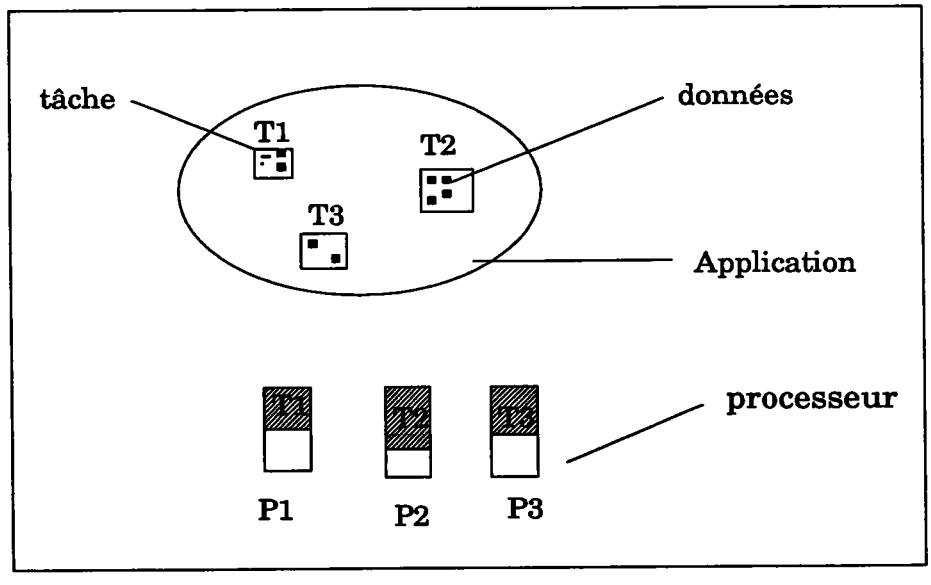


Fig. 1: Indépendance des tâches sur une machine à 3 processeurs

- ***Dépendance des tâches*** (Fig.2): en général les tâches des applications réelles sont souvent dépendantes:
  - du séquencement de l'algorithme,
  - des communications.

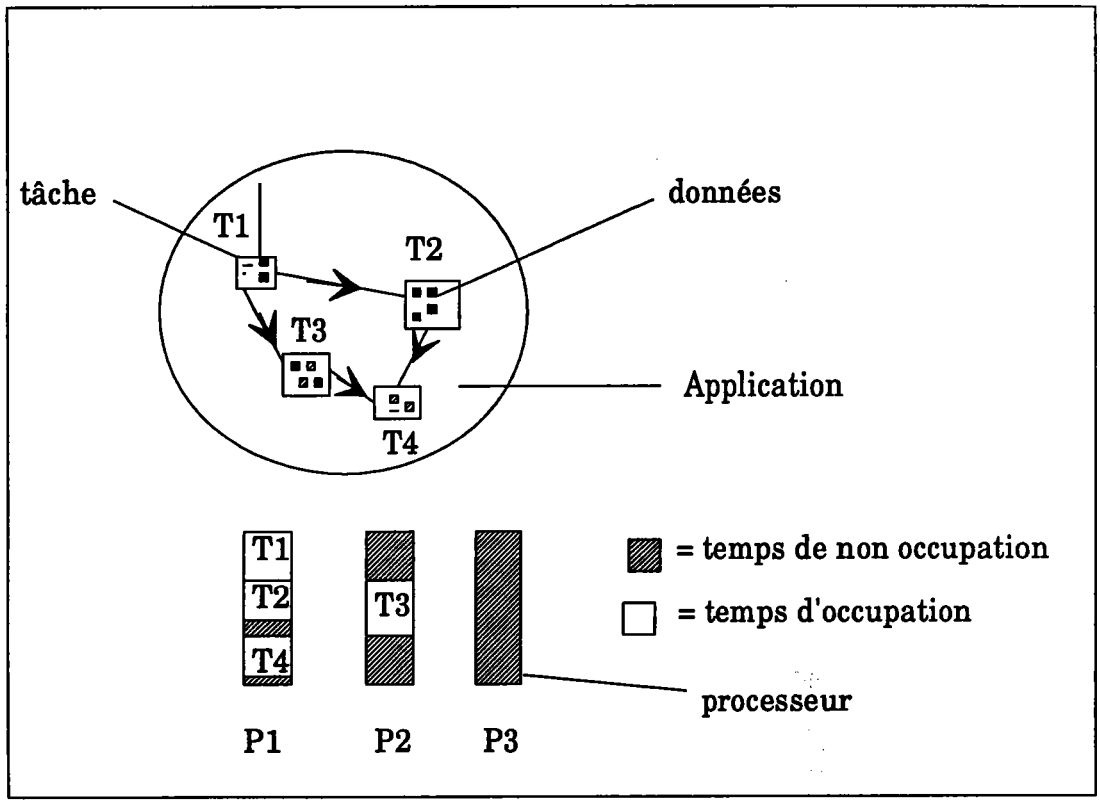


Fig. 2 : Dépendance des tâches sur une machine à 3 processeurs.

Dans ce cas certains processeurs restent inoccupés (P3) ou en attente (P1, P2).

2.2 Le pipelining

La chaîne de traitement est divisée en plusieurs étapes, chaque étape est traitée par un ou plusieurs processeurs différents (Fig.4).

Le pipelining permet de diviser la capacité de traitement par le nombre de processeurs du système à condition que le taux d'activité de chaque processeur soit maximal. L'inconvénient est que, lors du démarrage ou à la fin, certains processeurs doivent attendre que d'autres processeurs aient terminé.

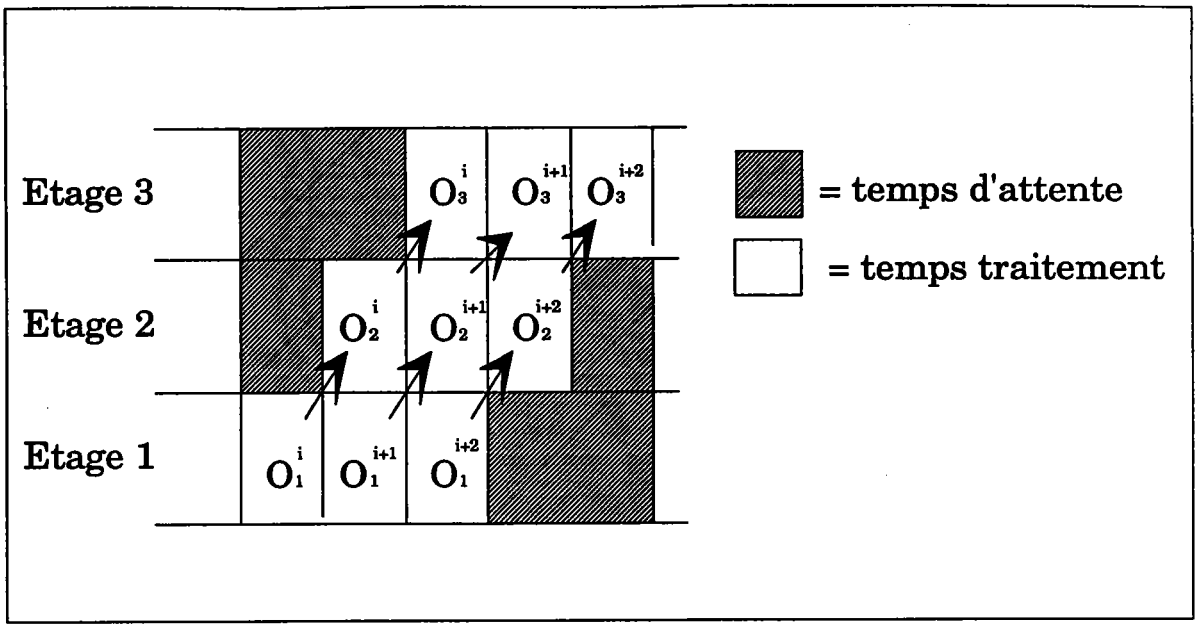


Fig.3: Exécution des opérations O_1 , O_2 , O_3 sur des données i

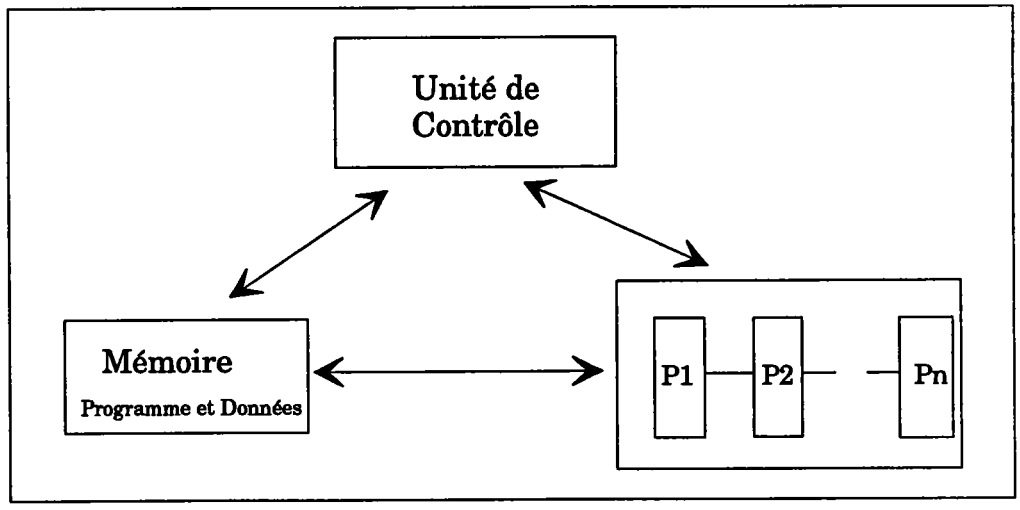


Fig 4: Structure d'un pipe-line

Le principe du pipelining est aujourd'hui appliqué par la plupart des microprocesseurs Intel(80486) et a été pleinement exploité pour la construction de super-ordinateurs de la classe pipeline comme le CRAY X-MP [CRA-92].

Le parallélisme concurrent et le pipelining peuvent coexister et être mis en oeuvre à différents niveaux.

2.3 Le faux parallélisme

Le faux parallélisme[GAE-90] est une illusion de parallélisme créée par des processeurs sériels programmés comme s'ils effectuaient un traitement parallèle. C'est le cas des systèmes d'exploitation multitâche. Un système d'exploitation multitâche est muni d'une horloge qui génère des interruptions périodiques, ce qui permet à chacune des tâches d'utiliser le processeur pendant une période. Lorsqu'une tâche attend une entrée de la part de l'utilisateur ou un transfert d'entrées/sorties, le processeur en profite pour exécuter une ou plusieurs autres tâches. Un pareil multiplexage temporel donne à l'utilisateur du système l'impression de disposer d'un grand nombre de processeurs travaillant en parallèle alors qu'il s'agit simplement d'une gestion astucieuse des files d'attentes. [ROU-87]

On peut aussi parler de paraléllisme en terme de grain. Le grain de parallélisme est défini comme la taille des tâches mesurées :

- nombre d'instructions exécutées,
- taille de la mémoire utilisée.

On distingue:

Le parallélisme à grain fin (Fig.5 (a)):

- petit nombre d'instructions par tâche,
- grand nombre de tâches.
- petite taille mémoire pour chaque tâche,
- plus économique,
- beaucoup de temps avec des conflits de temps externes.

Le parallélisme à gros grain (Fig. 5(b)):

- grand nombre d'instructions par tâche,
- petit nombre de tâches.
- nombre de communications limité,

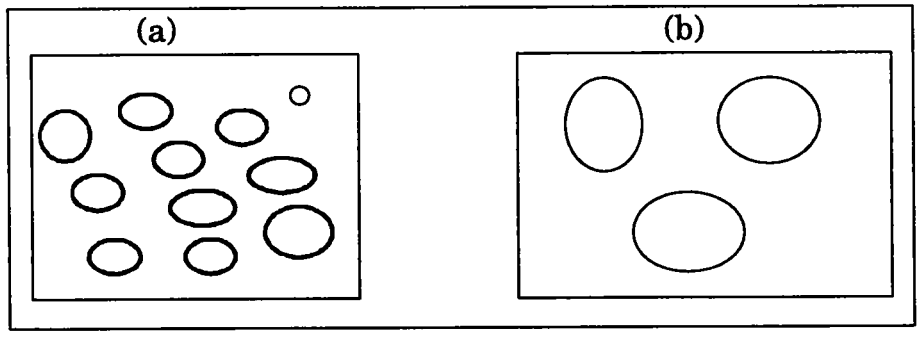


Fig. 5 (a):Parallélisme à grain fin (b) Parallélisme à gros grain

3 - Architectures Parallèles

L'architecture d'un réseau de processeurs est la manière dont un réseau de processeurs est agencé en vue d'une application spécifique. Cette architecture est organisée de manière à faciliter les communications entre les processeurs et à les limiter au strict minimum. On peut citer quelques familles d'architectures [BAN-91]:

Les systèmes multiprocesseurs, constitués d'un nombre élevé de processeurs relativement simples partageant des ressources communes, réalisent un parallélisme asynchrone .

Les machines à base d'une matrice de processeurs exploitent un certain nombre de processeurs de manière synchrone et permettent de réaliser un parallélisme spatial.

Les structures pipe-line exploitent un parallélisme temporel. Elles se présentent sous la forme de cascades d'unités appliquant chacune un traitement particulier aux données qui les traversent.

Les machines à flots de données : le séquencement des opérations est contrôlé par la disponibilité des données nécessaires à leur traitement. Ces machines permettent d'atteindre un haut degré de parallélisme.

Les structures systoliques. On peut les considérer comme des structures pipe-line à deux dimensions, car elles peuvent réaliser le parallélisme spatial et temporel. Les informations se propagent de manière pulsée dans le réseau.

Ces différents types de parallélisme peuvent être combinés.

3.1 Classification des architectures

Plusieurs critères peuvent être retenus, mais nous ne nous baserons que sur deux méthodes :

- classification de Flynn[CES-91], [DUB-91],
- classification par mémoires [CES-91].

3.1.1 Classification de Flynn

A la fin des années 70, M.J. Flynn a proposé une classification fondée sur l'analyse des architectures en fonction du nombre d'instructions et de chemins

de données existant dans l'architecture. Ce qui nous donne quatre classes de machines.

3.1.1.1 Architecture SISD (Single Instruction Single Data)

Une machine à architecture SISD est une machine monoprocesseur de type Von Neumann (Fig.6). Elle est composée d'une mémoire centrale, où sont stockés les données et les programmes, et d'un processeur.

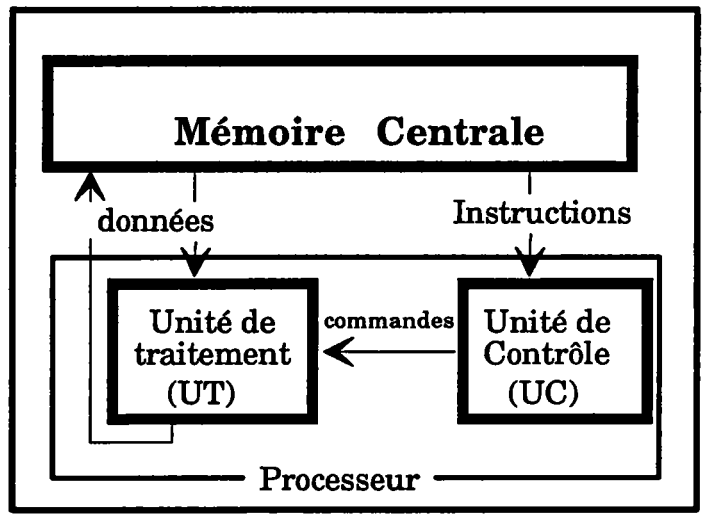


Fig 6 : Ordinateur de Von Neumann

Elle ne peut traiter en même temps qu'une seule instruction et ne peut aller chercher qu'une seule donnée. L'avantage d'une telle structure est qu'elle peut traiter n'importe quel problème à partir du moment où toutes les données sont connues.
Son inconvénient majeur est la lenteur, surtout si on veut faire des traitements temps réel.

3.1.1.2 Architecture SIMD (Single Instruction Multiple Data)

C'est la structure la plus simple à mettre en oeuvre. Un ensemble de processeurs élémentaires réalisent le même traitement de façon synchrone sur des données différentes (Fig. 7)

Les processeurs de ce type de machines sont généralement faibles en puissance de calcul par rapport aux processeurs des machines séquentielles. Pour le programmeur, tout se passe comme si son programme s'exécutait sur une machine séquentielle classique, à la seule différence que les instructions effectuent des opérations sur des ensembles de données au lieu d'opérations sur une seule donnée.

Des liaisons sont donc nécessaires entre les processeurs, mais l'idéal serait que chaque processeur soit connecté à tous les autres, ce qui n'est pas réalisable pour des réseaux importants. En général le traitement à effectuer est sous le contrôle d'un seul processeur appelé maître.

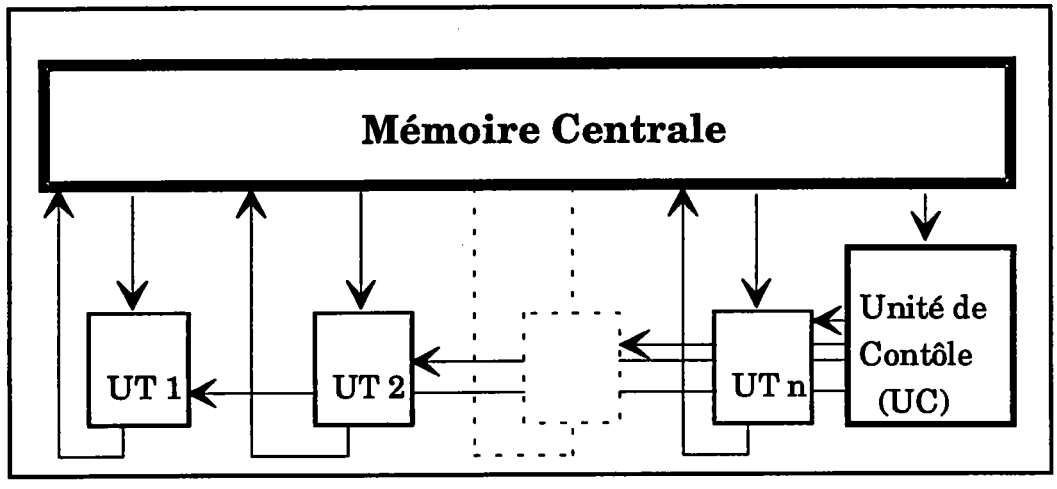


Fig 7 : l'architecture SIMD.

On trouve dans cette famille le processeur Iliac IV, DAP (4096 processeurs 1 bit), la Connection Machine CM (64K processeurs), la Mas Par (Massively Parallel (16K processeurs 4 bits) [CES-90], [LEA-92].

Les machines SIMD peuvent être connectées différemment suivant la nature du problème à traiter (Fig. 8).

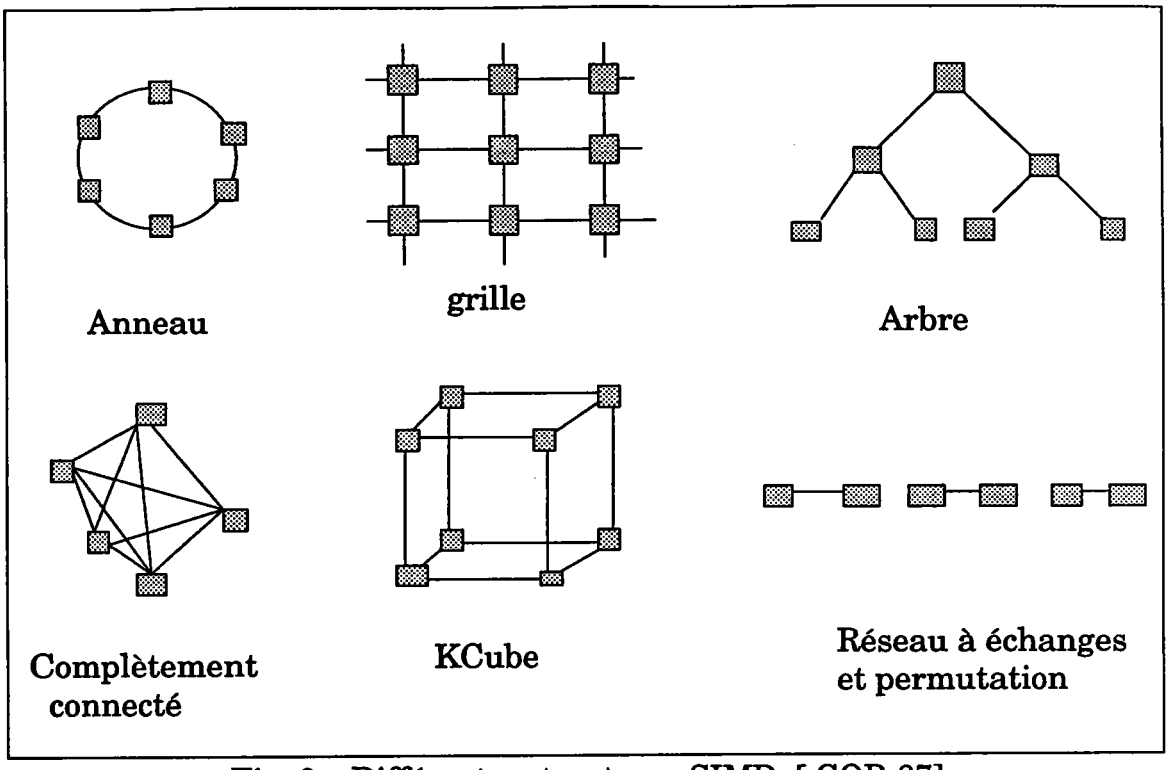


Fig 8 : Différentes structures SIMD [COR-87]

De telles architectures sont très efficaces, mais sont souvent adaptées à des algorithmes spécifiques.
Les architectures systoliques et vectorielles font partie de cette famille. En traitement d'image une structure de type SIMD est surtout adaptée au cas où les traitements sont identiques sur différentes sous-images.

3.1.1.3 Architecture MISD (Multiple Instruction Single Data)

Un ensemble de processeurs élémentaires montés en cascade exécutent simultanément différentes opérations sur les mêmes données. Ces architectures se réfèrent au modèle pipeline que l'on retrouve dans les RISC, les machines VLIW (Very long Instruction Word) et les supercalculateurs vectoriels. L'avantage de la structure pipe-line par rapport à celle de Von Neumann, est qu'elle ne nécessite que peu d'accès à la mémoire.

3.1.1.4 Architecture MIMD (Multiple Instruction Multiple Data)

Lorsque le flot des données est trop important, il n'est plus possible d'avoir une et une seule sortie. Il devient alors nécessaire de prévoir plusieurs flots de données qui sont traités par plusieurs processeurs : c'est le cas de la structure MIMD.

Un ensemble de processeurs élémentaires effectuent des traitements différents sur des données qui leur sont propres. Chaque processeur élémentaire possède sa propre unité de contrôle (UC) et de traitement (UT) (fig. 9).

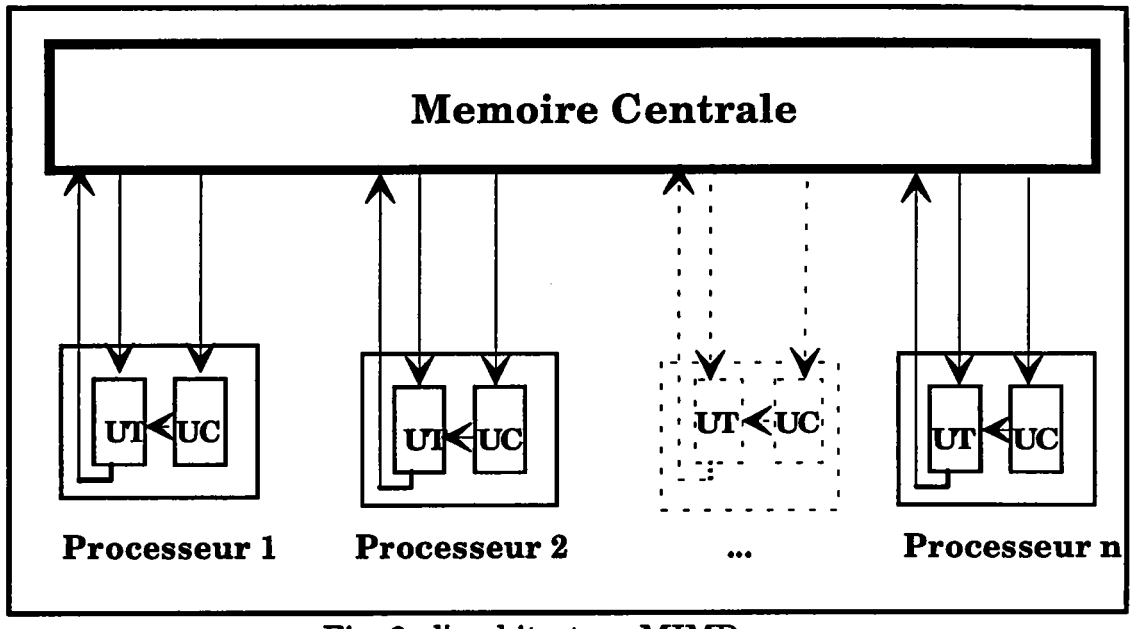


Fig 9 : l'architecture MIMD.

Dans cette classification (Fig. 10), les machines MIMD correspondent au mode de calcul concurrent, les SIMD au mode de calcul parallèle, les SISD à l'ordinateur séquentiel de Von Neumann et les machines MISD constituent une classe à part, les machines pipe-line.

		Flot de Données	
		Simple	Multiple
Instruction	Simple	**SISD** (Von Neumann)	**SIMD** (Vectorielles et cellulaires)
	Multiple	**MISD** (Pipe-line)	**MIMD** (Multiprocesseurs et passages de messages)

Fig 10 : Classification de Flynn [CES-91]

3.1.2 Classification par mémoire.

Une architecture multiprocesseurs est composée de processeurs et de mémoires, il faut donc un système permettant de connecter les éléments entre eux. Dans la classification de Flynn, les processeurs élémentaires n'ont pas de mémoire propre; ils accèdent tous à une mémoire centrale qui contient les données et les programmes. Les progrès de la technologie ont permis la réalisation de mémoires rapides et de grande taille. On a vu apparaître des machines parallèles où la mémoire est distribuée c'est à dire que chaque processeur élémentaire dispose de sa propre mémoire de données et de programmes.

3.1.2.1 Machines parallèles à mémoire partagée

C'est la classe des machines parallèles appelées supercalculateurs. Un ensemble de processeurs travaillent à l'aide de la même mémoire , mais peuvent exécuter des instructions différentes. L'accès à la mémoire commune se fait via un réseau d'interconnexions. Pour gérer les conflits d'accès, la mémoire commune peut être divisée en bancs de mémoires. A un instant donné, un banc n'est accédé que par un processeur, soit en lecture, soit en écriture.

Ces machines sont limitées à quelques dizaines de processeurs pouvant atteindre une grande puissance (processeurs vectoriels) d'où un très fort taux de parallélisme. On rencontre dans cette famille les multiprocesseurs(MIMD) et les machines vectorielles (SIMD), le Cray2, Cray XMP, l'Alliant FX, Nyu, RP3, ... [ROU-87], [KAI-89], [CRA-92].

3.1.2.2 Machines parallèles à mémoire distribuée (DMPC)

C'est la classe des machines MIMD à mémoire distribuée ou DMPC (Distributed Memory Parallel Computer). Chaque processeur travaille avec sa propre mémoire. Il est nécessaire d'ajouter un réseau externe d'intercommunication entre les processeurs. Les processeurs communiquent par messagerie.

Ce type d'architecture permet d'allier l'utilisation d'un très grand nombre de processeurs et la puissance de chacun d'eux. Mais si le nombre de processeurs est trop élevé, technologiquement il devient impossible que tous les processeurs accèdent à tous les autres processeurs.
Dans cette famille on rencontre les machine à base de Transputers (TNode, Volvox) et d'autres tels IPSC/860, NCube, CM5 (Connection Machine Five), ...

3.2 Conclusion

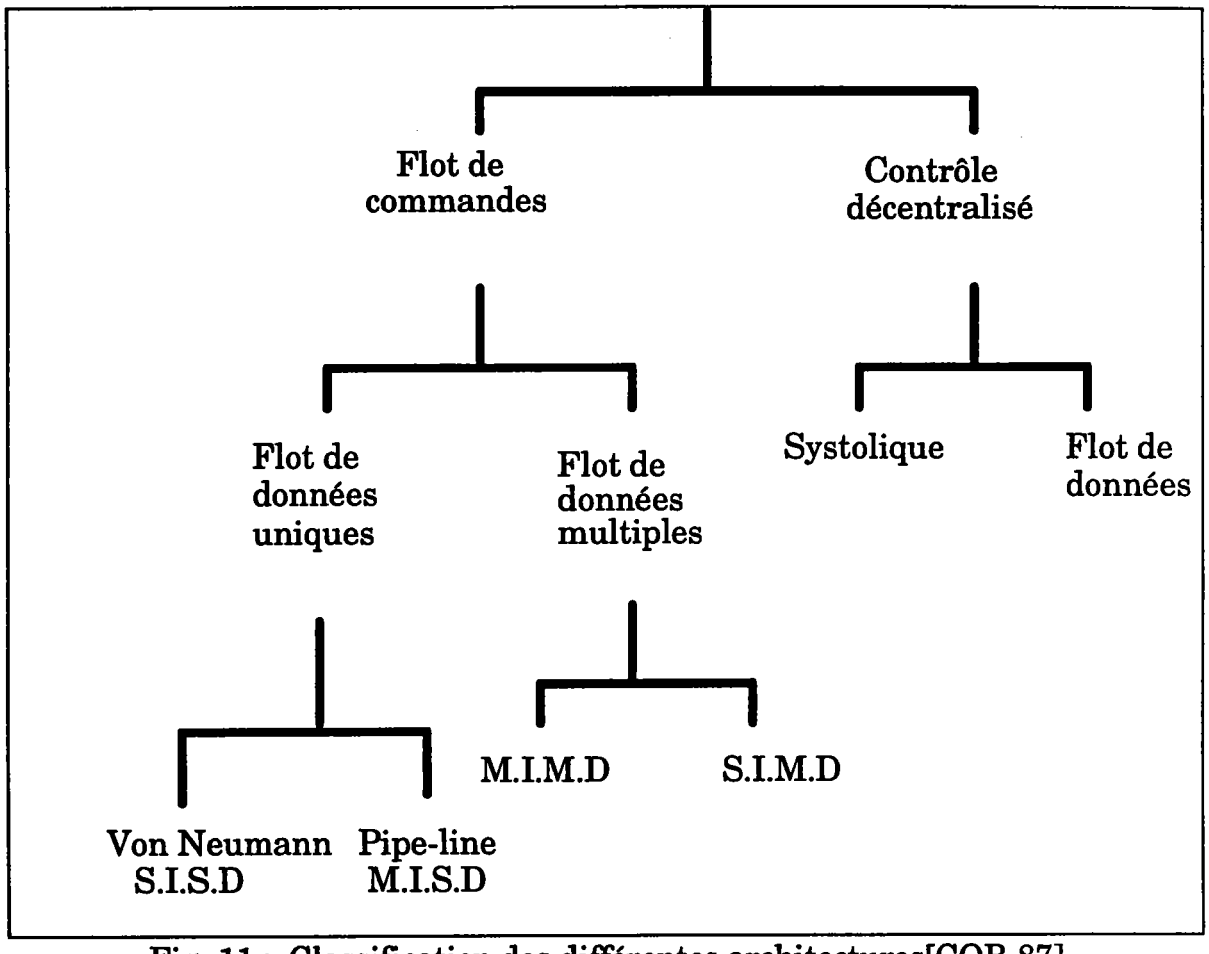


Fig .11 : Classification des différentes architectures[COR-87]

Les architectures (Fig. 11) parallèles permettent de traiter plus vite un grand nombre de données.

Les structures avec flux de données unique (SISD et MISD) peuvent accélérer le traitement. Mais des problèmes sont rencontrés lorsque les données arrivent simultanément ou à des intervalles très rapprochés. Ces problèmes de débit ont conduit à étudier des structures à flux de données multiples (MIMD et SIMD).

L'architecture MIMD permet de répondre à long terme à tous les types d'algorithmes. Elle permet par ses contrôles délocalisés d'envisager la réalisation d'algorithmes réguliers(voir paragraphe suivant) ou non, d'implémenter un parallélisme sur les tâches et sur les données à différents niveaux. Malheureusement, ce type d'architecture est plus complexe à mettre en œuvre.

4 - Classification des algorithmes

Le but de l'analyse algorithmique est une utilisation optimale d'architecture à microprocesseurs pour implanter un algorithme donné. On distingue deux classes d'algorithmes [DUB-91].

Les algorithmes réguliers : on connaît d'avance le placement des données et on peut évaluer le temps de traitement en fonction de la taille.Une répartition statique des données sur les différents processeurs peut être effectuée avant de commencer le traitement. En statique le placement des tâches sur chaque processeur est défini au départ et ne peut être modifié en cours d'exécution.

Les algorithmes irréguliers : on ne connaît pas le placement des données ni le nombre d'accès à ces données avant traitement. On effectue une allocation dynamique des données qui se fait suivant la disponibilité de chaque processeur en cours de traitement.

5 - Techniques de Parallélisation

Parmi les applications nécessitant une architecture parallèle, on distingue:

-celles où l'on cherche à augmenter la vitesse de traitement afin d'obtenir des temps de réponse aussi courts que possible,

- des applications temps réel, pour lesquelles la durée d'exécution est une contrainte impérative.

Les différentes manières d'exploiter la notion de parallélisme sont les suivantes:

5.1 Parallélisation par les données

Les données sont équitablement distribuées aux différents processeurs du réseau. Chaque processeur doit effectuer tous les calculs sur ses propres données. Cette technique est surtout appliquée aux algorithmes réguliers.

5.2 Parallélisation par les calculs

Un sous ensemble du domaine des calculs est affecté à chaque processeur. Si le calcul nécessite les données situées sur un autre processeur, il peut les obtenir par copie. Cette technique est surtout adaptée aux algorithmes irréguliers.

5.3 Parallélisation par mémoire virtuelle

Elle est appliquée en général sur les machines MIMD à mémoire distribuée. Elle consiste à gérer la mémoire de l'ensemble du réseau et à éliminer toutes les contraintes de communication et d'échange de données. Ainsi chaque processeur peut voir la mémoire commune comme un espace unique adressable.

5.4 Parallélisation SPMD

Le modèle Single Programm Multiple Data (SPMD) peut être considéré comme un compromis entre les architectures MIMD et SIMD. Il s'applique aux architectures MIMD mais chaque processeur doit exécuter le même programme. C'est un modèle type parallélisation des données avec plus de contraintes. Les processeurs se synchronisent après avoir effectué leur tâche. Les tâches sont effectuées de manière asynchrone et beaucoup plus efficacement.

5.5 La ferme de processeurs (farming)

C'est une exécution indépendante, concurrente et asynchrone. Un certain nombre de processeurs accomplissent une tâche bien déterminée sur des données qui leur sont envoyées par un processeur de contrôle. La synchronisation des tâches ne s'effectue qu'à la fin des différents traitements.

Le farming est une des techniques de parallélisation les mieux adaptées au problème des algorithmes à données séparables. Le principe consiste à répliquer l'algorithme sur tous les noeuds du réseau et à leur envoyer les données par paquets. Comme toutes les techniques de parallélisation cela nécessite :

- un découpage des données en blocs adaptés à l'algorithme,

- un traitement des blocs : chaque bloc envoyé dans le réseau est traité par un processeur disponible,

- les blocs ne revenant pas forcément dans le même ordre qu'au départ, il faut a priori prévoir un tri à leur retour pour la reconstruction des données résultats.

Le principe du farming général est le suivant:

- un processeur principal appelé "maître" a une vue globale sur tous les

autres processeurs du réseau appelés "esclaves" et il fournit un paquet de données à tout "esclave" disponible (Fig. 12). Cela nécessite que le maître ait un lien de communication avec tous les processeurs du réseau. On est en présence d'une structure en étoile.

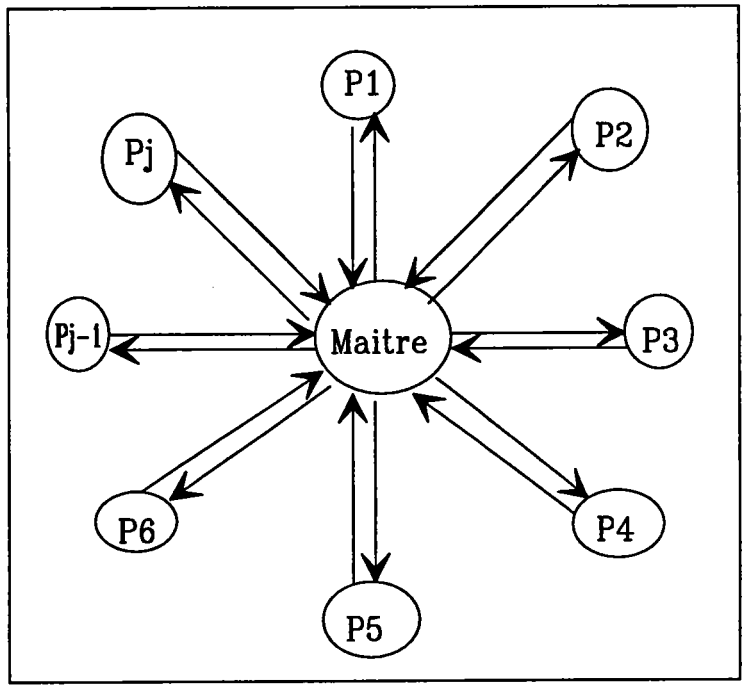


Fig .12: Principe du farming

L'avantage d'une telle structure est que le maître est capable de savoir à tout moment quels sont les processeurs qui travaillent et ceux à qui il faut envoyer un paquet de données.

5.6 Méthodes de parallélisation

Pour mener à bien tous ces modèles, il faut gérer correctement les communications et le placement des tâches, car les tâches participent à l'accomplissement d'un même travail. L'algorithme sera considéré comme un ensemble de tâches communicantes.

Placement des tâches: la décomposition en tâches peut être automatique ou effectuée par le programmeur. Comme elle n'est pas généralement unique, le choix du meilleur découpage est un problème difficile, car il est lié au temps de décomposition des tâches et à l'architecture.

Si les tâches sont trop importantes, le taux de parallélisme est faible.

Si les tâches sont trop petites, elles font perdre du temps en synchronisation.

De plus il faut trouver un bon ordonnancement des tâches pour minimiser le temps total d'exécution.

Structures des communications: deux modèles d'échange sont possibles:

- l'un basé sur le partage d'une zone mémoire dans laquelle les tâches déposent et récupèrent les informations échangées. L'utilisation d'une mémoire commune simplifie les communications et les synchronisations, mais elle n'est accessible que par un nombre limité de processeurs,

- l'autre se faisant par messagerie entre tâches expéditrices et destinataires. Pour obtenir un bon niveau de performances, il faut que le rapport entre la puissance des processeurs et le débit des liaisons inter-processeurs soit optimum. Ce rapport dépend du problème à traiter.

6 - Logiciel et Parallélisme

Il n'est pas encore possible d'exploiter pleinement le parallélisme en raison du retard de langages de programmation ou de compilateurs appropriés et de logiciels d'applications. Actuellement, deux voies de recherche sont explorées:

- La première approche consiste à paralléliser les langages déjà existants(C, Pascal, Fortran, ...). Cette méthode ne permet pas de tirer profit de toute la puissance offerte par les machines multiprocesseurs. Ces langages conduisent souvent à des programmes très longs qui n'offrent pas toute la souplesse de développement. En général, soit l'algorithme n'est pas adapté au traitement parallèle soit le langage de programmation ne permet pas d'exprimer le parallélisme à l'exécution.

- La deuxième, plus prometteuse conduit à développer de nouveaux langages de programmation spécialement adaptés à certains types de machines. Ces langages exploitent au mieux le parallélisme car seuls les composants nécessaires à leur mise en oeuvre sont réalisés. C'est le cas du langage OCCAM conçu pour les machines à base de transputers (voir Annexe). Cette approche ne répond pas bien à la demande des utilisateurs qui souhaitent compiler leurs applications sur machine parallèle sans les réécrire entièrement de manière à bénéficier facilement du gain de vitesse.

7 - Performances

7.1 Coût des communications

Deux processeurs voisins peuvent communiquer à travers un lien qui peut être bidirectionnel.

Si deux processeurs non voisins veulent communiquer, le message doit traverser plusieurs processeurs du réseau. Dans cette architecture :

- le message est stoppé sur chaque noeud intermédiaire, stocké , puis émis vers le processeur suivant. Cette technique est utilisée sur les machines à base de transputers (Tnode) et sur le Ncube car le routage se fait alors de manière logicielle.
- la communication peut être directe. Les processeurs ont un routeur matériel (iPSC/2, iPSC/860). Dans ce cas, le temps de communication est indépendant de la distance entre les deux processeurs.

7.1.1 Temps de communication entre 2 processeurs voisins

Le temps nécessaire pour acheminer un message de taille L (exprimé en octets) entre deux processeurs voisins est décrit comme la somme [MIG-92], [UBE-93] :

$$T = \beta + L\delta$$

où β désigne le temps d'initialisation du lien en secondes,
L la longueur du message en octets,
δ le débit du canal en micro-secondes/octet,
et $L\delta$ le temps de propagation.

Machines	β	δ
iPSC/1 (L<1000)	1000 µsec	4 µsec
iPSC/2 (L<100)	370 µsec	0,75 µsec
T800	5 µsec	1,1 µsec
iPSC/860	136 µsec	3,2 µsec

Tableau 1 : valeurs de ces constantes pour différentes machines [MIG-92]

7.2 Facteur de Gain

Le Gain est défini de manière particulière (contrairement aux autres systèmes) comme suit:

Soit un problème traité sur un seul processeur en un temps Tseq. Ce même problème est traité en parallèle sur p processeurs en un temps Tpar(p).

$$G = (Tseq - Tpar(1))/Tseq$$

$$G = 1 - Tpar(1)/Tseq$$

Le gain met en avant le coût de la parallélisation en terme de performance du processeur. Il est proche de 1 lorsque le processeur est très performant.

7.3 Facteur d'accélération

L'accélération A(p) peut être définie de la manière suivante:

$$A(p) = \frac{Tseq}{Tpar(p)}$$

On étudie le facteur d'accélération en faisant varier p.
L'accélération met en avant le gain de temps apporté par la parallélisation.

7.4 Facteur d'efficacité

L'efficacité E(p) d'un algorithme parallèle sur p processeurs peut être définie comme suit:

$$E(p) = \frac{A(p)}{p}$$

L'efficacité met en avant le coût de la parallélisation en terme de processeurs. Il est proche de 100% lorsque les p processeurs sont utilisés de manière optimale. Mais il est ralenti par certains facteurs:

- la partie purement séquentielle de l'algorithme (loi d'Amdhal)[GUS-88],
- les communications entre les processeurs,
- les temps d'attente des communications du réseau.

8 - Conclusion

Les architectures parallèles sont matériellement maîtrisées à l'heure actuelle. Malgré l'apparition sur le marché de plusieurs machines parallèles: à passage de mémoire, à mémoire partagée, SIMD, MIMD, hypercubes, etc, leur utilisation n'est pas toujours optimale car tout le problème réside dans leur programmation. En effet les langages de programmation classiques n'offrent pas toute la souplesse pour tirer le maximum de profit du parallélisme dans les divers problèmes. Au niveau logiciel apparaissent de nouveaux langages et systèmes d'exploitation mieux adaptés au parallélisme.

BIBLIOGRAPHIE

[BAN-91] Jean-Pierre Banâtre, "La programmation parallèle - Outils, méthodes et éléments de mise en oeuvre", Eyrolles, Paris, 1991.

[BAS-85] Jean Luc Basille, "Structures parallèles et traitement d'image", Thèse d'Etat, Université Paul Sabatier de Toulouse, 1985.

[BCT-85] J. L. Basille, S.Castan, J.Y. Latil, "Structures parallèles en traitement d'images ", Premier Colloque Image, Biarritz, Mai 1984.

[BUH-92] R.M. Burger, W.C. Holton, "Reshaping the microchip", Byte, p.137-150, fév. 1992.

[CAT1-86] Eric Catier, "Des processeurs à architecture parallèle" Electronique Industrielle N°107, p. 68-72, 1986.

[CAT2-86] Eric Catier, "Une architecture HYPERCUBE" Processeurs et systèmes N°5, Septembre 1986.

[CES-91] Cécile GERMAIN-RENAUD, Jean Paul SANSONNET "Les Ordinateurs massivement parallèles " Edition Armand Colin

[CLA-89] Philippe Clausse "Temps Réel, Multiprocesseur et Unix: Le choix de Readstone", Minis & Micros N°322/15 Mai 1989

[COR-87] M. Cornu "Les Architectures parallèles" Micro-Systemes, p.159-164, Septembre 1987.

[CRA-92] Cray Research, Inc., "MPP technology preview", Cray Research, 1992.

[DEG-84] D. Degrott, "Restricted AND-Parallelism" International conférence on fifth generation computer systems, pp. 471-478, november 1984

[DOR-91] A. Dorseuil and P. Pillot, "Le temps réel en milieu industriel - concepts, environnements, multitâches", Dunod , Paris 1991

[DUB-91] H. Dubois, " Analyse des systèmes multiprocesseurs: application à la mise en oeuvre sous contraintes d'algorithmes de traitement d'images" thèse d'Université, Université de Rennes I, Janvier 1991.

[DUN-90] T.H. Dunigan, "Performances of the intel ipsc/860 hypercube. Technical Report 11491, Oak Ridge, Tennessee 37831, June 1990.

[GAE-90] J.M. Gaetner, "Unités centrales parallèles, réseaux de processeurs", rapport interne, Conservatoire national des Arts et Métiers de Mulhouse, Novembre 1990.

[GUS-88] J.L. Gustafson, "Reevaluating amdahl's law", Comm. of ACM, 31(5) pp.. 532-533,1988.

[HEM-86] Yves Hemery, "L'architecture des processeurs vectoriels", Electronique Industrielle, N°112, pp. 89-93, 1986.

[HEL-92] Bénédicte Herrman, Laurent Philippe, "CHORUS/MIX, a Distributed UNIX on Multicomputers", Transputers'92 Advance Research and Industrial Applications, IOS Press, p. 142-157, 1992.

[HER-86] A.Herscovici, "Introduction aux grands ordinateurs scientifiques: ordinateurs scalaires, ordinateurs vectoriels, ordinateurs parallèles", Eyrolles, 1986.

[HIR-90] Ernest Hirsh, "Les transputers - Application à la programmation concurrente", Eyrolles, Paris, 1990.

[INM-91] INMOS Limited, "The transputer development and *i*q systems databook", second edition, 1991.

[INM-90a] INMOS Limited, "ANSI C toolset language reference", August 1990.

[INM-90b] INMOS Limited, "ANSI C toolset user manual", August 1990.

[INM-89] INMOS Limited, "The transputer databook", first/second edition, 1989.

[KAI-89] Kai Hwang and Fayé A. Briggs, "Computer architecture and parallel processing", International Edition, 1989.

[KUP-90] S. Kuppuswami an Touranchean, Evaluating the performances of transputers based hypercube vector computer. La Lettre du Transputer vol 4, 1990.

[LAN-90] Jean Lantier, "Le Transputer, une alternative au microprocesseur ?", Soft & Micro, Mars 1990, pp 200-208.

[LEA-92] T.Lévy-Abégnoli, "Les systèmes d'exploitation vont sauver le parallélisme", Electronique International, 1er Octobre 1992.

[MIG-92] Serge Miguet, Virgine Poty, "Revisting the allocation of regular data arrays to a mesh of processors" AAA 92, pp. 80-96, Septembre 1992.

[MOP-86] J.A. Montagnon, E.Pichat, "Architecture des ordinateurs, tome 1, Le sous-système central", Masson, 1986.

[PAS-91] O. Pasquier and J. P. Calvez, "Utilisation du transputer pour les applications temps réel", La lettre du transputer et des calculateurs distribués, N° 10, Juin 1991, pp. 7-34.

[PDJ-90] Alain Pirson, D. David, J. L. Jacquot et T. Court, "Formalisation en occam et simulation sur un réseau de Transputers d'un processeur systolique de traitement d'images", Traitement du Signal, vol 7, n°1, 1990, pp 41-51.

[PIN-90] A. Pinti, N. Schaltenbrand, M. Toussaint, J. Gresser, R. Luthringer, R. Minot et J. P. Macher, "Etude d'un réseau de neurones multi-couches pour l'analyse automatique du sommeil sur T-Node", La lettre du transputer et des calculateurs distribués, N° 8, Déc. 1990, pp. 21-32.

[PIR-90] Alain Pirson, "Conception et simulation d'architectures parallèles et distribuées pour le traitement d'image",Thèse d'université, Université de Grenoble, Mai 1990.

[PRE-82] Kendall Preston Jr and Leonard Uhr, "Multicomputers and image processing - Algorithms and programs", Academic Press Inc., London, 1982

[QUIN-85] Patrice Quinton, "Les hyper-ordinateurs" La Recherche, n°167, p.740-749, juin 1985.

[RAN-88] Randy Allen and Steve Johnson, "Compiling C for Vectorization, Parallelization, and Inline Expansion ", Proceedings of the SIGLPLAN '88, Conference on Programming Language Design and Implementation, Atlanta, Georgia, June 22-24, 1988, pp 241-249.

[REM-87] C. Remy "Supercalculateurs : la course à la puissance", Informatique Hebdo, N°1, pp. 41-47, Novembre 1987.

[ROS-86] A.w. Roscoe, C.A.R. Hoare, "The laws of Occam programming", Oxford, 1986.

[ROU-88] Rousseau M. "les architectures parallèles ", INFORMATIQUE HEBDO No3 du 10 Décembre 1987, pages 25-31.

[ROU-87] M. Rousseau "Les Architectures parallèles" Informatique Hebdo, N°3, pp. 25-31, Décembre 1987.

[RYA-92] B. Ryan, "Built for speed", Byte, p.123-136, fév. 1992.

[UBE-93] S. Ubeda "Algorithmes d'amincissement d'images sur machines parallèles" Thèse d'Université, Université Claude-Bernard- Lyon I, Février 1993.

Chapitre II

LE TRAITEMENT PARALLELE DE L'IMAGE

- Introduction

Le traitement d'images est sans aucun doute un domaine privilégié pour l'étude et la mise en oeuvre du parallélisme. En effet les algorithmes de traitement d'images nécessitent une puissance de calcul justifiant l'utilisation de machines parallèles.
Les traitements possibles sur l'image sont les traitements de bas, moyen ou haut niveau.

Dans le traitement de bas niveau, les tâches sont caractérisées par une forte régularité et récursivité des opérations. Les traitements sont effectués sur l'image entière ou une fenêtre sur l'image. Il s'agit des filtres, des opérateurs d'histogramme et de la génération de gradients.

En traitement de moyen niveau, on cherche d'abord des connaissances sur des objets de l'image. Les traitements consistent à repérer, à caractériser ou à modifier ces objets de l'image. Il s'agit des opérateurs pour le suivi de contours, l'étiquetage, etc.

En traitement de haut niveau, les tâches sont moins bien définies. Les traitements sont effectués sur un extrait de l'information contenue dans l'image.

Ainsi, la quantité des données à manipuler et la complexité des calculs font appel à une puissance de calcul importante et notamment lorsque le temps réel est requis. La puissance de calcul que peut produire une machine parallèle permet d'accélérer l'exécution de ces divers traitements. Les techniques d'exploitation du parallélisme doivent être mises en œuvre pour réduire le temps de calcul.

2- Classification des algorithmes

Le but est de distinguer les algorithmes récursifs et non récursifs à savoir ceux pour lesquels un réel parallélisme spatial est possible et ceux pour lesquels on ne peut pas envisager de traiter simultanément les pixels d'une même image.

2.1 Algorithmes récursifs

Un pixel a un passé, un présent, un futur. Le traitement d'un pixel dépend du traitement des pixels qui le précèdent dans le parcours de l'image (filtres de KALMAN [DUB-91], algorithme du gradient, algorithmes sur les statistiques

de l'image, programmation séquentielle). Dans ce cas le seul parallélisme possible sur les données est le type pipeline

2.2 Algorithmes non récursifs

Le traitement d'un pixel ou d'un bloc de pixels est indépendant du traitement des pixels précédents ou des blocs de pixels précédents.
Le traitement d'une image découpée se fait réellement en parallèle sur chaque bloc même si des échanges entre processeurs sont parfois possibles.
Il existe plusieurs types d'algorithmes non récursifs suivant la répartition des données.
Les différents comportement des données sont:

- accès aux données locales à un processeur,
- accès aux données des processeurs voisins,
- accès à des données distantes.

***Algorithmes à accès voisins*:** ce sont les algorithmes de pré-traitement des images avant analyse (filtrage, convolution, etc. ...). Chaque processeur traite en parallèle une partie de l'image. Mais les processeurs ont parfois besoin d'accéder aux échantillons dans un voisinage immédiat.

Algorithmes à accès locaux : pour un traitement donné, il ne faut accéder qu'à un pixel ou à un groupe de pixels.

3 - Codage d'images

Les techniques de compression et de codage ont pour but de réduire le nombre moyen de bits par pixel à transmettre ou à stocker. Le taux de compression est défini par le rapport entre le nombre de bits par pixel de l'image originale et le nombre moyen de bits par pixel nécessaires à sa transmission ou à son stockage. Des taux de compression élevés ne peuvent être obtenus que par des méthodes qui tolèrent une distorsion de l'image originale en préservant l'information utile. Les méthodes de compression se divisent en deux catégories:

- les méthodes spatiales ,
- les méthodes par transformation .

Les méthodes spatiales agissent directement sur les pixels de l'image pour les coder convenablement.

Tandis que dans les méthodes par transformation, on ne code pas une image mais les coefficients numériques de sa transformée .

3.1 Le Codage spatial

3.1.1 Codage par quantification vectorielle

La quantification vectorielle, appliquée au codage d'images, fournit des caractéristiques intéressantes pour coder une image à bas débit. Le but recherché étant une bonne qualité visuelle subjective plutôt que la reproduction exacte de l'image.
La quantification vectorielle consiste à remplacer un ensemble de pixels appelé vecteur par un représentant choisi dans un dictionnaire préétabli (Fig. 1). Ce dictionnaire, ainsi composé des éléments les plus représentatifs de l'image, est transmis au décodeur. Une fois la recherche (règle du plus proche voisin) terminée, seul l'index du vecteur du dictionnaire est transmis au décodeur permettant ainsi un fort taux de compression.

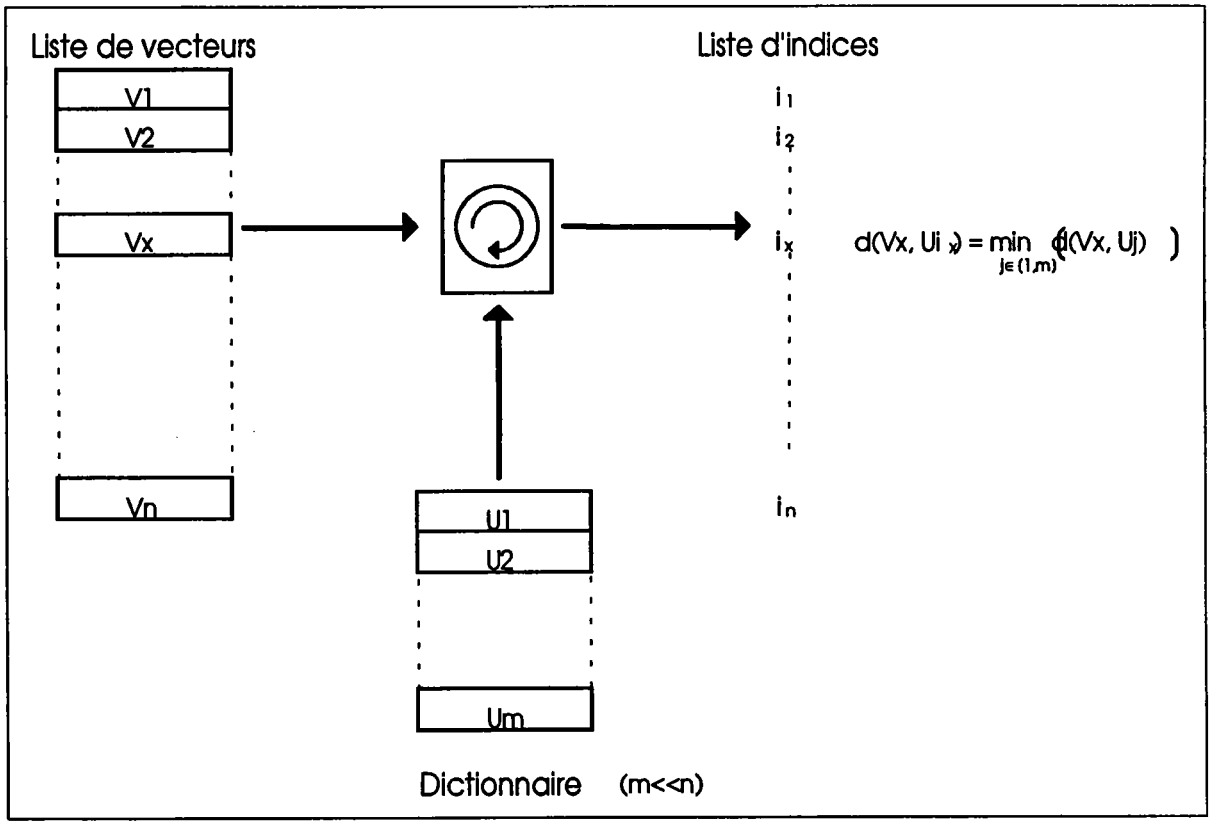


Fig 1 : Principe de la quantification vectorielle

La recherche exhaustive du plus proche voisin dans le dictionnaire se fait par distance.

Généralement une des quatre distances suivantes est utilisée.

- distance euclidienne = $\sum_{i=0}^{k-1}(Xi-Wi)^2$
- distance absolue = $\sum_{i=0}^{k-1}|Xi-Wi|$
- distance pondérée = $\sum_{i=0}^{k-1}|Xi-Wi|\ /2^i$
- distance sup. = $\max_{i=0,k-1}(|Xi-Wi|)$ (ou distance de Minkowski)

avec X un vecteur d'entrée, W un vecteur du dictionnaire et k la dimension de ces vecteurs.

Avantages

- la quantification vectorielle permet d'obtenir un taux de compression assez élevé avec des vecteurs de petite taille, en effet plus les vecteurs sont de taille limitée, meilleure est la qualité subjective.

- Simplicité du décodeur.

Création du dictionnaire:

Il existe différentes méthodes pour créer le dictionnaire:

- La méthode de " treillis " basée sur l'étude géométrique des meilleurs représentants d'une partition, elle permet d'obtenir des contours nets et une limitation du coût de calcul par rapport à celui d'un quantificateur scalaire [MAR-86].

- La méthode LBG (Linde, Buzo et Gray) [LBG-80], le dictionnaire est généré à partir d'une "séquence d'entraînement" ou "training set ", ensemble de vecteurs issus de l'image, et d'un dictionnaire initial. Le principe consiste à partitionner l'ensemble d'apprentissage à partir du dictionnaire initial, ensuite chaque partition est remplacée par son centre de gravité de façon à obtenir un nouveau dictionnaire. Ces deux étapes sont répétées jusqu'à convergence.
Un tel algorithme itératif ne converge pas en général vers la distorsion moyenne minimale, mais vers un optimum local dépendant du dictionnaire initial et de la nature de l'image. Le choix de ce dictionnaire initial devient donc capital.

Choix du dictionnaire initial : Le choix du dictionnaire initial est important , car c'est de lui que va dépendre en partie la validation du dictionnaire final.

Méthode aléatoire: Elle consiste à utiliser le N premiers vecteurs de la séquence d'entraînement. En codage d'images , la distribution des vecteurs de la séquence d'entraînement étant inconnue, cette approche ne peut fournir qu'un dictionnaire fortement sous-optimal.

Méthode des histogrammes multidimensionnels: Elle permet de connaître les vecteurs les plus probables et d'en détecter les pics. Mais, elle est coûteuse en mémoire pour des vecteurs de grande taille .

Méthode de 'Découpage': Elle consiste à construire itérativement des dictionnaires de taille croissante jusqu'à l'obtention du nombre de représentants voulu.

Elle peut être schématisée comme suit:

- a) on calcule le vecteur V barycentre de tous les vecteurs Ui de la séquence d'entraînement qui sera le premier vecteur du dictionnaire.

$$V = 1/N \, \Sigma Ui$$

- b) à partir du barycentre de la séquence d'entraînement , on forme deux nouveaux vecteurs en ajoutant deux perturbations opposées:

$$V1 = V + \varepsilon$$
$$V2 = V - \varepsilon$$

ε est un vecteur perturbateur fixe

- c) Ce nouveau dictionnaire est optimisé en utilisant l'algorithme LBG c'est à dire qu'on calcule la distance entre V1, V2 et chaque vecteur Ui de la séquence d'entraînement, puis deux classes sont formées. Les centres de ces deux classes vont former un nouveau dictionnaire.

- d) Par itération du procédé, on atteint la taille fixée du dictionnaire.

3.2 Codage par transformation

Il s'agit de transformer le plan image dans un plan fréquentiel par une opération linéaire de manière à obtenir des données transformées décorrélées. L'efficacité de la méthode dépend de la qualité de la décorrélation (Fig.2). En effet, le spectre de l'image transformée est caractérisé par une forte concentration de l'énergie dans les basses fréquences. Cela s'explique par le fait qu'une image contient beaucoup de plages uniformes, donc il faudra peu de bits pour coder les coefficients situés dans les hautes fréquences. La transformation ne s'applique pas sur l'image entière mais sur des blocs

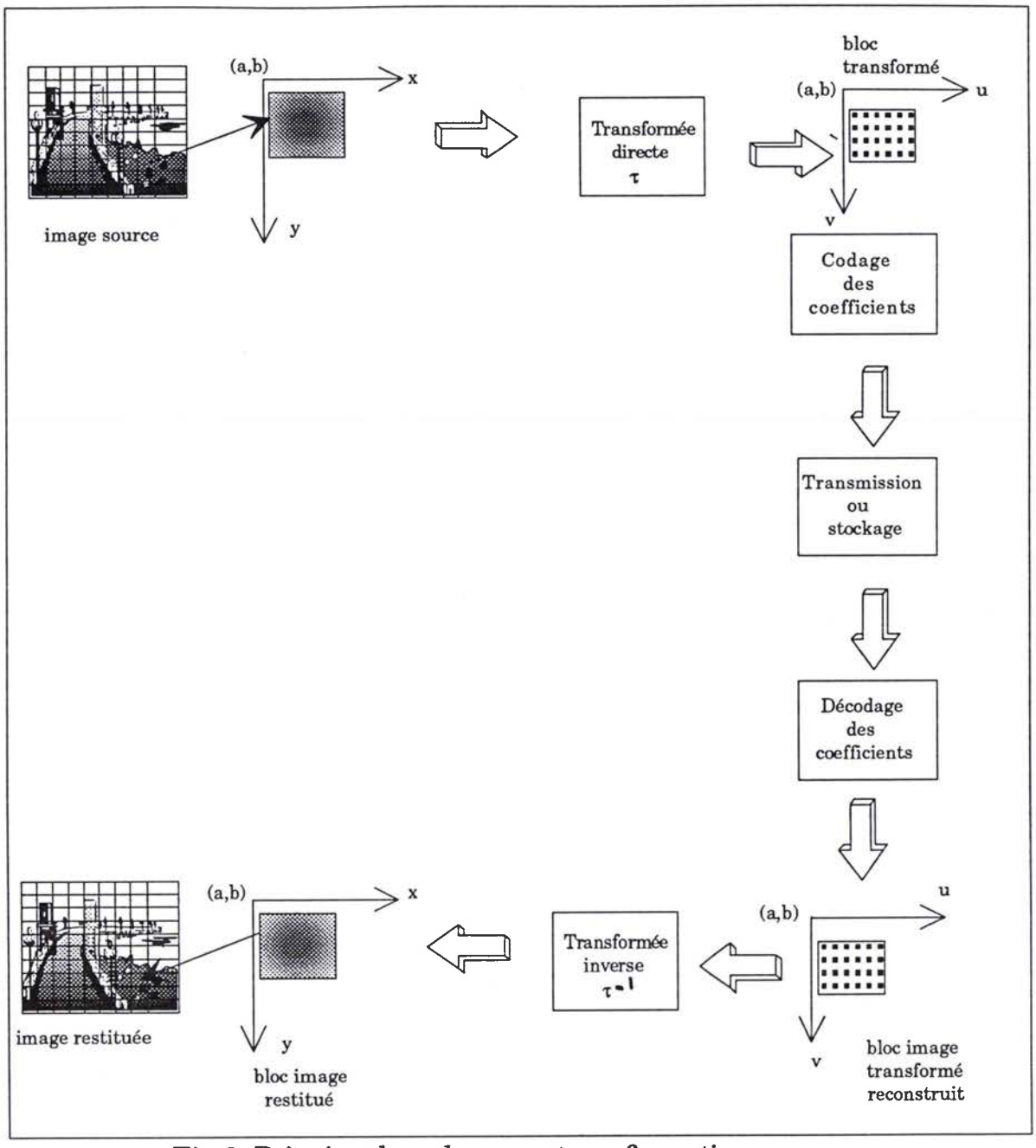


Fig 2: Principe du codage par transformation

'images de taille 8x8 ou 16x16. Au décodage, il suffit d'appliquer à l'ensemble des coefficients transformés la transformation linéaire inverse de celle utilisée au codage. On obtient ainsi l'image initiale au détriment de quelques dégradations.

- Elle est bien adaptée à l'exploitation des décorrélations spatiales de l'image.

- Les erreurs sur les coefficients sont diluées spatialement au niveau de

l'image restituée lors du décodage.

3.1.1 Les transformées orthogonales

Les transformées orthogonales permettent de concentrer l'énergie en un petit nombre de composants décorrélés. On ne considère que les coefficients supérieurs à un seuil prédéfini. Elles permettent aussi la séparation facile de la moyenne et une classification simple. Il existe plusieurs transformations :

- Karhunen Loève,
- Haar,
- Hadamar ,
- Fourrier,
- Cosinus Discrète [AHM-74],
- les Ondelettes.

Théoriquement, la transformation de Karhunen Loève est optimale mais peu utilisée à cause de sa complexité de mise en oeuvre. Tandis que la transformée en Cosinus Discrète (TCD) à deux dimensions est la plus utilisée pour sa facilité d'implémentation. De plus, ses performances sont proches de celles de Karhunen Loève [MAS-86].

La Transformée en Cosinus Discrète (T.C.D.):

La Transformée en Cosinus Discrète TCD (transformée orthogonale de type Fourrier) [AHM-74], est une opération linéaire qui consiste à transformer le plan image dans un plan fréquentiel de manière à obtenir des données décorrélées.
Cette transformation ne s'applique pas sur toute l'image, mais sur des blocs de taille limitée NxN pixels.
Pour chaque bloc de NxN pixels on calcule la T.C.D. qui est donnée par la formule suivante :

Pour u et v de 0 à N-1 :

$$F(u,v) = \frac{4.c(u)c(v)}{N^2}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\cos\frac{(2x+1)u\pi}{2N}.\cos\frac{(2y+1)v\pi}{2N}$$

$c(u) = 1/\sqrt{2}$ si u = 0
$c(u) = 1$ si u = 1, 2....N-1.

f(x,y) représente la luminance du pixel au point (x,y) dans l'image.
F(u,v) représente le coefficient transformé au point (u,v).

Pour chaque pixel d'un bloc image 8x8, on calcule à l'aide de cette fonction un nombre qui sera l'élément de la matrice résultat : la matrice des coefficients. La TCD d'un bloc donné représente les fréquences spatiales. Ainsi un bloc uniforme donc de fréquences spatiales faibles donne un bloc de coefficients de la TCD dont l'essentiel est le coefficient F(0,0) soit la moyenne des luminances du blocs. Les autres coefficients sont proches de zéro.

Avantages:

- une conservation de l'énergie (loi de Parceval),
- une concentration de l'énergie totale en moyenne statistique sur un nombre de coefficients inférieur à celui du domaine transformé,
- une diminution de la taille des vecteurs à quantifier,
- une décorrélation des signaux.

3.3 - Le Codage Sans Perte

C'est un codage statistique réversible car, après décompression on retrouve l'image originale. Il permet d'optimiser le codage en vue de compresser les données sans pour autant perdre de l'information. Si on considère une image numérisée représentée sur 256 niveaux de gris (8 bits). Soit {pi }, la probabilité d'apparition du niveau de gris i (i = 0 à 255) dans l'image. Le nombre de bits moyen nécessaire par échantillon pour coder cette image sans perdre de l'information peut être mesuré par l'entropie H [RIC-86]

$$H = -\sum_{i=0}^{255} p_i \log_2 p_i$$

Dans une image codée sur 8 bits par pixel, l'entropie est généralement de 4 à 6 bits par pixel. Il est possible de comprimer les images sans perte d'information mais le taux de compression est généralement faible (≈ 2).
Les méthodes prédictives de compression sans perte éliminent la redondance entre deux pixels voisins en ne codant que la partie non prévisible ou erreur de prédiction. Le nombre de bits alloués au codage est inversement proportionnel à sa probabilité (Codage d'Huffmann).

3.3.1 Le codage d'Huffmann

C'est un codage entropique, il tient compte de la fréquence d'apparition des symboles pour déterminer le code.
On assigne des mots de code courts aux niveaux de gris les plus fréquents, et des mots de code longs au niveaux de gris les moins fréquents.

Nous allons l'illustrer par l'exemple ci-dessous:
Supposons qu'une image soit composée des valeurs de luminance présentées par le tableau suivant:

Luminance	Répartition %
0	11
1	30
2	40
3	14
4	5

p0 = 0,11
p1 = 0,30
p2 = 0,40
p3 = 0,11
p4 = 0,05
pk = 0 pour $5 \leq k \leq 255$

Tableau 1 : Répartition des luminances

Le principe de la construction l'arbre de Huffmann (Fig. 3) consiste à constituer d'abord une liste contenant les probabilités d'apparition de chaque niveau de gris (Tableau 1). Puis on extrait à chaque étape, les deux valeurs les plus faibles de la liste. On en fait la somme pour constituer une nouvelle liste qui est le point de départ de l'étape suivante.

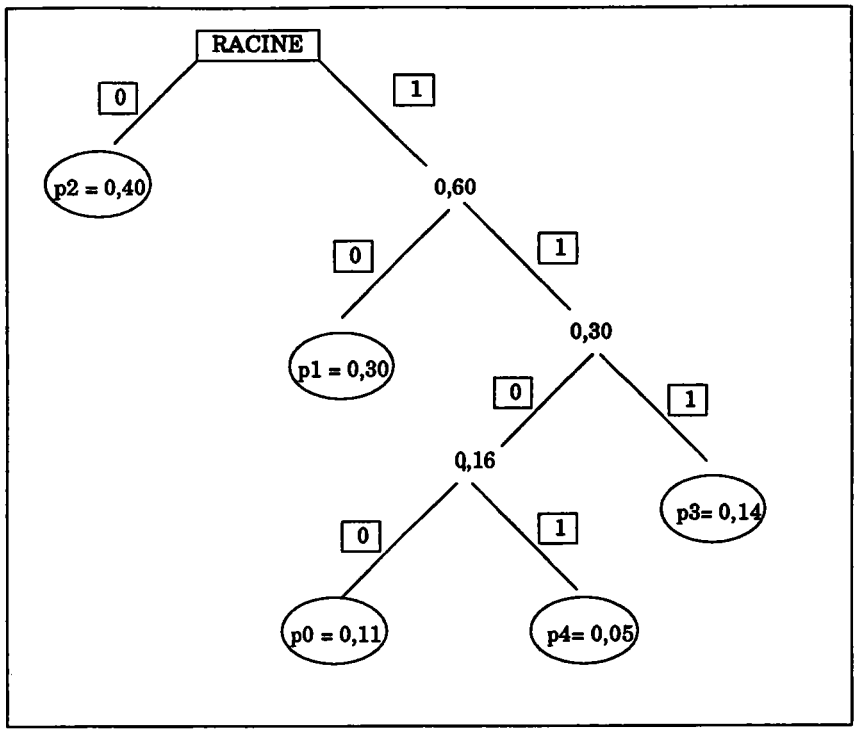


Fig.3 Arbre de codage de Huffmann

La table de Huffmann (Tableau 3) est construite en partant de la dernière suite et en affectant le code 0 à chaque branche de gauche et 1 à chaque branche de droite.

Luminance	Mot de code
0	1100
1	10
2	0
3	111
4	1101

Tableau 3: Codes de Huffmann

Le décodage des mots de code se fait sans problème dès l'instant que le décodeur connaît l'arbre de construction. Dans le cas d'une séquence d'images , il faudra construire une table contenant le code de chaque image. Si la séquence est très variée, le gain en débit sera négligeable.

3.3.2 LEMPEL-ZIW (1978)

Le but est de coder une chaîne de symboles de longueur variable par des codes de longueur variable (ou fixe), les chaînes longues contenant les symboles fréquents. La table de codage est construite au fur et à mesure [MAC-90].
L' algorithme adaptatif de mise à jour est peu commode.
Il limite la taille de la table de codage (mémoire + complexité).
Il a une mauvaise performance sur les messages courts.

3.3.3 Codage Arithmétique

Le codage arithmétique ne regroupe pas les symboles d'entrée en blocs/chaînes comme dans le cas du codage d'Huffmann ou de Lempel-Ziw. Il y a une subdivision successive de l'intervalle unité selon la partition p(0), p(1). L'intérêt est qu'il est facile de faire varier p(0), p(1) d'où une possibilité de changer la modélisation. En général il offre un bon taux de compression mais est plus complexe à réaliser.

4 - Méthodes de parallélisation d'un algorithme de codage d'image

Le parallélisme offre des structures d'architectures très performantes pour réduire le temps de traitement d'une application en traitement d'images. Le processeur de base que nous utiliserons sera le transputer, un microprocesseur de type RISC adapté au parallélisme (voir Annexe). Pour optimiser les calculs sur une application donnée, on ne peut plus se contenter d'une architecture fixe , c'est pourquoi la tendance actuelle est l'architecture reconfigurable telle celle du supernode au moyen de commutateurs programmables.
Une autre approche permet d'envisager un découpage différent tel que les transputers effectuent des mêmes opérations sur des zones différentes de l'image.
Mais dans les deux approches se pose le problème de la communication des données: les vitesses des liens de communication des données sont limitées, et il n'est pas possible non plus d'envisager l'accès à une mémoire commune puisqu'il n'en existe pas. Ce qui nous a conduit dans un premier temps à évaluer les possibilités de transmissions et leur efficacité puis définir des modèles de parallélisation adaptés à notre algorithme et au processeur.

4.1 Présentation générale de l'algorithme

Une image possède en général une grande redondance, son stockage nécessite beaucoup de place mémoire et sa transmission nécessite beaucoup de temps.
Pour réduire la quantité de l'information transmise par l'intermédiaire d'un canal de transmission d'images numérisées, il est nécessaire de comprimer les

images transmises. La réduction de débit peut affecter la qualité de l'image. Mais de puissantes méthodes de compression d'images numérisées qui n'affectent pas trop la qualité de l'image ont été développées. C'est dans ce cadre qu'un algorithme de codage d'images à bas débit (Fig.4) a été mis au point au CERLOR (Centre d'Etudes et de Recherche de Lorraine) [BAI-92].

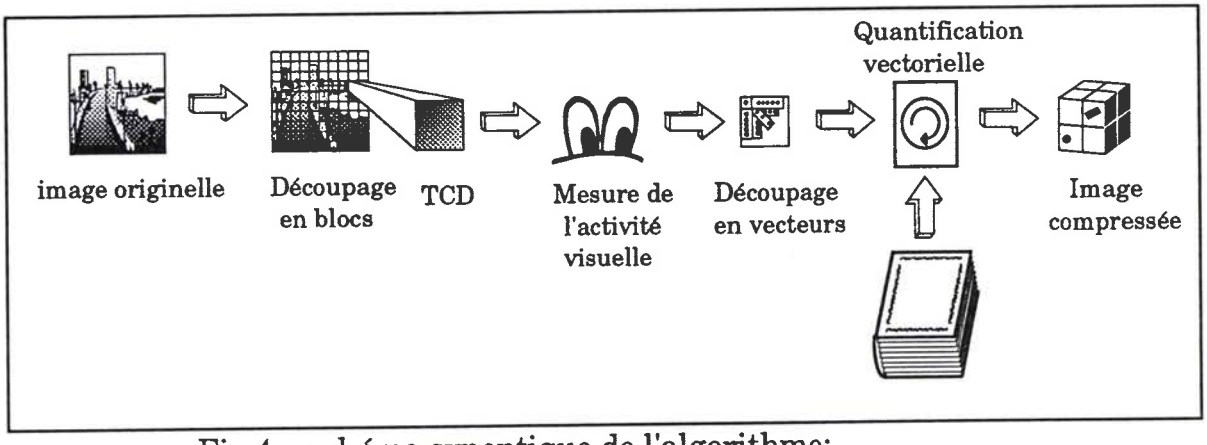


Fig 4 : schéma synoptique de l'algorithme:

Cet algorithme est fondé sur la reconnaissance et la reconstitution des évolutions des luminances dans une image.L'algorithme peut être décrit comme suit:

-***Un découpage en blocs:*** une image numérique est découpée en blocs 8 x 8 pixels.

- ***Transformée en Cosinus Discrète (T.C.D.):*** cette opération transforme ces blocs pixels 8 x 8 en blocs 8 x 8 dans le plan fréquentiel.

- ***Une classification:*** le but de la classification est de séparer les blocs suivant des critères subjectifs liés à la perception visuelle afin d'améliorer la qualité de l'image codée. Cette identification se fait en comparant les paramètres de classification issus de ces traitements à des seuils obtenus expérimentalement. Une série de tests de seuils est effectuée sur une image donnée avant de choisir les seuils définitifs avec lesquels l'image sera codée. Pour un bloc, il s'agit de déterminer à quelle classe spatiale il appartient. La recherche se fait en comparant les valeurs des paramètres du bloc à des seuils fixés au moins pour une image donnée. C'est une recherche optimale car la procédure de comparaison s'arrête dès qu' on trouve qu'un bloc appartient à un premier groupe.

- ***Une extraction des vecteurs:*** pour chaque bloc, on extrait des parties du bloc qu'on appelle vecteurs (un vecteur est une suite finie de coefficients représentatifs successifs extraits d'un bloc de coefficients) et qu'on juge suffisantes pour coder le bloc sans trop affecter la qualité de l'image. Des

vecteurs de taille 5 ou 3 seront retenus pour le codage (Fig. 5).

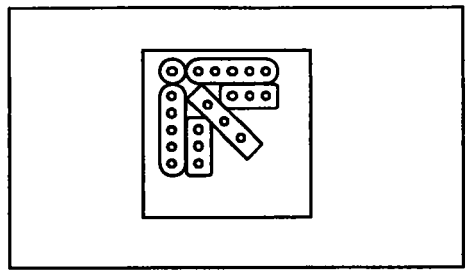


Fig 5: Formation des vecteurs de coefficients transformés

- ***la quantification vectorielle :*** Elle consiste à coder un vecteur de coefficients transformés obtenus par la classification par l'adresse du vecteur du dictionnaire préétabli, qui est le plus proche. La recherche du vecteur du dictionnaire le plus proche peut être effectuée par recherche séquentielle exhaustive ou par recherche optimale à partir de la distance entre les vecteurs. La taille du vecteur à coder est évidemment la même que celle des vecteurs du dictionnaire.

4.2 Analyse de l'algorithme

- Cet algorithme sert à coder des images de faible qualité ,

Par exemple pour transmettre en une seconde une image noir et blanc de de 256 lignes de 256 pixels sur 256 niveaux de gris on obtient

- chaque image contient 1024 blocs de 8*8 pixels,
- la T.C.D. par bloc nécessite 1024 multiplications et 896 additions,
- le calcul du type de bloc nécessite 52 multiplications et 59 additions par bloc,
- le calcul de la quantification vectorielle est étroitement lié à la taille des dictionnaires.

Le calcul de la distance entre 2 vecteurs de c composantes demande 2c-1 additions et c multiplications.

Le calcul de la quantification d'un vecteur par un dictionnaire de d vecteurs demande d*(2c-1) additions et c*d multiplications. Une qualité visuelle bonne est obtenue avec des dictionnaires de d=256 vecteurs.

Le nombre de vecteurs à quantifier vectoriellement ainsi que leurs tailles dépendent très fortement du contenu de l'image. Cependant en première estimation, chaque bloc nécessite le codage de 1,1 vecteurs de longueur 5 et de 0,65 vecteurs de longueur 3. Par seconde, la quantification vectorielle demande donc 1,95 millions de multiplications et 3,45 millions d'additions

	Millions d'additions par seconde	Millions de multiplications par seconde
T.C.D.	0,92	1,05
Calcul du type de bloc	0,06	0,05
Q.V.	3,45	1,95
TOTAL	4,43	3,05

Tableau 3 : Statistiques des calculs dans l'image

4.3 Traitement mono-transputer

Un IBM PC 386, équipé d'un coprocesseur arithmétique, cadencé à 25MHz est équipé d'une carte INMOS (B008) qui permet de commander 10 Transputers T800; chaque Transputer a une vitesse de liens de 20Mb/sec. Le langage de programmation est le C (MicroSoft version 5) sous MS-DOS et le C Toolset d'Inmos compatible à la norme ANSI pour les Transputers.

L'algorithme a été d'abord porté sur un seul transputer. Pour une image noir et blanc de de 256 lignes de 256 pixels sur 256 niveaux de gris on obtient

Taille Dico	**16**	**64**	**256**
Découpe (sec)	0,30	0,30	0,30
TCD (sec)	14,08	14,08	14,08
Classification (sec)	0,74	0,74	0,74
Quantification (sec)	6,5	25,19	97,82
Tps Total (sec)	20,87	39,56	112,189

Tableau 4: Différents résultats

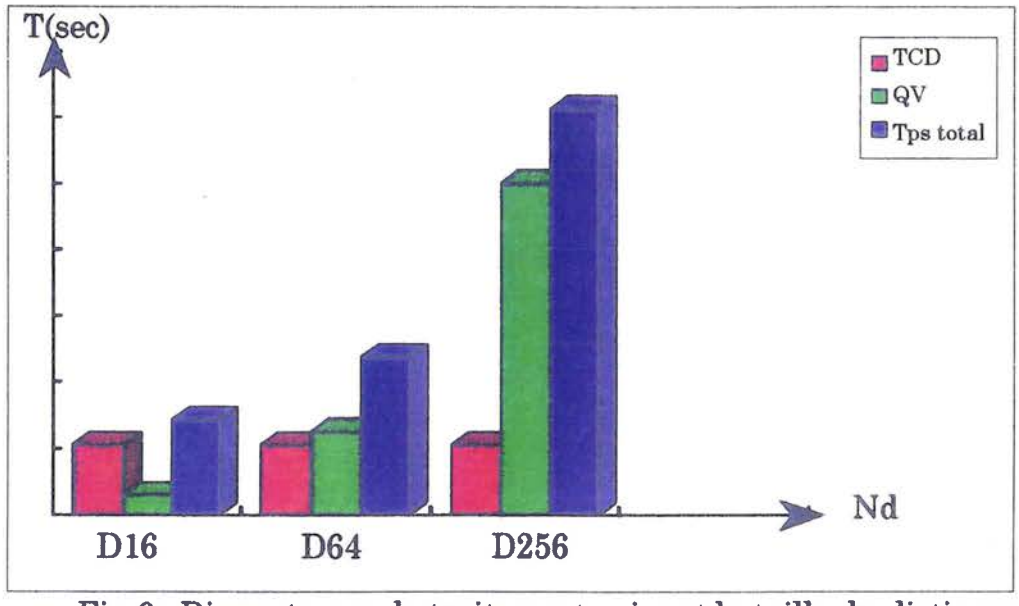


Fig 6 : Divers temps de traitement suivant la taille du dictionnaire

Avec:

- Nd la taille du dictionnaire,
- T(sec), le temps de traitement en secondes.

L'analyse de ce graphe permet de tirer les conclusions suivantes:

- Premièrement, le temps de découpage de l'image est constant et négligeable pour n'importe quelle taille du dictionnaire.

- Deuxièmement l'opération TCD est constante, ce qui est logique car elle se fait sur toute l'image.

- Enfin, l'opération quantification vectorielle est très coûteuse en temps de calcul car elle nécessite une recherche exhaustive sur tout le dictionnaire ce qui fait appel à de nombreux calculs complexes. Plus la taille du dictionnaire augmente, plus le temps de traitement de la quantification vectorielle augmente.

On peut déterminer le temps de calcul moyen N d'un vecteur du dictionnaire

$$\begin{cases} k + 16N = 285748 \\ k + 64N = 477579 \\ k + 256N = 1227359 \end{cases} \Rightarrow N = 3915 cycles = 0{,}25 \text{ sec}$$

En tenant compte de tous ces résultats nous allons essayer de trouver une technique de parallélisation adaptée à l'algorithme. Le flot d'entrée de l'algorithme sera de type image et le flot de sortie un tableau d'entiers obtenu par codage.

4.4 Parallélisation logicielle

Le fait de disposer d'un réseau de transputers modifie radicalement l'approche du problème. Pour exploiter la puissance offerte par le réseau de transputers, il est nécessaire non seulement de "penser" l'application en tâches indépendantes qui doivent pouvoir s'exécuter chacune sur n'importe lequel des modules dont on dispose, mais d'indiquer au système quel process doit se dérouler sur quel transputer, et avec quels canaux il doit communiquer avec les autres process.

L'algorithme a été mis en oeuvre sur un réseau de transputers T800. Diverses méthodes de parallélisation ont été testées sur différentes configurations de réseaux de transputers car, le transputer avec ses quatre liens permet toutes sortes d'architectures, aussi bien de type MIMD que de type SIMD.

4.4.1 Parallélisation par les tâches

Structure en pipe-line: C'est l'approche la plus répandue du parallélisme. Son principe repose sur la décomposition du travail, en général répétitif, qui doit être effectué en tâches élémentaires sur chaque transputer. L'image sera divisée en blocs et chaque transputer effectue successivement un traitement différent sur chaque bloc. Les transputers travaillent ici sur des blocs différents. Dans le cas d'un réseau de 3 transputers, il est permis, par exemple d'envisager un découpage qui assurerait le calcul de la TCD sur le transputer T1 pendant que le transputer T2 calcule l'activité visuelle du bloc TCD reçu de T1 et que le transputer T3 effectue la quantification vectorielle des classes de vecteurs reçues de T2 et envoie le résultat à T1.

Mais, pour avoir une parallélisation efficace, il faut effectuer un bon partitionnement des données, ce qui revient à définir la taille optimale des blocs à envoyer dans le réseau.

- Une première solution consiste à partitionner l'image en blocs 8*8 et les propager à travers le réseau un à un. Les résultats sont les suivants

Taille Dico	16	64	256
Tps sur T1 (sec)	14,88	14,88	14,88
Tps sur T2 (sec)	0,40	0,40	0,40
Tps sur T3 (sec)	6,80	26	98
Tps Total (sec)	15,01	26,78	98,97
Tseq (sec)	20,57	39,56	112,19
Accélération	1,37	1,47	1,13
Efficacité (%)	46	49	37

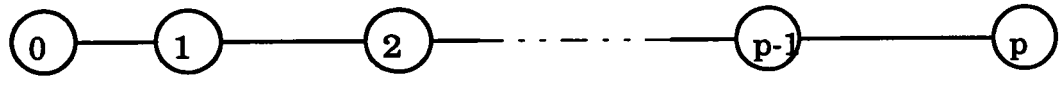
Tableau 5: Différents résultats

Temps d'une communication entre 2 processeurs voisins	0,079 sec/octet
Nombre de communications entre 2 processeurs voisins	1922
Nombre total de communications	3844
Temps total des communications	1,23 sec

Tableau 6: statistique des communications

- L'autre solution consiste à déterminer un partitionnement efficace de l'image en vue de déterminer le nombre de paquets optimal.

Supposons qu'on veut envoyer un message de longueur L sur une chaîne de (p+1) processeurs (de 0 à p)



le temps de communication Tcom sera de:

$$Tcom = p(\beta + L\delta)$$

avec β le temps d'initialisation du lien en secondes,
δ le débit du canal en octets/sec,
et $L\delta$ le temps de propagation.
Si on veut envoyer le message par paquets de taille L/n

$$Tcom(n) = p(\beta + \frac{L}{n}\delta) + (n-1)(\beta + \frac{L}{n}\delta)$$

$$Tcom(n) = p(\beta + \frac{L}{n}\delta) + (n-1)(\beta + \frac{L}{n}\delta)$$

Pour obtenir le nombre de paquets optimal , il suffit de dériver cette fonction par rapport à n

$$\frac{dTcom(n)}{dn} = \beta - (p-1)\frac{L}{n^2}\delta$$

Le nombre de paquets optimal est :

$$n_{opt} = \sqrt{\frac{(p-1)L\delta}{\beta}}$$

Plus les processeurs sont nombreux, plus il est intéressant d'avoir des paquets longs.
Le temps de communication optimal est:

$$Tcom_{opt} = \left(\sqrt{L\delta} + \sqrt{(p-1)\beta}\right)^2$$

Dans notre cas : δ = 1,1µs , β = 5µs ; p=3; $n_{opt} = 198$
Donc on envoie par le réseau des paquets 192 octets c'est à dire 3 paquets de 64 octets. Les résultats obtenus sont les suivant:

Taille Dico	16	64	256
Efficacité(%)	81	78	79

Les résultats sont satisfaisants et le nombre de communications dans le réseau est réduit d'un facteur 3.

Le pipeline présente de nombreux avantages:
- un enchaînement correct des opérations élémentaires qui correspondent aux différents étages du pipe,
- une facilité de mise en oeuvre,
- il est bien adapté aux images discrètisées,
- il se prête surtout aux algorithmes séquentiels.

Mais ses inconvénients sont dûs au nombre de communications élevé, de plus il n'est pas adapté à tous les algorithmes.

4.4.2 Parallélisation par les données

L'image est divisée en plusieurs parties et chaque transputer exécute le même algorithme sur une partie. C'est la méthode SPMD. Les transputers peuvent communiquer entre eux s'ils doivent s'échanger des données ou des résultats. L'intérêt de cette architecture, est que comme les communications se font par message et que chaque transputer possède sa propre mémoire, on peut facilement envisager l'extension de l'architecture à un grand nombre de transputers.

Nous disposerons de:

- un transputer maître T800 avec 2 M octets de RAM,

- 6 transputers de travail T800 avec 1 M octets chacun,

- une station IBM-PC connectée au transputer maître.

Le PC joue ici uniquement le rôle de serveur de fichiers. Le transputer maître ne participe pas aux traitements. Il est utilisé pour piloter les autres transputers, c'est lui qui distribue les images aux transputers de travail et range les images résultats en provenance de ces mêmes transputers.
Le nombre de communications dépend de la découpe de l'image traitée mais aussi du nombre de transputers. Le temps de traitement sur chaque transputer dépendra aussi de la taille et surtout de la nature de la partie de l'image qu'il doit traiter.
Nous adopterons des architectures (anneau, arbre binaire) où il est possible de découper l'image en sous ensembles sur lesquels les calculs peuvent être réalisés de manière indépendante tout en minimisant le taux des communications.

Dans la suite, les valeurs suivantes seront considérées:

- Nt : nombre de transputers,
- Nd : nombre de vecteurs du dictionnaire,
-Tseq : temps de traitement séquentiel sur un seul processeur en secondes,
- Tpar : temps de traitement parallèle sur Nt transputers en secondes,
- Eff : efficacité en pourcentage,
- Acc : accélération.

4.4.2.1 Structure en Anneau

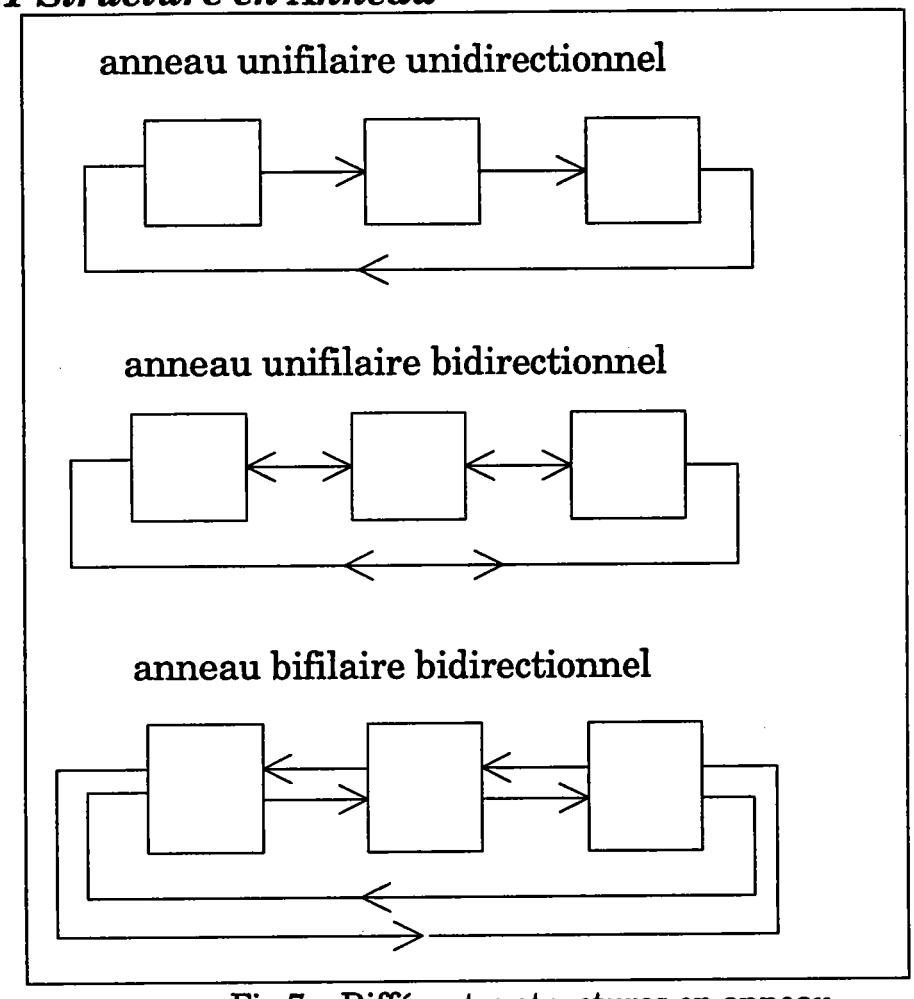


Fig 7 : Différentes structures en anneau.

Dans un anneau, chaque processeur a 2 voisins immédiats. L'anneau peut être unidirectionnel, unidirectionnel unifilaire ou bifilaire(Fig. 7)

Une structure d'anneau bifilaire unidirectionnel a été implémentée sur 2, 3, 4, 5 transputers T800. Le maître assure le découpage de l'image en parties et l'envoie aux processeurs soit directement ou via d'autres processeurs. La réception des parties résultats s'effectue de la même manière.

Temps d'une communication entre 2 transputers voisins	0,079sec/octet
Nombre de communications entre 2 transputers voisins	2
Nombre total de communications	6
Temps total des communications	1,171 sec

Tableau 7 : Structures des Communications dans le cas d'un réseau à 3 transputers.

	Nd =	**16**		
Nt	2	3	4	5
Tpar	11,22	7,22	5,84	4,79
Tseq	20,87	20,87	20,87	20,87
Eff	94	96	89	87
Acc	1,86	2,89	3,57	4,35
	Nd =	**64**		
Nt	2	3	4	5
Tpar	21,15	13,88	11,43	9,29
Tseq	39,56	39,56	39,56	39,56
Eff	93	95	86	85
Acc	1,87	2,86	3,46	4,26
	Nd =	**256**		
Nt	2	3	4	5
Tpar	59,04	42,98	35,05	29,84
Tseq	112,19	112,19	112,19	112,19
Eff	95	87	80	75
Acc	1,90	2,61	3,20	3,76

Tableau 8: Résultats des simulations

Nt	2	3	4	5
Acc16	1,86	2,89	3,57	4,35
Acc64	1,87	2,86	3,46	4,26
Acc256	1,90	2,61	3,20	3,76

Tableau 9 : Accélération en fonction du nombre de transputers

Nt	2	3	4	5
Eff16	94	96	89	87
Eff64	93	95	86	85
Eff256	95	87	80	75

Tableau 10 : Efficacité en fonction du nombre de transputers

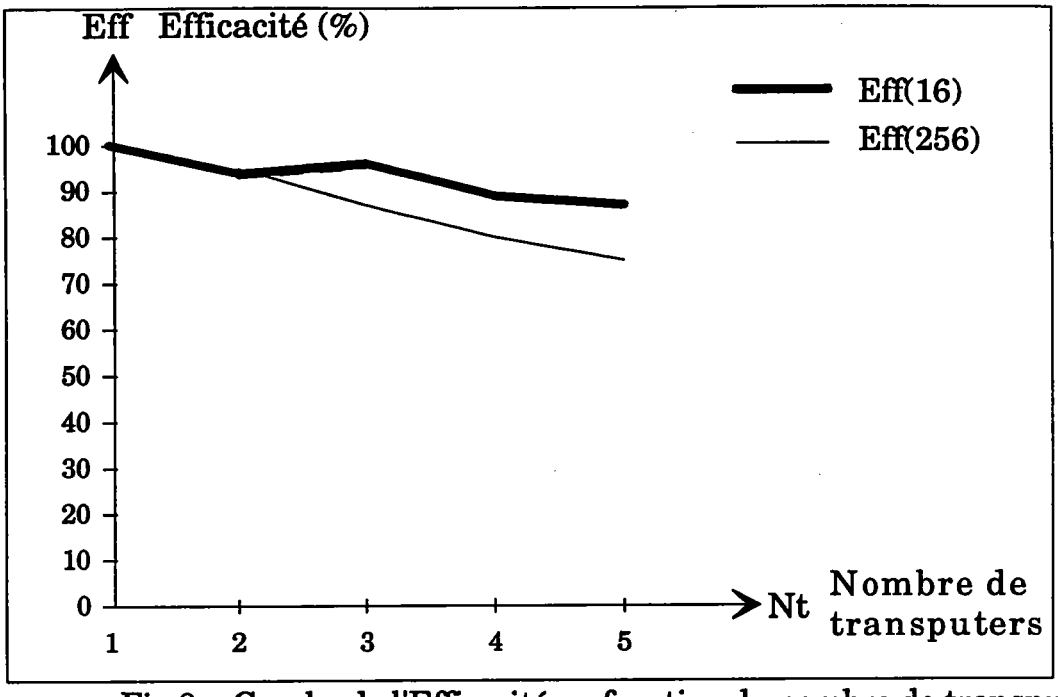


Fig 8 : Courbe de l'Efficacité en fonction du nombre de transputers

4.4.2.2 Structures en Arbre

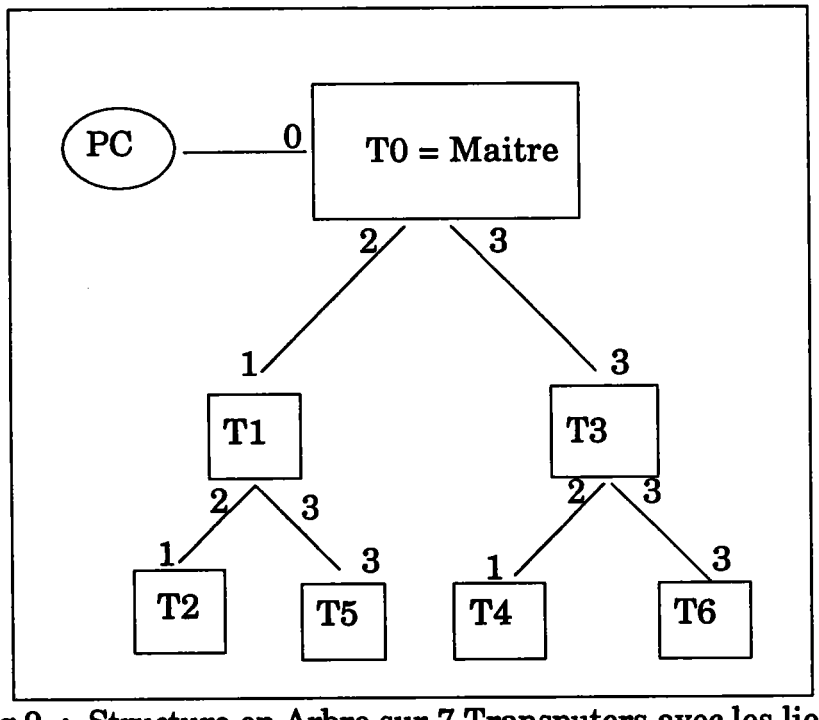


Fig 9 : Structure en Arbre sur 7 Transputers avec les liens.

Temps d'une communication entre 2 transputers voisins	0,079sec/octet
Nombre de communications entre 2 transputers voisins	2
Nombre total de communications	12
Temps total des communications	1,698 sec

Tableau 11: Structure des communications dans le cas d'une structure d'arbre à 7 transputers

	Nd =	**16**		
Nt	2	3	4	5
Tpar	11,59	7,78	6,24	5,14
Tseq	20,87	20,87	20,87	20,87
Eff	90	89	83	81
Acc	1,80	2,68	3,34	4,06
	Nd =	**64**		
Nt	2	3	4	5
Tpar	21,04	15,63	11,77	9,99
Tseq	39,56	39,56	39,56	39,56
Eff	94	84	84	79
Acc	1,88	2,53	3,36	3,96
	Nd =	**256**		
Nt	2	3	4	5
Tpar	59,04	45,97	39,78	34,10
Tseq	112,19	112,19	112,19	112,19
Eff	95	81	70	66
Acc	1,90	2,44	2,82	3,29

Tableau 12: Résultats des simulations

Nt	2	3	4	5
Acc16	1,80	2,68	3,34	4,06
Acc64	1,88	2,53	3,36	3,96
Acc256	1,90	2,44	2,82	3,29

Tableau 13 : Résultats de l'accélération en fonction du nombre de transputers

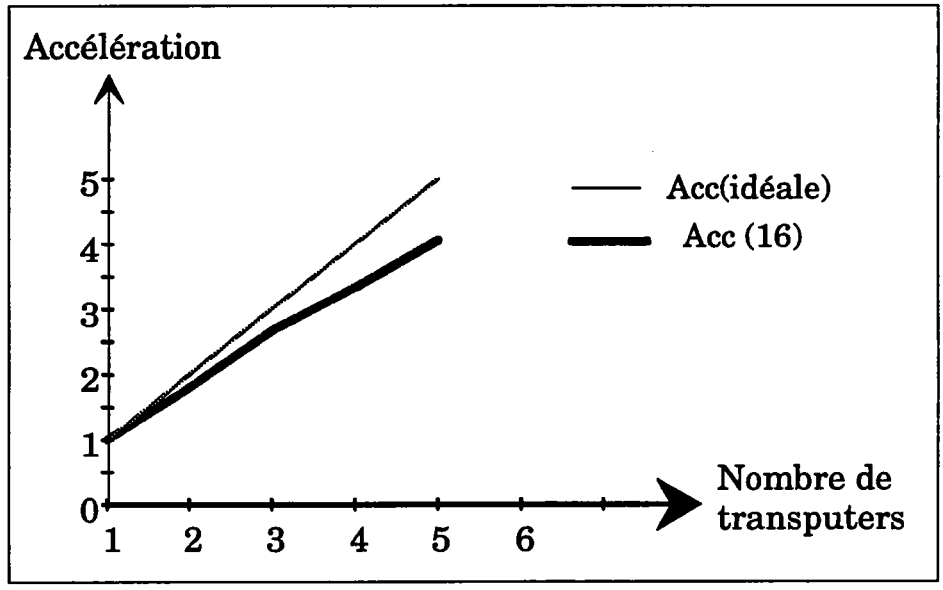


Fig 10 : Accélération en fonction du nombre de transputers

Les principaux avantages du SPMD sont:

- une bonne performance pour le système de communications:
  - moins de communications ,
  - les communications sont indépendantes les unes des autres,
- le modèle CSP (Communicating Sequential Process) du transputer permet:
- d'exploiter les différents types de parallélisme géométrique,
- une exploitation du parallélisme très facile,
- et une réorganisation automatique des données.

Mais, le découpage de l'image n'est jamais optimal car étant donné que le temps des calculs dépend de la nature de l'image à traiter et compte tenu qu'on a aucune information au préalable sur l'image, certains transputers n'atteindront jamais un taux d'activité maximal ce qui introduit des temps d'attente dans le réseau.

4.4.3 La ferme de processeurs (Farming)

Un maître "Fermier" distribue à ses esclaves les données sur lesquelles ils appliquent le traitement avant de lui rendre le résultats. L'algorithme peut être décrit comme suit :

- un découpage des données à traiter en blocs,
- le traitement de ces blocs sur les transputers disponibles,
- le rangement des blocs résultats (pour restituer un résultat cohérent).

4.4.3.1 Semi-farming (INMOS)

Dans notre étude le réseau global sera remplacé par une structure d'arbre, c'est à dire une vision locale au niveau de chaque Transputer. Chaque nœud de l'arbre voit ses Transputers "fils". Pour avoir une vision globale au niveau du maître, qui a besoin de savoir où envoyer ses paquets de données, il faut remonter l'information que connaît chacun des Transputers sur ses fils vers son père. Cela revient à dire que certains Transputers vont recevoir leur paquet de données d'un Transputer intermédiaire et non pas directement du maître. Les paquets résultats passent aussi par des Transputers intermédiaires pour arriver au maître (Fig. 11).

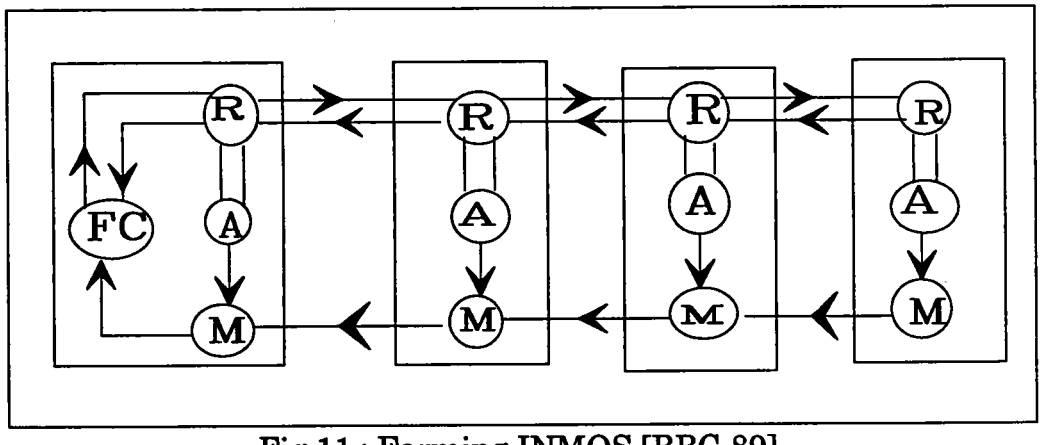


Fig 11 : Farming INMOS [BRC-89]

Cela nécessite un canal de communication pour les paquets de données.

Le Transputer nous offre deux possibilités:

-1) Multiplexage sur un seul canal, mais la taille importante des paquets résultats demande un temps assez grand au niveau des communications pour être transmis. Par conséquent des demandes de paquets de données peuvent être bloquées par la communication d'un paquet résultat, ce qui fait apparaître des temps d'attente au niveau des étages inférieurs de l'arbre.

-2) Allocation d'un lien supplémentaire par communication, mais un lien supplémentaire nous conduit à l'utilisation de deux liens par chemin de communication et le réseau sera réduit en un simple "pipe". Or un "pipe" est un arbre unaire; plus le degré de l'arbre est petit, plus on introduit des temps d'attente dans le réseau, et plus on fait donc chuter l'efficacité.

Processus Maître:
FC: Ce processus est composé de 2 sous processus qui travaillent en parallèle Envoi et Réception:

- Envoi divise l'image en blocs et les envoie lorsqu'il reçoit une demande du processus Rooter du 1er Transputer du réseau.

- Réception reçoit le bloc résultat provenant du processus Mergers des autres Transputers via le 1er Transputer.

Processus Esclaves:
Les Transputers "esclaves" sont composés de trois processus R, A et M:
R = Rooter: demande un bloc au processus Rooter du Transputer précédant en envoyant un octet sur le canal réservé à cet effet. Lorsqu'il obtient un paquet il attend une demande provenant du Transputer suivant ou de l'algorithme qu'il dessert.
A = Algorithme: traite le bloc après l'avoir demandé à son Rooter puis envoie le résultat à son Merger. A noter qu'il envoie au début et à la fin de son traitement un message à son Rooter R pour lui indiquer s'il a fini ou non .
M = Merger: reçoit le bloc résultat provenant de son Algorithme et du Merger du Transputer suivant pour les passer au Merger du Transputer précédent.
Les processus R, A, M tournent en boucle infinie.
Les processus RF, MF sont des versions simplifiées de R et M pour le dernier Transputer (fond de la ferme).

Les mesures suivantes ne tiennent pas compte du temps de rangement des blocs résultats.

	Nd =	**16**		
Nt	2	3	4	5
Tpar	11,66	7,84	6,17	5,24
Tseq	20,87	20,87	20,87	20,87
Eff	89	88	85	79
Acc	1,79	2,66	3,38	3,98
	Nd =	**64**		
Nt	2	3	4	5
Tpar	21,85	15,21	11,70	10,35
Tseq	39,56	39,56	39,56	39,56
Eff	90	86	84	77
Acc	1,81	2,60	3,38	3,82

Tableau 14: Résultats des simulations

4.4.3.2 Farming en arbre binaire

Les mesures ne tiennent pas compte du temps de rangement de l'image résultat.

	Nd =	**16**		
Nt	2	3	4	5
Tpar	11,40	7,27	5,92	4,80
Tseq	20,87	20,87	20,87	20,87
Eff	91	95	88	86
Acc	1,83	2,89	3,52	4,34
	Nd =	**64**		
Nt	2	3	4	5
Tpar	20,30	14,08	11,40	9,22
Tseq	39,56	39,56	39,56	39,56
Eff	98	93	86	85
Acc	1,95	2,81	3,47	4,29

Tableau 15 : Résultats des simulations

Le farming en structure arbre binaire que nous avions développé donne de meilleurs résultats que celui d'Inmos de type pipeline. En effet en structure arbre binaire, le taux des communications est réduit, mais le problème majeur dans le farming en général pour le type de traitement que nous avions à traiter est le rangement des blocs résultats. Et dès que le nombre de transputers devient élevé, le réseau devient très lourd à gérer et on n'est pas à l'abri d'un blocage (dead lock)[BRC-89], c'est à dire qu'à partir d'un certain moment tous les transputers sont en attente. C'est un problème inévitable dans la gestion du farming.

4.4.4 Comparaison

La courbe ci-dessous représentant les accélérations moyennes pour les différentes méthodes adoptées

Nt	Acc(idéale)	Acc(SPMD)	Acc(Farm)	Acc(Pipe)
1	1	1	1	1
2	2	1,86	1,85	1,67
3	3	2,50	2,35	2,30
4	4	3,60	3,20	2,98
5	5	4,20	3,75	3,00

Tableau 16 : Accélération moyenne pour les différentes configurations .

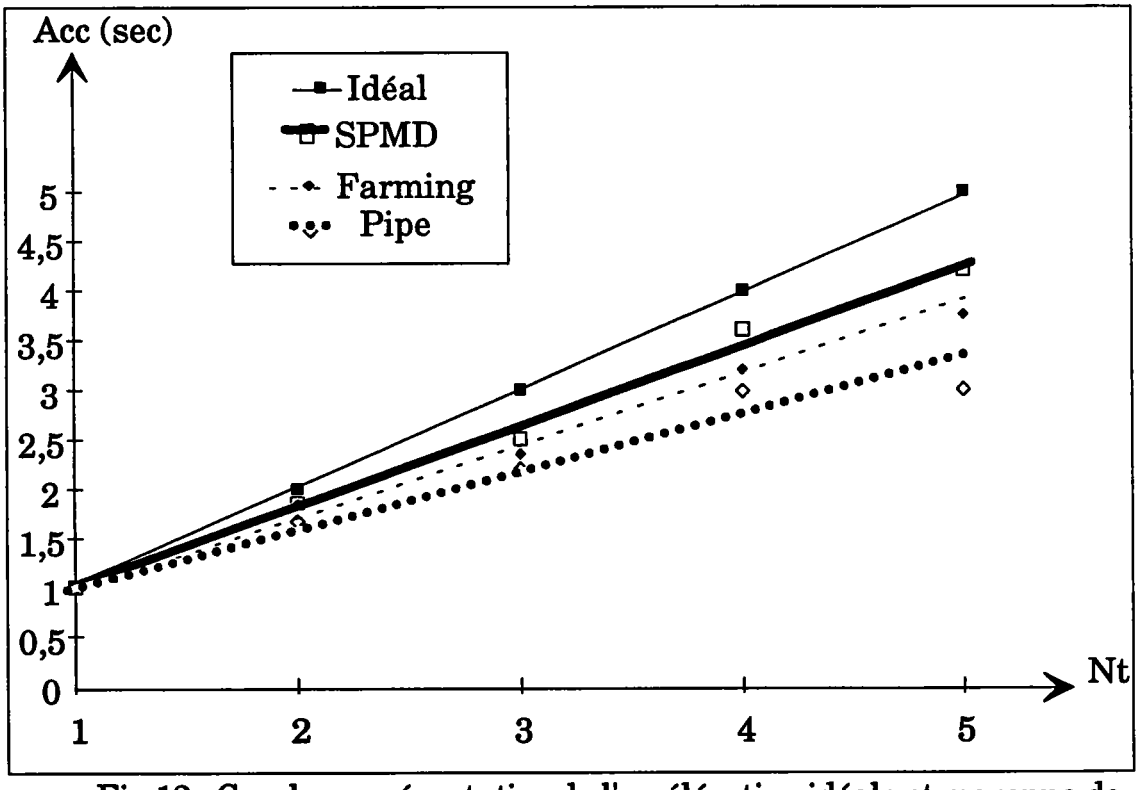


Fig 12 : Courbe représentative de l'accélération idéale et moyenne de notre système.

La courbe idéale n'est jamais atteinte dans un réseau car il y aura toujours des temps d'attente lors des communications.
Pour un nombre de Transputers élevé, la gestion des communications devient délicate et introduit des temps d'attente dans le réseau ce qui dégrade l'accélération, mais l'accélération suit une loi linéaire ce qui nous permet de dire que notre algorithme est stable.
Un réseau simple comportant peu de Transputers peut donner de bons résultats s'il travaille hors saturation avec un taux de communication faible.

Le partitionnement de tâches n'offre pas nécessairement de performances très élevées mais, basé sur un principe applicable à tous les niveaux, il permet des résultats honorables pour un nombre de transputers limité.
Le partitionnement de données qui est une certaine conception du pipe-line avec l'introduction de branches parallèles et de boucles de retour apporte des résultats significatifs mais conduit à des structures qui deviennent complexes et relativement figées. En revanche son champ d'application est plus vaste.
La limitation du nombre de liens disponibles du transputer nous a conduit à développer un type de farming en arbre binaire. Si on ne tient pas compte du rangement de l'image résultat, les performances semblent être honorables à condition d'éviter tous les phénomènes de blocages du réseaux ce qui est un problème très complexe à résoudre. Mais on est obligé en traitement d'images de trier les paquets résultats à l'arrivée pour les ranger dans un bon ordre.

5 Conclusion

Ces recherches ont fait l'objet d'une programmation parallèle SPMD et Farming en structures pipeline, anneau, arbre binaire et en farming à l'aide d'un réseau de 1, 2, 3, 4 ou 5 transputers T800. En cas d'utilisation de plusieurs transputers dans le cas de la programmation SPMD, l'image est découpée en autant de parties, dont les tailles peuvent être modulées en vue d'accélérer le traitement. Les méthodes utilisées sont aisément extensibles à d'autres processeurs.

La première conclusion de ces essais est que la gestion du parallélisme n'est pas évidente.
Deuxièmement, des outils généraux de surveillance font cruellement défaut.
Troisièmement, les transputers sont technologiquement dépassés, certains D.S.P. ou nouveaux processeurs en technologie submicronique affichent des performances au moins 10 fois meilleures.
Quatrièmement, le monde du transputer est trop petit pour offrir un support de bon niveau.
Enfin un constat est qu'il sera difficile d'atteindre nos objectifs dans le cadre

d'un traitement parallèle à faible coût compte tenu des performances technologiques du transputer T800 et de la stabilité de notre algorithme. Ainsi nous allons remplacer les parties logicielles les plus consommatrices en temps de calcul par des cartes spécialisées.

BIBLIOGRAPHIE

[AHM-74] N.Ahmed, T Natarqjan, K.R. Rao, "Discrète Cosine Transform", IEEE Trans. On Comm, Vol COM. -22, N°1, pp. 90-93, January 1974.

[BAI-92] J. Baina, "Codage hybride adaptatif d'images en vue de la transmission à bas débit" Thèse d'Université, Université de Metz, Janvier 1992.

[BAR-92] " La compression: une nécessité pour les images numériques", Electronique N°14, janvier 1992, pp. 46-54.

[BRC-89] N. Breton et J. Choain, "Farming et Transputers", Document interne TEKELEC Inmos ,juillet 1989.

[DUB-91] H. Dubois, " Analyse des systèmes multiprocesseurs: application à la mise en oeuvre sous contraintes d'algorithmes de traitement d'images" thèse d'Université, Université de Rennes I, Janvier 1991.

[HUD-92] O. Hudry, " Parallélisme et complexité algorithmique en optimisation combinatoire.", L'écho des RECHERCHES N°50, pp.13-23Analyse, 4ème trimestre 1992.

[KER-91] E. de Kervadec, " Comparatif : La compression de données clé de la performance ?, Soft & micro, pp.131-138, Janvier 1991.

[LBG-80] Y. Linde, A. Buzo and R.M. Gray, "An algorithm for vector quantizer design", IEEE Trans. Commun., vol COM-28, Jan, pp.84-85.

[MAC-90] F. Marchal, " Compression Lempel-Ziv-Welsh" Note techhnique N° 43 niveau 3, Mars 1990.

[MAR-86] J.P Marescq, "Etude de schémas de quantification vectorielle adaptative multiclasses. Application au codage de séquences d'images télévisuelles "Thèse de Docteur-Ingénieur, Université de Rennes I, Dec 86.

[NAS-88] N.M. Nasrabadi and R. A. King, "Image coding using vector quantization - A review, IEEE Trans Commun., vol. 36, N°8, August 1988, pp 957-971.

[PET-85] PETER H. Westerink, E. Dick, Boekee et J. Biemond "Subband Coding of Images Using vector Quantization ".

[RIC-86] C. Richard " Compression du Signal Vidéo " Techniques de

l'ingénieur E5500 pp.1-9, Juin 1986.

[SAR-89] V. Sarkar, "Partitioning and scheduling parallel programs for multiprocessors", Edition Pitman, 1989.

[SEC-93] M. Seck, A. Laprévote et A. Tosser-Roussey " Parallélisation d'un algorithme de codage d'images à bas débit sur réseau de Transputers ", Quatorzième Colloque sur le Traitement du Signal et des Images, GRETSI Acte II pp.1047-1050, Juan les Pins , Septembre 1993.

[SEC-92] M. Seck, "Architectures parallèles et traitement d'imges", Rapport de Recherche , Document interne , TDF-C2R, Décembre 1992.

[UBE-93] S. Ubeda " Algorithmes d'amincissement d'images sur machines parallèles" Thèse d'Université, Université Claude-Bernard - Lyon I, Février 1993.

[VIN-88] P. Vincent "Placement de processus sur un réseau de Transputers", Journée Firtech Architectures futures, 9-10 Novembre 1988

[WAC-91] Waye E. Carlsons, " A survey of computer graphics image encoding and storage formats", Computer Graphics. vol 25, N°2, pp.67-78, Avril 1991.

[ZAO-89] Y. Zhao "Etude et réalisation d'un système de transmission d'images numériques à bas débit", Thèse d'Université, Nancy I, Oct 89.

Chapitre III

REALISATION MATERIELLE

1 Introduction

Afin d'accélérer la vitesse de traitement pour atteindre nos objectifs, les parties logicielles critiques de l'algorithme ont été remplacées par des cartes spécialisées. C'est dans cette optique que l'on a développé une carte prototype TCD et un quantificateur vectoriel. Cette partie du système de traitement faisant l'objet de dépôts de brevets, certains détails techniques ne seront pas développés.

2 La Carte prototype TCD

Pour accélérer le traitement de la TCD, une carte prototype pouvant fonctionner dans un environnement parallèle a été mise au point à TDF-C2R. Elle intègre un circuit INMOS : le A121, un DSP (Digital Signal Processor) spécialisé pour la TCD et la TCD inverse.

2.1 Description

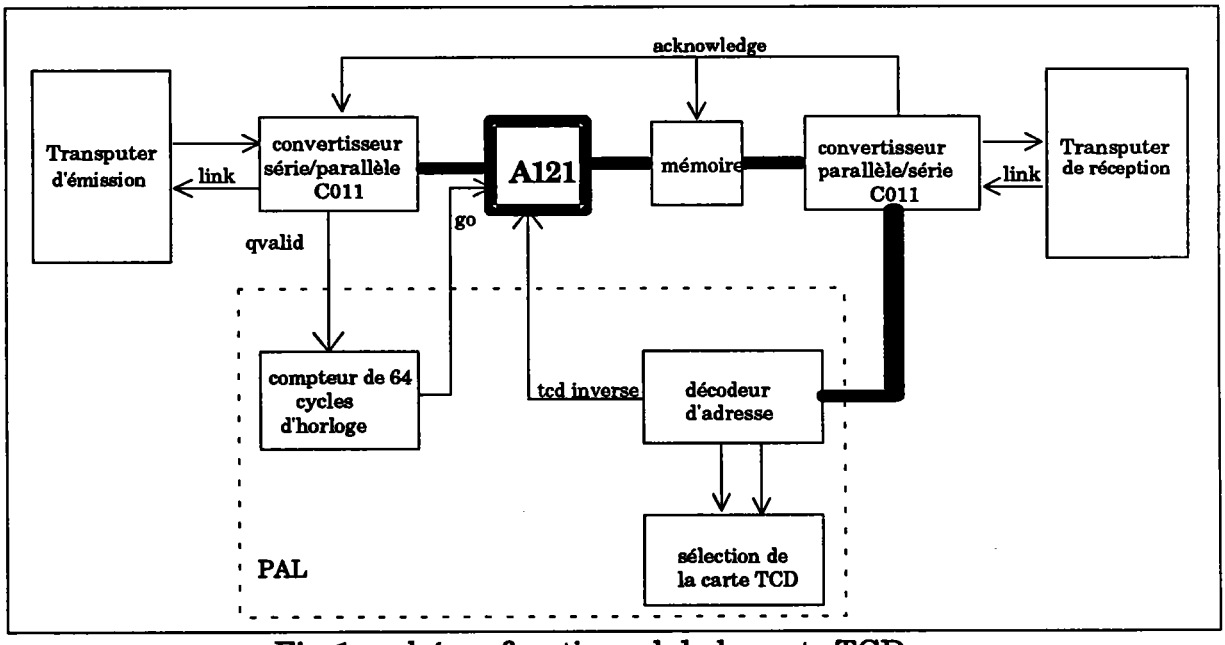


Fig 1: schéma fonctionnel de la carte TCD

Cette carte comprend:

- un DSP A121 qui fonctionne en "pipe line". Il manipule trois blocs à la fois. Le premier est présenté en sortie sous forme de coefficients transformés, le second en phase de traitement et enfin le dernier est récupéré en entrée sous forme de pixels. Ces trois opérations sont exécutées en même temps,

- 2 convertisseurs "série/parallèle" ou "parallèle/série", C011 composant INMOS compatible avec le A121,
- une logique de commande nécessaire au fonctionnement de la carte est stockée dans un PAL (Programmable Array Logic).

Un PAL comporte deux couches ET OU avec possibilité de programmation dans le réseau ET. Les premières technologies utilisaient des transistors bipolaires et des fusibles, par la suite sont apparus des transistors MOS simples ou à grilles flottantes (EPROM).
Les PAL sont utilisés pour réaliser des fonctions combinatoires en ET OU. On rencontre des structures PAL :

- simple ET OU,
- ET OU avec bouclage de certaines sorties sur les entrées,
- même chose avec des bascules en sortie.

Le principe du PAL est de griller ces fusibles internes afin d'obtenir des équations logiques.

2.2 Principe de fonctionnement

Le transputer d'émission (maître) dialogue avec les convertisseurs série/parallèle C011 par des liens. En sortie du C011 on dispose d'un bus 8 bits. Chaque bloc est constitué de 64 éléments. A chaque émission de données, un signal QVALID indique qu'une donnée est présente sur le bus.
Le signal va permettre d'incrémenter un compteur situé dans le PAL. Au bout des 64 impulsions un signal GO doit être émis pour effectuer une nouvelle TCD sur le bloc suivant.
Le QVALID indique aussi au second C011 qui fait la conversion inverse (parallèle/série) qu'une donnée a été émise. Celui-ci confirme par IACK qu'il a bien reçu l'information, ce qui autorise le traitement de la donnée suivante. IACK est utilisé aussi pour mémoriser le résultat en sortie de l'A121 dans un registre, ce qui évite la prise en compte de données parasites.

Une sélection du mode de fonctionnement (Fig.2) est réalisée par des informations provenant de l'extérieur de la carte (transputer de réception). Pour cela, il suffit d'utiliser le lien d'entrée du second C011 de sortie et grâce à la conversion, valider à travers son bus parallèle une des trois configurations autorisées par le module de décodage intégré dans le PAL.

1 - Si A0...A7 = $F0, la carte TCD est transparente aux transputers (c'est le mode par défaut à la mise sous tension). Les transputers communiquent par le lien les reliant comme ils le font d'habitude. Ce mode est nécessaire à la bonne initialisation des transputers.

2 - Si A0...A7 = $0F, le décodeur ordonne à la carte de faire la TCD en validant les buffers appropriés. La carte TCD effectue la TCD sur les blocs pixels qui lui sont envoyés. Dans ce cas, le transputer 1 envoie des blocs de pixels au transputer 2. Ces blocs sont interceptés par la carte qui transmet au transputer 2 les coefficients transformés.

3 - Si A0...A7 = $AA, le décodeur ordonne à la carte de faire la TCD inverse en validant les buffers appropriés (les mêmes que ceux pour faire la TCD). De la même manière la carte TCD effectue la TCD inverse.

Le transputer de réception des blocs transformés peut commuter constamment d'un mode à l'autre.

2.3 Sens des communications

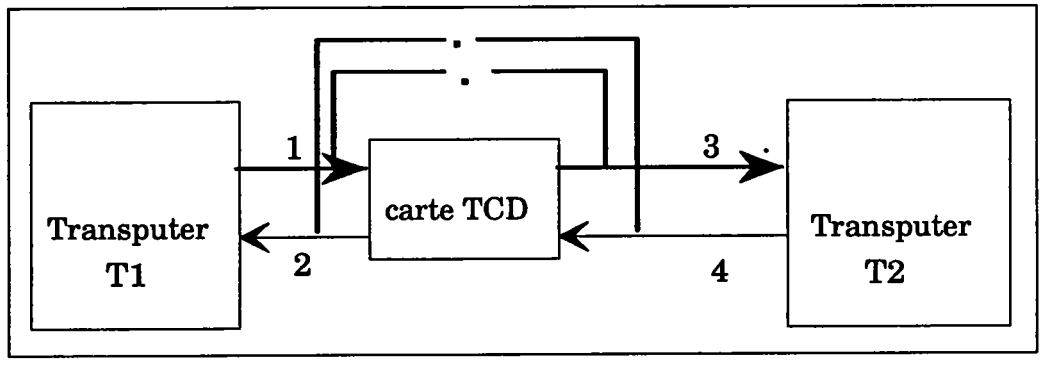


Fig 2: synoptique des communications

Par le fil 1 passent:

- ❶ des données provenant de T1 destinées au transputer T2 (mode transparent pour la carte TCD),
- ❷ des blocs de pixels qui seront transformés par la carte TCD en blocs de coefficients TCD, (mode TCD)
- ❸ des blocs de coefficients TCD qui seront transformés en blocs de pixels par la carte TCD (mode TCD inverse).

Par le fil 3 passent :

- les données ❶ ,
- les coefficients TCD de ❷,
- les pixels de ❸.

Par le fil 2 passent:

- ❹ les données provenant de T2 pour le transputer T1 (mode transparent pour la carte TCD),

Par le fil 4 passent :

- les données ❹,
- ❺ des ordres adressés par T2 à la carte TCD,
- les ordres permettant de modifier l'état de la carte TCD.

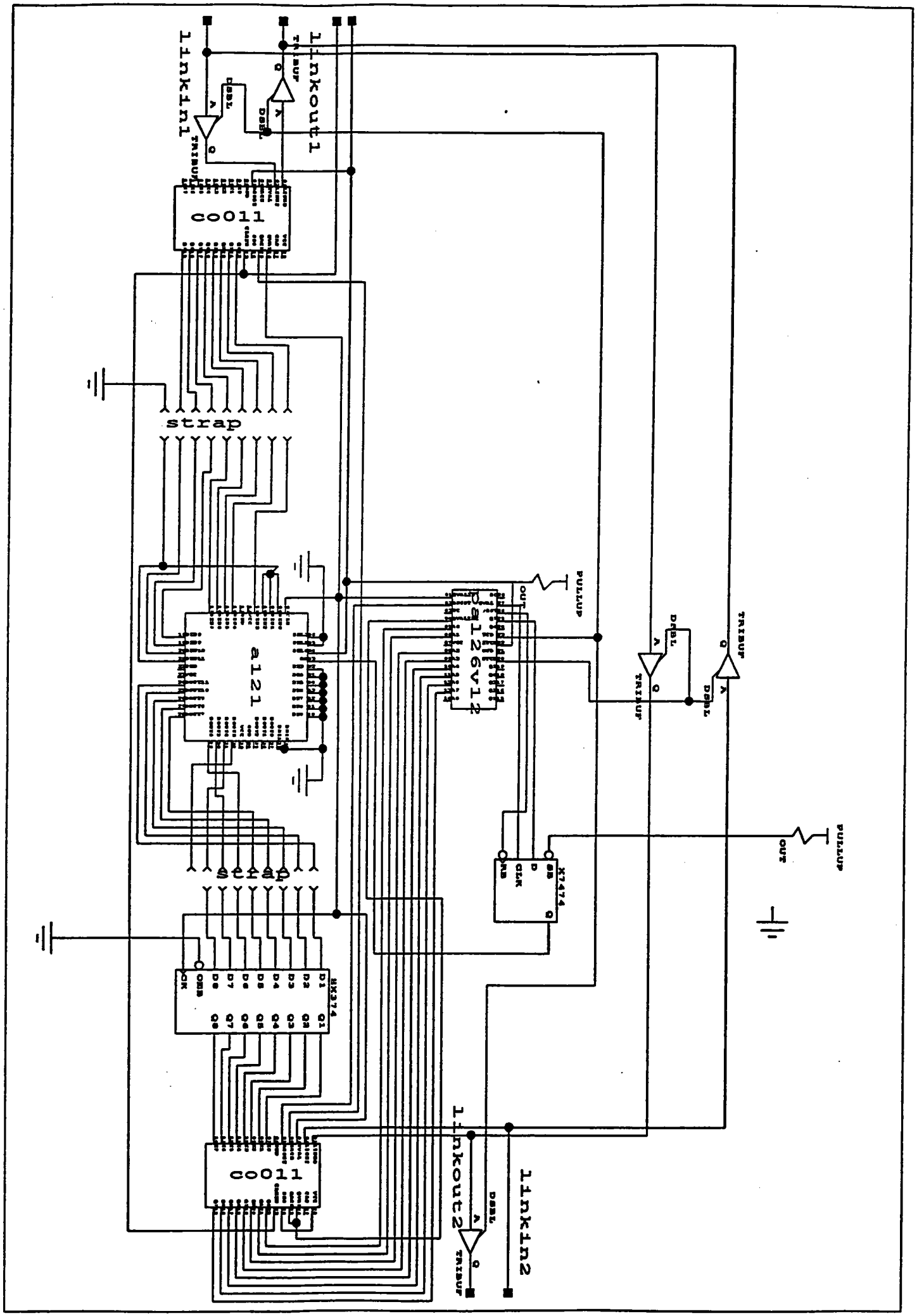


Fig 3: Schéma de la carte TCD pour Transputers

2.4 Les performances comparées

La méthode choisie consiste à intercaler la carte entre deux Transputers c'est à dire sur un lien. Pour cela, on dispose d'un PC équipé d'une carte B008 composée de 10 slots, chaque slot est destiné à recevoir un TRAM (Transputer Module). La carte prototype est placée sur un slot entre le transputer T1 et le transputer T2. La carte peut fonctionner avec des liens à 10 Mbits/sec ou 20 Mbits/sec.

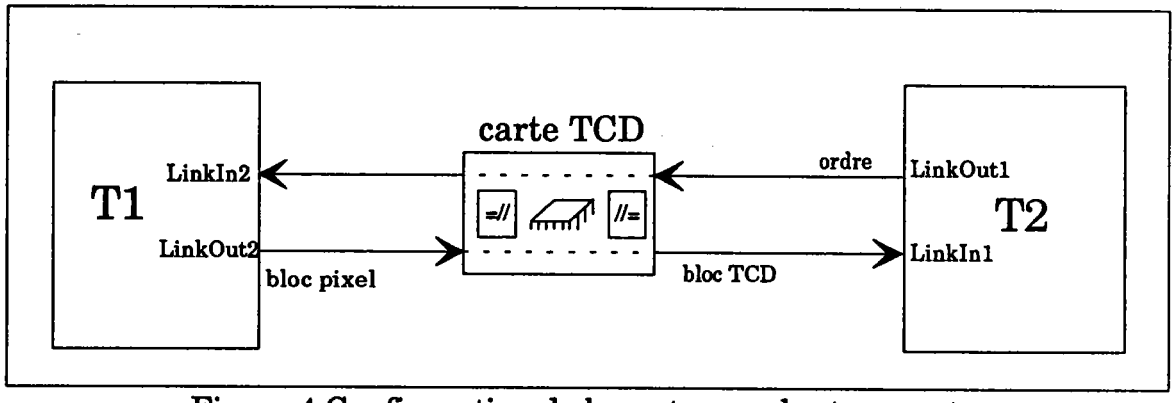


Figure 4:Configuration de la carte avec les transputers

On calcule la TCD d'une image noir et blanc (femme.img) de taille 256 x 256 pixels.
Le transputer T2 envoie la commande par son LinkOut1, le code traverse la carte qui est transparente et est reçu par le transputer1 sur LinkIn2. Dans le même temps la carte TCD change de mode. Ensuite le transputer1 envoie les blocs pixels 8*8 par son LinkOut2, les blocs traversent la carte car elle est sur le lien et le transputer T2 les reçoit en blocs de coefficients transformés par son LinkIn1.

Les résultats obtenus ont été comparés avec ceux pour un traitement séquentiel sur un seul processeur.

Débit du lien	10 Mbits/sec	20 Mbits/sec
Tps carte TCD avec communications.	0, 25 sec	0, 15 sec
Tps des communications	0,162 sec	0,0929 sec
Tps carte TCD hors communication	0,088 sec	0,056 sec
Tps séquentiel de la TCD sur 1 T800	14,090 sec	14,090 sec
Temps gagné avec la carte	14,002 sec	14,034 sec

Tableau 1 : Performances sur la carte TCD

Utilisation de la carte:

Nous utiliserons une structure en anneau sur deux transputers d'abord sans la carte TCD puis avec pour mesurer le gain de performances qu'elle apporte. Dans la suite, on considérera les valeurs suivantes:

- Nd le nombre de vecteurs du dictionnaire,
- Tseq l'exécution séquentielle sur un seul transputer en seconde,
- Tpar le temps de traitement parallèle sur deux transpusters en seconde,
- R(TCD,Tseq) le pourcentage du temps de traitement séquentiel de la TCD par rapport au temps de traitement total Tseq,
- Acc l'accélération.

Le découpage de l'image en blocs se fait au niveau du transputer T1 (maître) qui envoie les blocs un à un après avoir effectué la TCD sur le bloc. Le transputer T2 se charge de l'extraction des vecteurs puis de leur quantification vectorielle, ceci dans le cas où la carte TCD n'est pas utilisée. Mais si on utilise la carte, T1 envoie des blocs pixels à T2 via la carte qui les transforme en bloc TCD.

Nd	16	64	256
Tseq	20,87	39,56	112,19
Tpar	11,26	21,21	79,61
Acc	1,85	1,86	1,40

Tableau 2 : Performances sans la carte

Nd	16	64	256
Tseq (sec)	20,87	39,56	112,19
R (TCD,Tseq)	67,7%	35,6%	12,5%
Tpar(tcd)	5,57	20,60	78,57
Acc	3,74	1,92	1,42

Tableau 3 : Performances avec la carte

Nd	16	64	256
Accélération sans la carte	1,85	1,89	1,40
Accélération avec la carte	3,74	1,92	1,42

Tableau 4: Mesures de l'Accélération

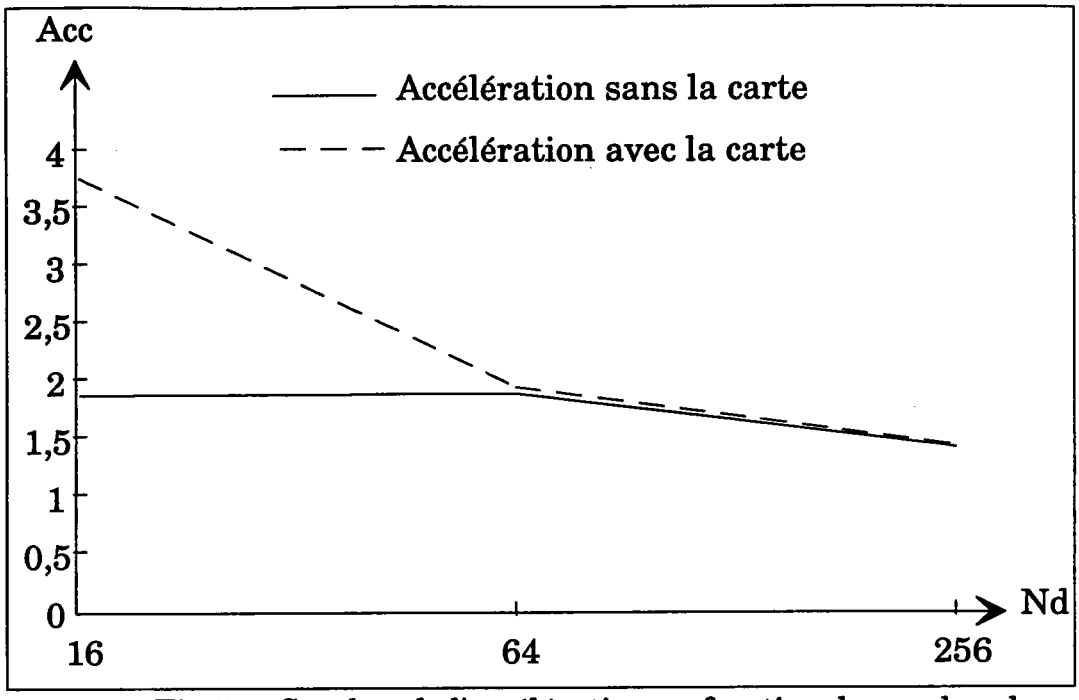


Fig 5 : Courbes de l'accélération en fonction du nombre de transputers.

L'adjonction de la carte TCD apporte des résultats performants quand le temps de traitement séquentiel est le plus important dans le temps total de traitement, comme c'est le cas avec un dictionnaire de 16 vecteurs. Mais si le dictionnaire devient important, d'où un temps de quantification élevé, l'utilisation de la carte ne donne pas les résultats escomptés car une bonne parallélisation sur les données donne sensiblement les mêmes résultats.

2.5 Conclusion

Cette carte conçue dans un environnement parallèle, donne des résultats significatifs. Elle peut être utilisée avec des TMS320C40 ou des I860 ou tout autre processeur utilisant des liens. Elle peut être implémentée sur des cartes de circuit imprimé ou être enfichée dans des emplacements libres de la carte mère d'un micro-ordinateur. Ce qui lui confère une grande facilité de mise en oeuvre et une grande souplesse d'utilisation.

On peut encore améliorer le parallélisme de fonctionnement ce qui permet d'augmenter la vitesse de traitement, en prévoyant plusieurs cartes montées en parallèle.

La "suppression" du calcul de la TCD permet d'avoir des possibilités de fonctionnement, en temps réel, en effet le calcul TCD est "supprimé" grâce à l'emplacement de la carte sur le lien(dans le cas du transputer, le débit du lien est 10 à 20 Mbits/sec). Donc cette carte est un atout pour accélérer le traitement dans tout dispositif faisant appel à la TCD (codage, autofocus, JPEG, MPEG, ...).

Problèmes rencontrés:
- le composant A121 contient 128 pixels au départ donc il faudra tenir compte de ces 128 coups d'horloge en sortie,
- avec la carte, les coefficients sont calculés sur 12 bits, on n'en retient que 8 par troncature et non par arrondi ce qui pose des problèmes de précision,
- l'ordre d'arrivée des coefficients est transposé par rapport à l'ordre de départ, donc les programmes utilisant la TCD devront être modifiés.

3 Le quantificateur vectoriel

Malgré les résultats significatifs apportés par la mise en oeuvre de la carte prototype TCD dans un environnement parallèle, nos objectifs ne sont toujours pas atteints car on est limité par la lenteur de l'opération de quantification vectorielle. C'est dans ce cadre que nous avions développé au Laboratoire d'Electronique Numérique et des Réseaux de Neurones de TDF-C2R un quantificateur vectoriel bit/série. La technologie utilisée est celle des FPGA (Field Programmable Gate Array ou prédiffusé programmable sur site) de XILINX pour la mise au point. Une évolution ultime de l'interface de quantification serait un ASIC (Application Specific Integrated Circuit).
Ce quantificateur vectoriel aura pour vocation initiale la compression d'images mais, il pourra être utilisé pour d'autres applications.

3.1 La quantification vectorielle (Q.V.)

On rappelle que le principe de la quantification vectorielle repose sur le fait qu'un vecteur V = (vi ; i = 1, ..., k) peut être remplacé par un vecteur U = (ui; i=1, ..., k) de même dimension qui lui est son plus proche voisin. Ce vecteur U (codeword) appartient à un ensemble fini et ordonné de N vecteurs appelé Dictionnaire (codebook) (Fig. 6).

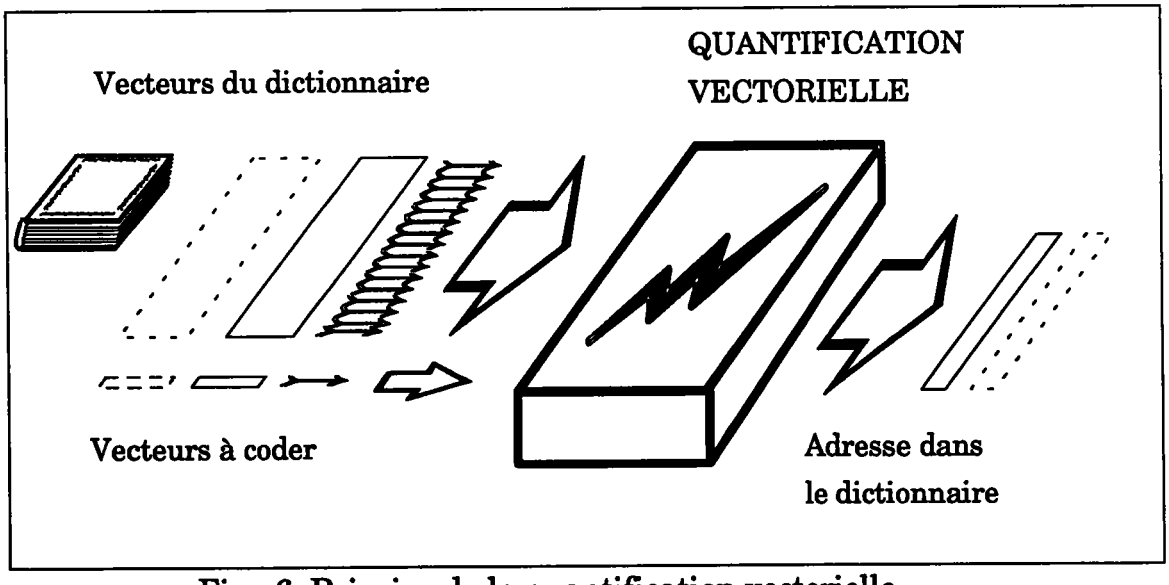


Fig . 6: Principe de la quantification vectorielle.

3.2 Les quantificateurs présents à ce jour sur le marché

3.2.1 Togaï

Ce circuit est destiné au tri et à la classification. Il possède trois algorithmes de classification:

- 1) la distance de Malahanobis $\Sigma(Xk*Wk)$ pour k=0 à kmax,
- 2) la distance moyenne absolue $\Sigma|Xk-Wk|$ pour k=1 à kmax,
- 3) la distance moyenne quadratique $\Sigma(Xk-Wk)^2$ pour k=1 à kmax.

Aucune information technique n'étant disponible à ce jour, il nous est impossible d'évaluer ce circuit.

3.2.2 Le Q.V. série en technologie VLSI

Le quantificateur [BRE-86] a été intégré dans un VLSI suivant un concept dit: Structured Tiling (design constitué de cellules qui sont disposées régulièrement sur une grille). Nous allons l'étudir en détail.

3.2.2.1 Description

La distance utilisée est la MRRVQ (Mean Residual Reflected Vector Quantizer = Différence des Moindres Carrés d'un Vecteur Réfléchi) qui exploite des vecteurs de dimension 16 et un dictionnaire constitué en arbre de 16*16 vecteurs.
L'algorithme est donc constitué:

- du calcul de la moyenne des composants de chaque vecteur ,
- de la soustraction de cette moyenne de chaque élément de chaque vecteur,
- d'une rotation suivant l'axe des Y afin d'obtenir l'information significative en haut,
- d'une rotation suivant l'axe des Y afin d'obtenir l'information significative à gauche.

Cette série d'opérations classe les vecteurs, afin d'avoir les pixels de plus forte valeur en haut à gauche. A la suite du calcul de la moyenne des luminances, le dispositif effectue un repérage du cadran de plus forte intensité. Cette indication détermine les axes des rotations à appliquer au bloc. L'information est directement exploitable en sortie de l'additionneur (Fig 7). La structure arborescente de ce dernier donne comme résultat intermédiaire la somme partielle des 4 cadrans.

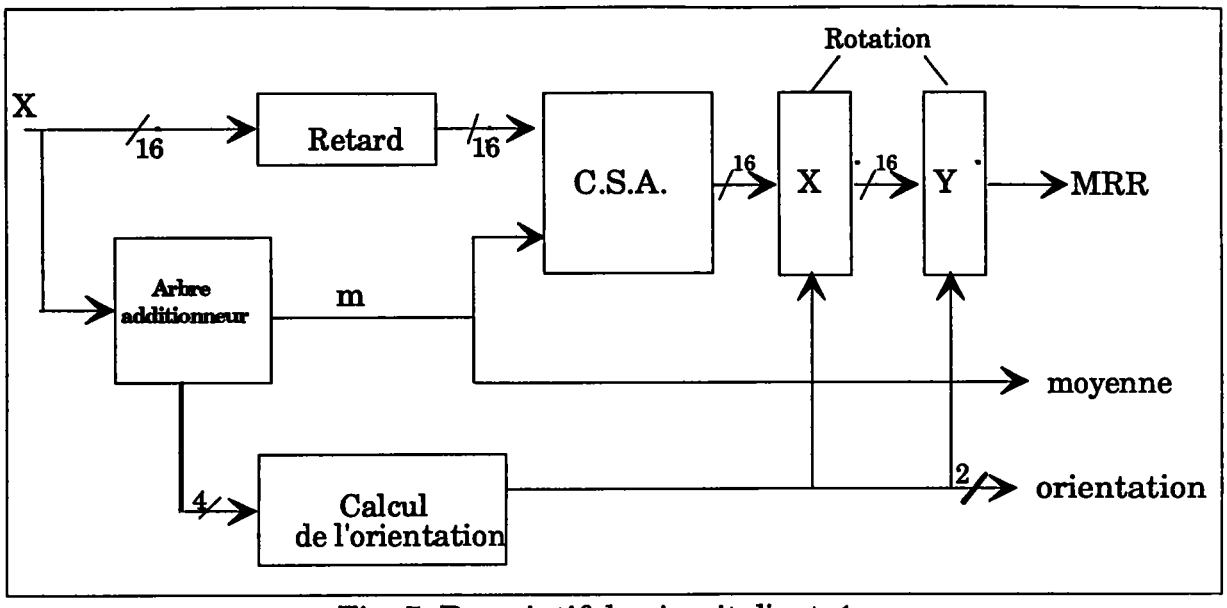


Fig. 7: Descriptif du circuit d'entrée

La disposition des vecteurs, ainsi classés, optimise la recherche du vecteur le plus proche du dictionnaire. En effet, l'orientation imposée des vecteurs facilite la recherche et minimise les traitements et par ce fait le temps d'exécution.

A chaque vecteur sont affectées les données suivantes : la moyenne, l'orientation (imposée par les rotations), l'adresse du vecteur le plus proche dans le dictionnaire.

Pour réaliser l'ensemble de ces calculs, le système est constitué de 3 éléments distincts (Fig. 8):

- un circuit d'entrée (front end chip) calculant la moyenne et l'orientation,
- deux Q.V. qui comparent le vecteur d'entrée avec le niveau de chaque dictionnaire,
- deux mémoires de type mémoire morte contenant le dictionnaire.

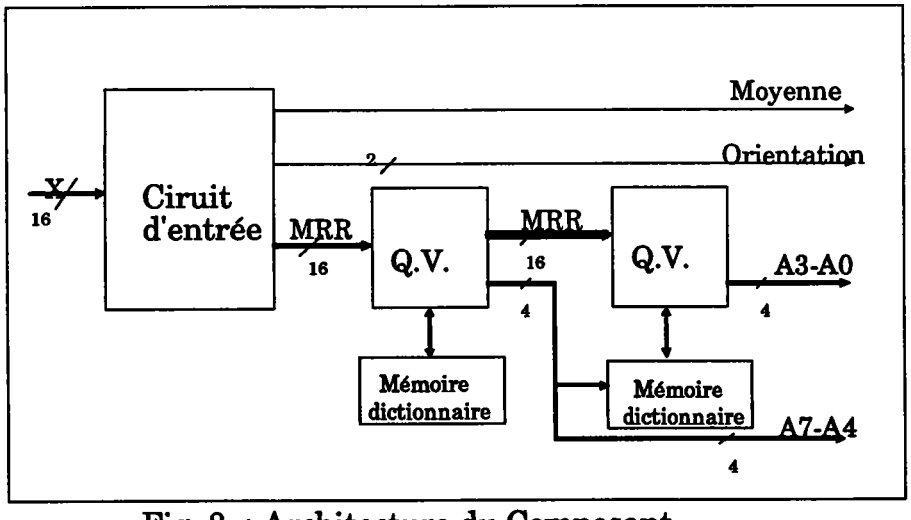


Fig. 8 : Architecture du Composant

Les données sont entrées en série avec le LSB en premier.

Le Q.V. est constitué d'un registre d'entrée, d'un multiplexeur, d'un soustracteur permettant de calculer bit à bit la différence entre le vecteur d'entrée et les vecteurs contenus dans le dictionnaire. L'étage suivant calcule le carré de la différence et fait la somme de tous les résultats précédents.
Le dernier étage est constitué d'un comparateur qui évalue chaque résultat et donne la plus petite distance obtenue ainsi que son adresse. Bien sûr, le quantificateur gère aussi les mémoires contenant les dictionnaires.

Malheureusement ce type de structure est beaucoup trop lent pour faire un traitement temps réel de l'image.

3.3.3 LE Q.V. série en VLSI destiné au codage d'images temps réel

L'intégration d'un quantificateur vectoriel [RBT-90] n'a été réalisée que partiellement afin de valider le concept d'une architecture dite "systolique". Une architecture systolique est constituée de macro éléments qui sont répétés X fois et qui sont reliés les uns aux autres.
A chaque coup d'horloge les données et les résultats sont propagés aux macro éléments adjacents suivant l'axe des X et des Y donc suivant une structure à deux dimensions (Fig.9).

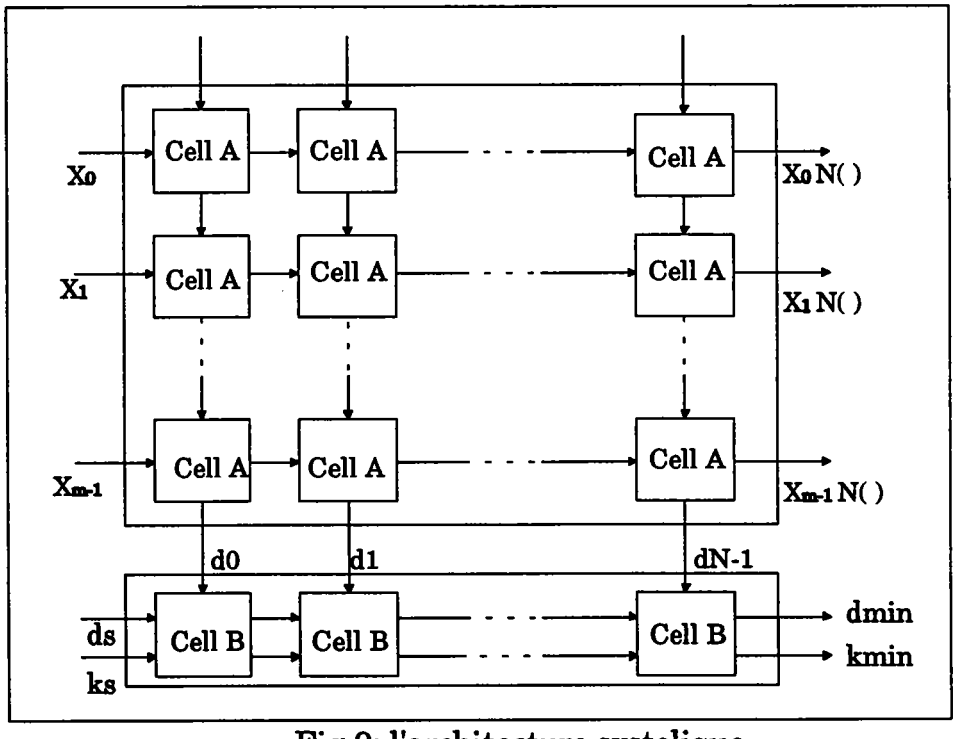


Fig 9: l'architecture systolique

L'algorithme utilise la distance euclidienne .

Chaque élément de traitement est constitué (Fig.10) :
- d'une cellule mémoire vive contenant un bit d'un vecteur du dictionnaire,
- d'une logique arithmétique calculant la différence puis le carré entre

l'élément mémorisé et un bit du vecteur d'entrée,
- d'un additionneur qui somme les résultats de la cellule N avec celui de la cellule N-1
- de deux registres qui stockent les données pour ensuite les transmettre aux éléments adjacents.

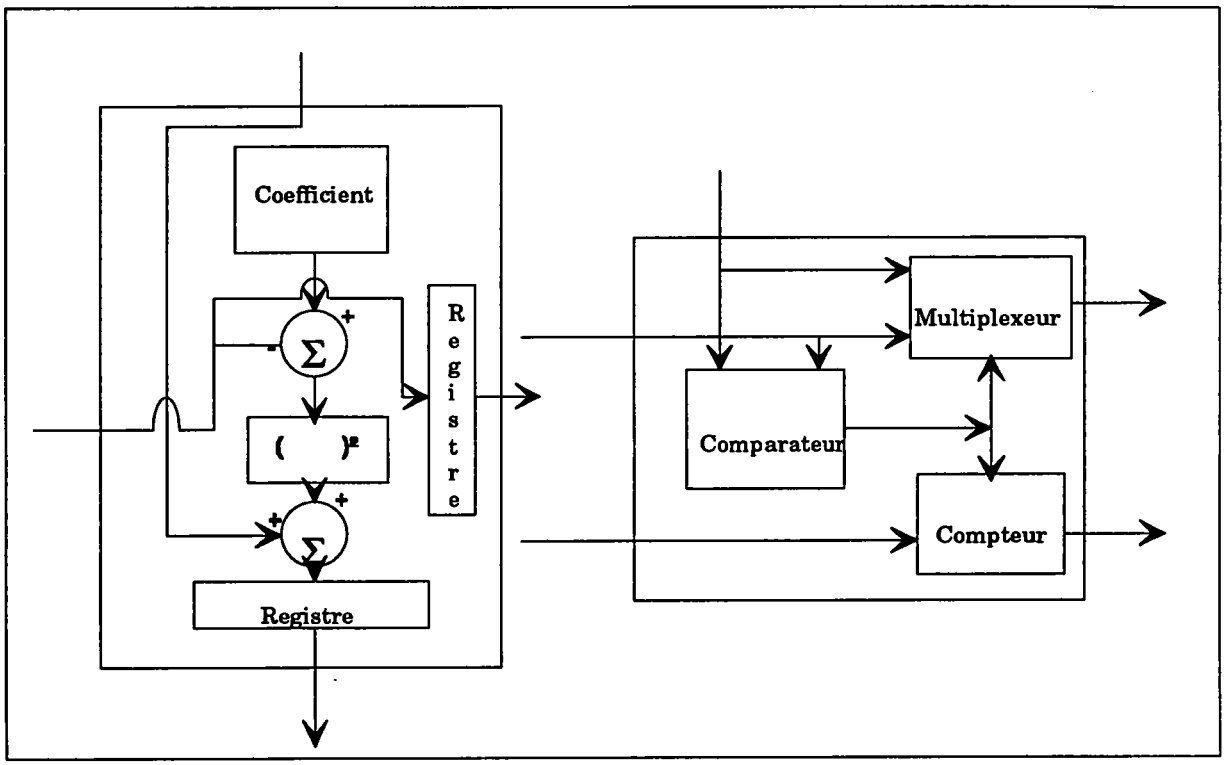


Fig 10: Cell. A Elément de traitement Fig 11: Cell. B Comparaison.

Le résultat des comparaisons successives ainsi que l'adresse du vecteur le plus proche sont obtenus grâce à des cellules de comparaisons (Fig.11). Chaque cellule de comparaison est constituée d'un registre, d'un multiplexeur et d'un compteur. L'architecture systolique ne donnant pas un résultat sous une forme parallèle, chaque résultat de comparaison est transmis d'une cellule à l'autre; donc, il faut parcourir l'ensemble des cellules afin de déterminer la distance la plus petite.

La taille du Q.V. ainsi développé possède autant d'éléments suivant l'axe des Y qu'il y a d'échantillons dans le vecteur d'entrée. Sa profondeur, suivant l'axe des X, est égale à la taille du dictionnaire.
Le nombre de comparateurs est bien évidemment égal au nombre de vecteurs contenus dans le dictionnaire.

Cette architecture gourmande en surface de silicium et en nombre d'entrées/sorties offre plusieurs avantages :
- architecture et cellules répétitives,
- une quantification en continu : à chaque coup d'horloge un nouveau vecteur d'entrée peut être présenté,
- une logique de commande simple.
Le nombre de coups d'horloge nécessaire dépend directement de la dimension

du vecteur d'entrée et de la taille du dictionnaire.

Mais, cette architecture est difficilement exploitable aujourd'hui en raison de l'important niveau d'intégration et du nombre de connexions externes qui lui sont nécessaires. Le second handicap majeur provient de l'interconnexion des macro cellules faisant croître le nombre de coups d'horloge proportionnellement au nombre de vecteurs traités.

3.3.4 Micro Devices [MID-90]

3.3.4.1 MD 1210 Comparateur à logique floue

Ce composant en technologie CMOS permet de calculer la distance de HAMMING en mode bit à bit, ou de calculer la distance absolue dès que le champ est supérieur à un bit. Il a une fréquence de fonctionnement de 20MHz. Chaque composant effectue une comparaison entre un vecteur connu et 8 autres inconnus ou l'inverse. La longueur maximum des vecteurs à comparer est limitée par la profondeur de l'accumulateur qui est un registre de 16 bits.

Le calcul de la distance se fait sur un octet. Le vecteur inconnu est entré sous forme série. Le calcul de la distance s'opère en mode bit à bit avec le LSB en premier. Les 8 vecteurs du dictionnaire sont contenus dans des mémoires vives qui sont gérées par la logique de commande du composant.

Ce composant offre plusieurs possibilités à travers ses registres de commande :

- invalider le résultat d'une opération sur un ou plusieurs vecteurs,
- choix de la longueur du champ de calcul (de 1à 8),
- identification du composant (dès qu'il y a association de boîtiers) ,
- établissement d'une valeur de seuil afin de sélectionner les vecteurs pertinents,
- valider les interruptions,
- gestion des mémoires.

Lorsque qu'on met en cascade plusieurs boîtiers (32 max), soit un dictionnaire de 256 vecteurs, chaque composant doit être identifié et chaque registre de seuil doit contenir la même valeur. Au terme de la comparaison, le vecteur le plus représentatif de chaque quantification est alors comparé aux autres au travers d'un bus spécifique. On retrouve ses coordonnées dans chaque registre résultat de tous les boîtiers associés.

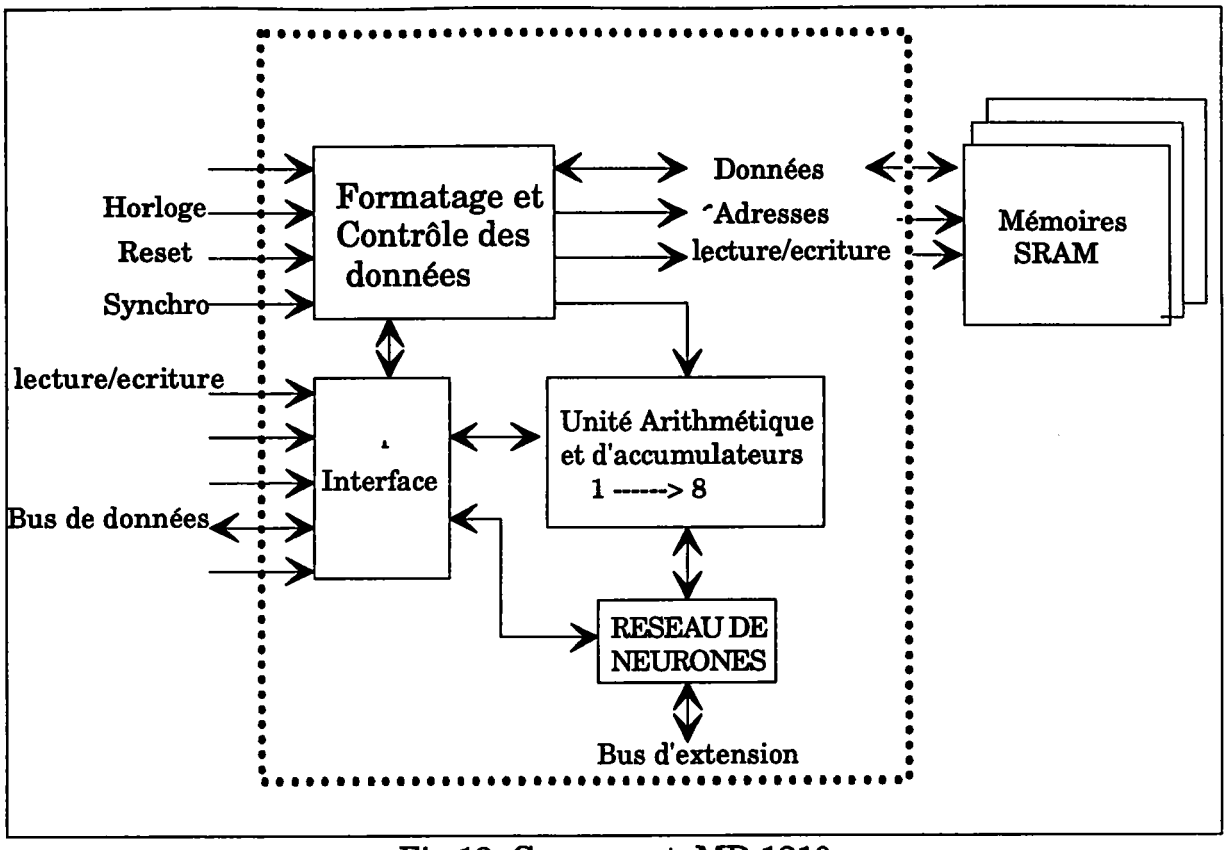


Fig 12: Composant MD 1210

3.3.4.2 MD 1212 Corrélateur de données floues

Ce composant en technologie CMOS fonctionne à 50 Mhz. Il calcule la distance de HAMMING entre un vecteur d'entrée de longueur maximale de 128 bits et une trame de même profondeur mémorisée dans le composant au départ de chaque corrélation.
On peut étendre ce calcul à 32 vecteurs en mettant en cascade autant de boîtiers.
Les éléments de mémorisation de la trame dite de référence sont accessibles via un bus 8 bits, tandis que le vecteur d'entrée est présenté sous une forme série. Lorsque l'on associe plusieurs boîtiers les données à comparer transitent dans chaque boîtier du LSB au MSB. (Fig.13)
Ce composant est doté aussi d'un registre de masque permettant de ne valider que les bits pertinents et d'un registre de seuil en sortie du comparateur.
Un bit est positionné à la fin de chaque corrélation et le résultat est disponible dans un registre via un bus 8 bits de sortie.
L'association de plusieurs boîtiers est possible mais elle nécessite l'adjonction d'un additionneur en sortie pour donner le résultat global du calcul de la distance.

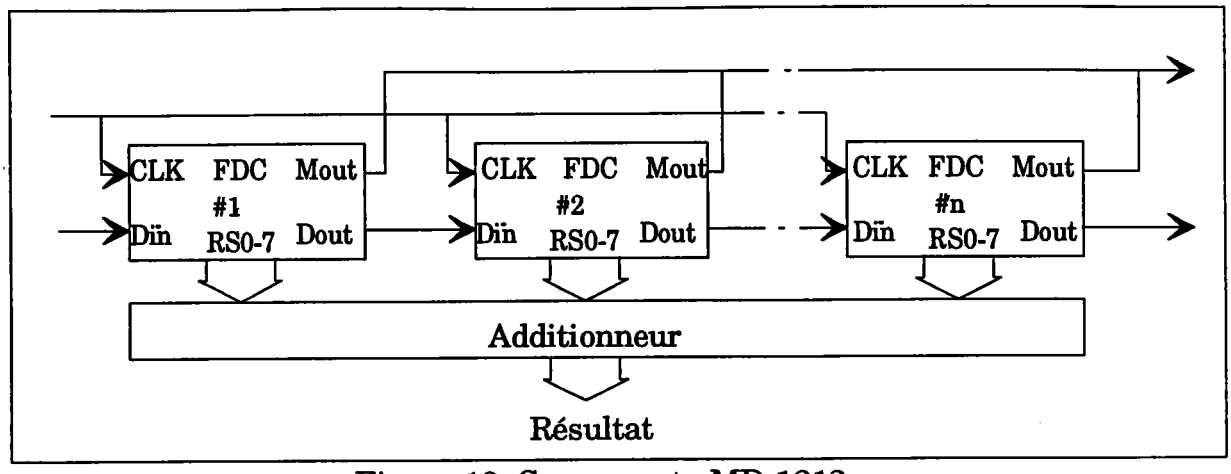


Figure 13. Composant MD 1212

Ce composant est destiné à la comparaison de tous types de signaux :
- Radar / Sonar
- Synchronisation
- Reconnaissance / Comparaison d'images

Son utilisation dans le codage d'images n'est pas possible car la distance de HAMMING calculée sur des bits n'est pas adaptée pour la compression.

3.3.5 Le Réseau de Neurones

Le réseau de neurones constituant le module de comparaison utilise des fonctions de logique élémentaire. Il est composé de 142 neurones et identifie la valeur la plus petite parmi les résultats du calcul d'erreur provenant de 8 accumulateurs différents.

Dans le module de comparaison, le bit de poids le plus fort, qui n'est pas à zéro, inhibe les neurones associés au mot traitant les bits de poids plus faible. Cela est facilement explicable à travers la figure 14. Le bit de poids le plus fort est présenté en premier et simultanément sur les portes N^2_1 et N^3_{11}. La porte ET (N^2_1) détecte si l'un des MSB de chaque accumulateur est à 0. Il transmet le résultat aux portes suivantes, OU Exclusif (N^3_{i1}), qui comparent ce résultat avec chaque MSB. Dans le cas où les deux bits ont la même valeur, il autorise la comparaison sur les bits N-1 à N-7. Dans le cas contraire ces bits sont ignorés. Une simple porte OU (N^1_{i1}) effectue ce test. Les portes ET (N^2_i) donnent le résultat de chaque comparaison sur un bus. (Dans la figure 14 on peut suivre pas à pas l'étape décrite ci-dessus avec les valeurs données à chaque bit.)

Lorsque l'on associe plusieurs composants, un réseau de plus petite taille est utilisé. Les distances sont comparées deux à deux. Ce second réseau se trouve à l'intérieur du composant en parallèle sur le bus de sortie "résultat de comparaison". On retrouve l'adresse du vecteur le plus proche dans le Win register (Reg.7).

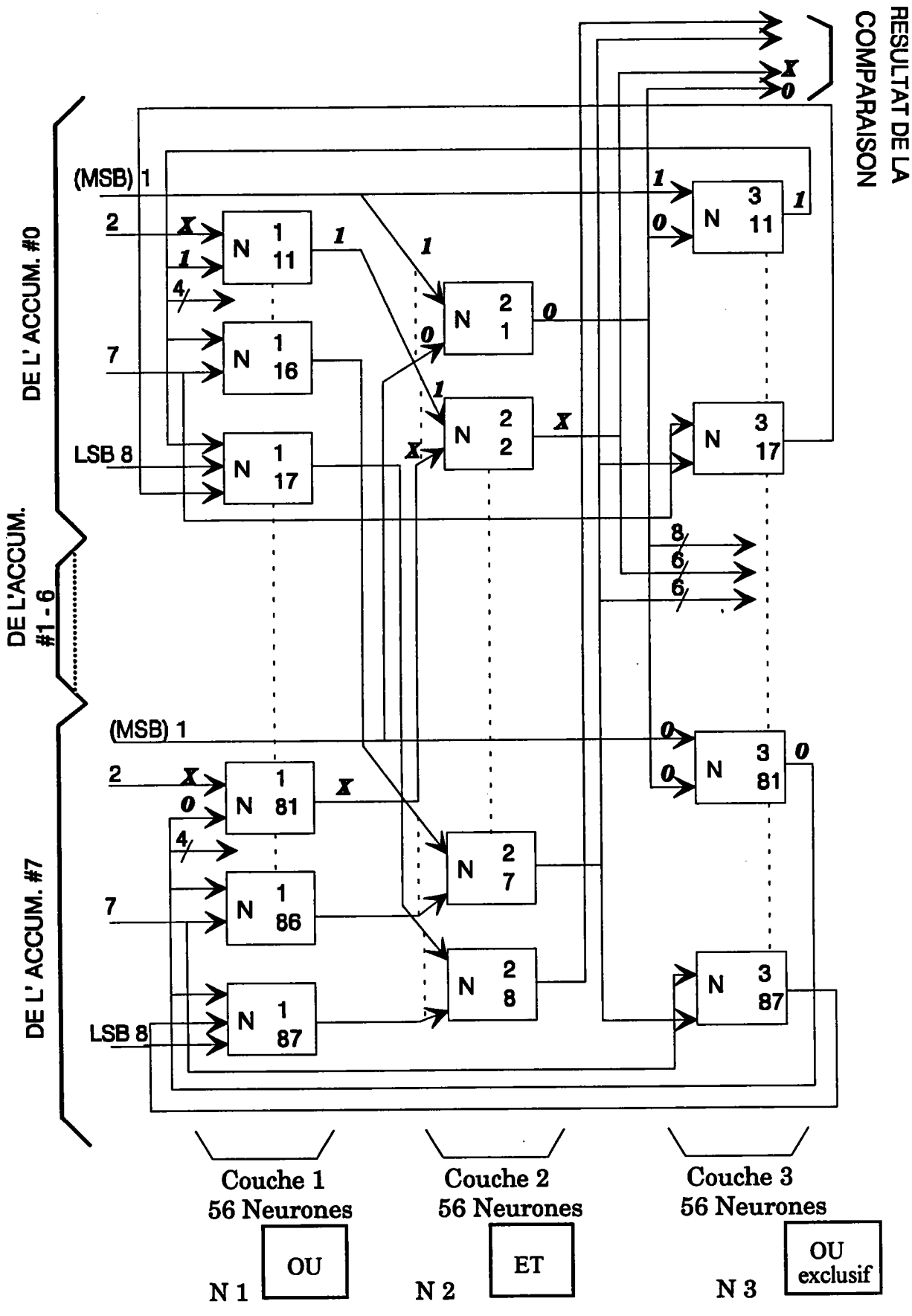


Fig. 14: Structure du Réseau de Neurones

La commercialisation de ce composant démontre l'importance grandissante de la Q.V. dans le codage de données. Son créneau d'utilisation ne s'étend pas au codage d'images même en augmentant sa fréquence de fonctionnement. Il n'offre pas la possibilité de faire des comparaisons en vague. Le temps de traitement est directement proportionnel au nombre et à la dimension des vecteurs.
Son architecture n'est pas adaptée à notre étude.

3.3 Le quantificateur vectoriel développé

Le composant a été développé suivant une architecture bit série. Elle est composée d'un ensemble de macro éléments qui, associés les uns aux autres, permettent de calculer la différence entre deux vecteurs.
Le premier est dit d'entrée et le second est contenu dans le dictionnaire. Cette association est répétée autant de fois qu'il y a de vecteurs présents dans le dictionnaire.
La distance utilisée pour comparer deux vecteurs entre eux est la distance en valeur absolue et ceci en raison de sa facilité de mise en oeuvre :

$$d_1(X,Y) = \sum_{i=1}^{k} |x_i - y_i|$$

3.3.1 Composition globale

Le schéma est généré à partir de l'algorithme de quantification vectorielle. On dispose d'un dictionnaire contenant N vecteurs U de dimension k, d'un vecteur d'entrée V de même dimension à quantifier, et d'une norme à calculer.

```
distance = 0;
indice vecteur_plus_proche = 0;
distance_minimum = k*255;

Pour i = 1 à N;
        Pour j = 1 à k
        • distance = distance + Valeur Absolue ( U[i][j] - V[j] );
        Prochain j;
        • Si distance < distance_minimum  Alors
        distance_minimum = distance;
            • indice_vecteur_plus_proche = i;
Prochain i;
```

L'algorithme de quantification vectorielle consiste donc à déterminer l'indice de ce plus proche vecteur dans le Dictionnaire. L'indice peut se calculer en utilisant la distance absolue.

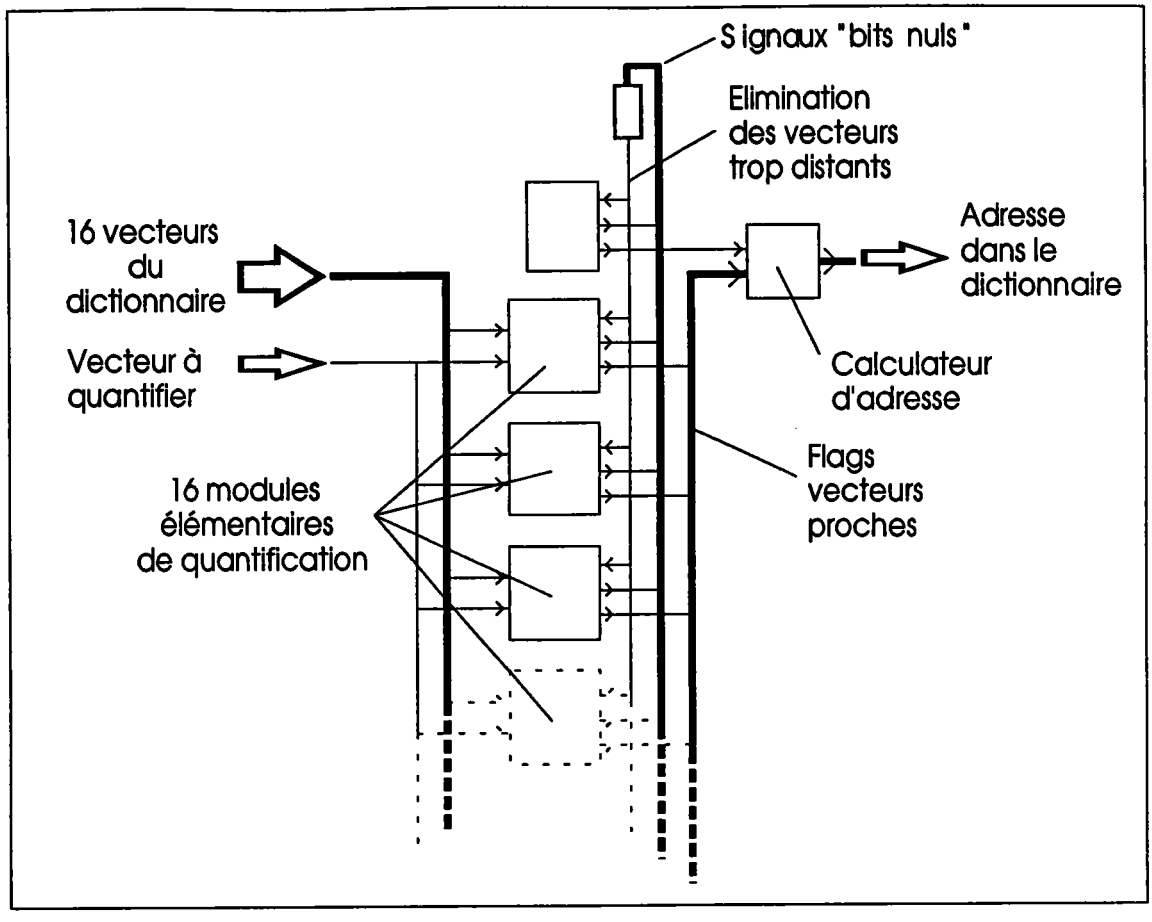


Fig 15 : Schéma global du quantificateur.

Le principe est de comparer ce vecteur d'entrée aux vecteurs du dictionnaire pour donner l'adresse de son plus proche voisin. Le dictionnaire est découpé en vagues de 16 vecteurs. On répartit la comparaison entre 16 modules élémentaires de quantification.
Chaque module élémentaire doit calculer la distance entre le vecteur d'entrée et le vecteur du dictionnaire dont il a la charge. Dès que l'on dispose des 16 distances, une opération de comparaison commence.
Le but est de comparer entre elles des distances binaires sur 12 bits. Il faut déterminer quel est le module élémentaire qui a trouvé la distance la plus petite.

Nous allons montrer dans le tableau suivant un processus de comparaison des distances sur 12 bits, qui va nous permettre de déterminer le vecteur le plus proche:

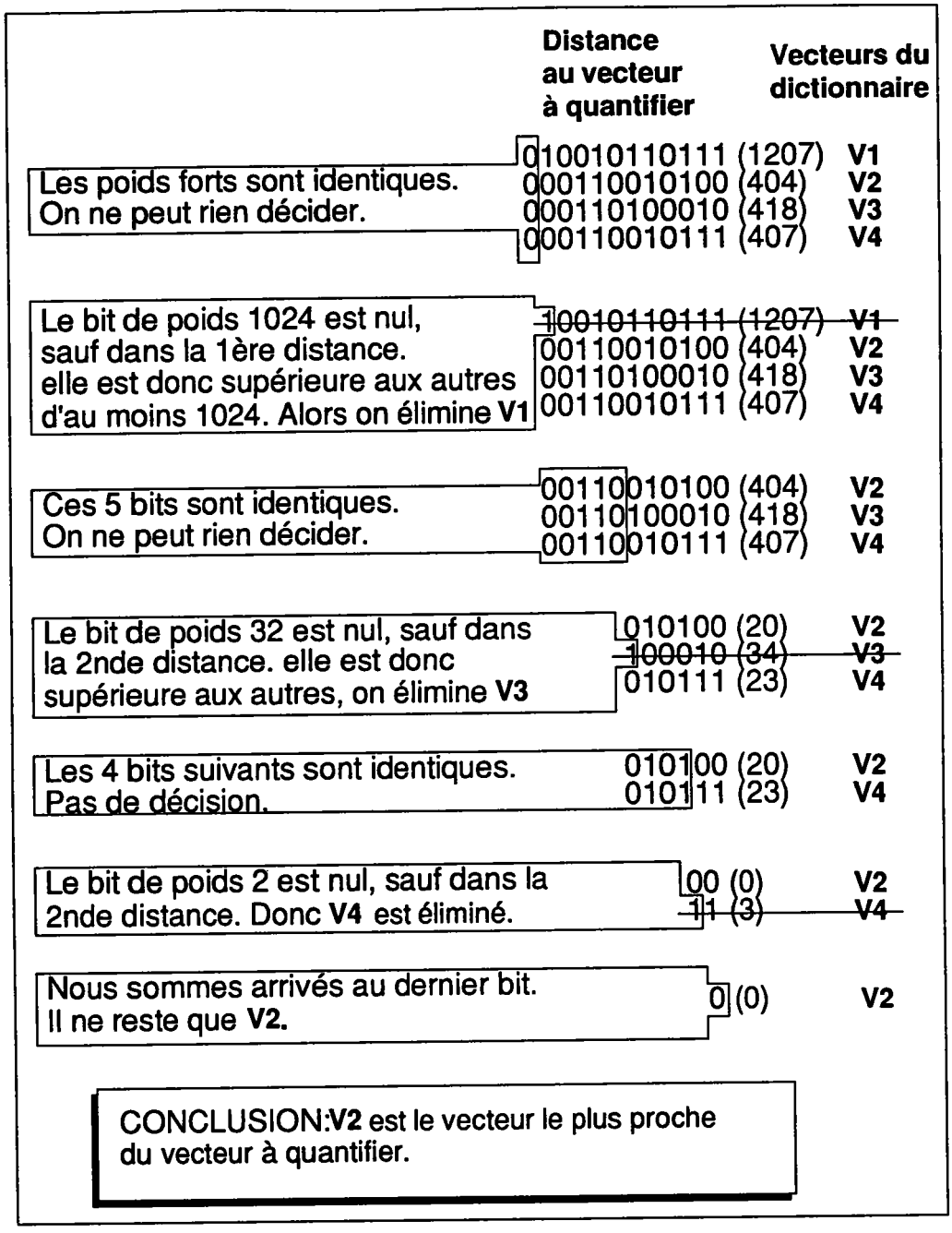


Fig 16: Recherche la distance minimale.

Le quantificateur réalise la fonction suivante: comparaison d'un vecteur X de taille *i* à une ou plusieurs vagues de 16 vecteurs Y d'un dictionnaire.
La dimension d'un vecteur peut aller jusqu'à 16 octets. Le nombre maximal de vagues de vecteurs est de 16.
En sortie, nous disposons de l'indice (ou adresse) du vecteur du dictionnaire le plus proche du vecteur d'entrée.

Remarque: En cas d'égalité de distance entre le vecteur d'entrée et les vecteurs du dictionnaire, il n'y a pas de moyen a priori pour décider qu'un de ces vecteurs est plus pertinent. Aussi en sortie du quantificateur, un seul de ces vecteurs est désigné. Une priorité est donnée au premier vecteur dans une

vague et à la première vague dans le dictionnaire. Il est possible d'ordonner le dictionnaire pour favoriser tels ou tels vecteurs.

Les vecteurs sont entrés sous une forme série, coordonnée par coordonnée, en partant du LSB (bit de poids faible) et en finissant par le bit de signe. Cela permet un calcul direct de la distance. On rentre simultanément le vecteur à coder et une vague de 16 vecteurs du dictionnaire. Pour les vagues suivantes, il faut répéter le vecteur d'entrée pour le comparer à d'autres vecteurs. Après épuisement du dictionnaire le quantificateur désignera par son adresse le vecteur le plus proche.

3.3.2 Module quantificateur élémentaire

Le module quantificateur élémentaire prend en charge la comparaison entre le vecteur en entrée et l'un des 16 vecteurs du dictionnaire. Il effectue successivement la différence entre les coordonnées, la valeur absolue de ces différences et la somme des valeurs absolues.
Le résultat est donc la distance entre le vecteur d'entrée et le vecteur du dictionnaire. Il est stocké dans un registre bidirectionnel. Le processus se termine par la comparaison des distances entre modules.

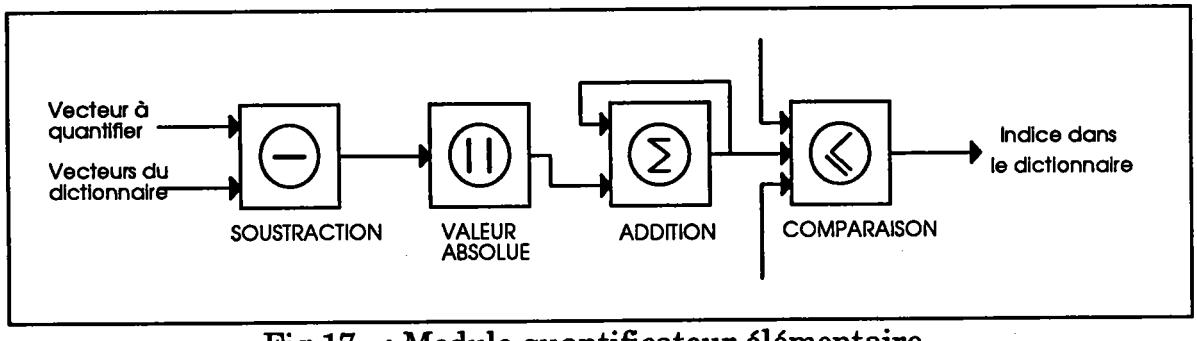


Fig 17 : Module quantificateur élémentaire.

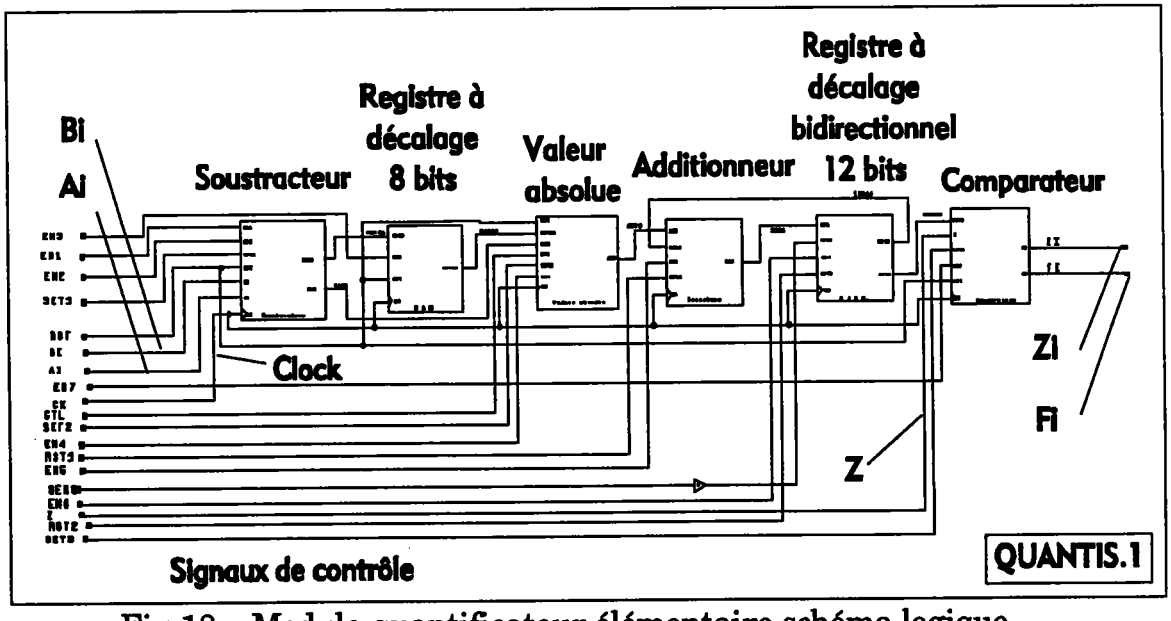


Fig 18 : Module quantificateur élémentaire schéma logique.

Les coordonnées sont codées sur 8 bits signés (-128 à +127), présentées en série à l'entrée du soustracteur (LSB en premier, MSB et bit de signe en dernier).

Le Soustracteur

Le principe du soustracteur est assez simple. La soustraction d'une coordonnée du vecteur à quantifier avec une autre d'un vecteur du dictionnaire nécessite 8 bits en entrée et 9 bits en sortie.

Le registre à décalage de 8 bits

Sa fonction est de mémoriser le résultat de la soustraction pendant le temps nécessaire à l'arrivée de la retenue.
Il permet d'attendre ce bit de signe et de calculer alors la valeur absolue du résultat, sur 8 bits non signés (0 à +255).

Module de valeur absolue

Ce module calcule la valeur absolue de la soustraction des coordonnées. On a 9 bits en entrée et 9 bits en sortie. Si le signe est positif les données ne sont pas modifiées. S'il est négatif, le module donne le complément à 2 du résultat.

L'additionneur

L'additionneur est rebouclé sur lui même par le registre à décalage de 12 bits permettant de stocker la somme des valeurs absolues (LSB est en premier). Les 12 bits (0 à +4095) permettent de travailler avec des vecteurs d'un maximum de 16 coordonnées. Le résultat final est la distance entre vecteurs sur 12 bits (LSB en premier).

Remarque: La somme maximale que l'on peut atteindre dans l'additionneur est (vecteurs de 16 coordonnées)

$16\times[(-128)-(+127)]=111111110000(4080)$

$16\times[(+127)-(-128)]=111111110000(4080)$

Il n'y a donc jamais de retenue à la fin du calcul sur 12 bits.

Registre à décalage bidirectionnel 12 bits

Ce registre a deux fonctions:
- la première est d'accumuler la somme des distances par un rebouclage sur l'additionneur,
- la deuxième est de permettre la sortie du résultat dans l'ordre inverse pour permettre la comparaison qui se fait en commençant par le bit de poids fort alors que le calcul se fait en partant des bits de poids faible.

Le comparateur

Le comparateur effectue la comparaison des distances entre les 16 modules suivant l'algorithme décrit dans la figure 16. Il a pour sortie un signal de détection de bit à zéro et un flag. Ce flag (signal binaire) sert à désigner le module quantificateur ayant trouvé la plus petite distance .
En entrée du comparateur un signal lui indique que d'autres modules ont un bit nul dans leur distance, ce qui déclenchera son élimination éventuelle (abaissement du flag).

3.3.3 Traitement de vagues de vecteurs

Pour traiter des vagues successives de vecteurs, après chaque vague, on mémorise la distance du vecteur le plus proche. Puis on compare les distances de chaque nouvelle vague à cette distance minimale. Le processus d'élimination des plus grandes distances est traité par un comparateur.

3.3.4 Encodage de l'adresse des vecteurs

L'encodage a pour but de réduire l'information apportée par l'ensemble des flags en cours de quantification à une donnée simple et finale: l'adresse dans le dictionnaire du vecteur le plus proche du vecteur à quantifier.
C'est à ce niveau que se décide la priorité accordée à un vecteur ou à un autre s'ils sont à même distance du vecteur à quantifier.
Une priorité est donnée à l'indice le plus faible dans une vague et à la première vague parmi plusieurs. D'où la présence d'un encodeur prioritaire et d'un compteur de vagues.

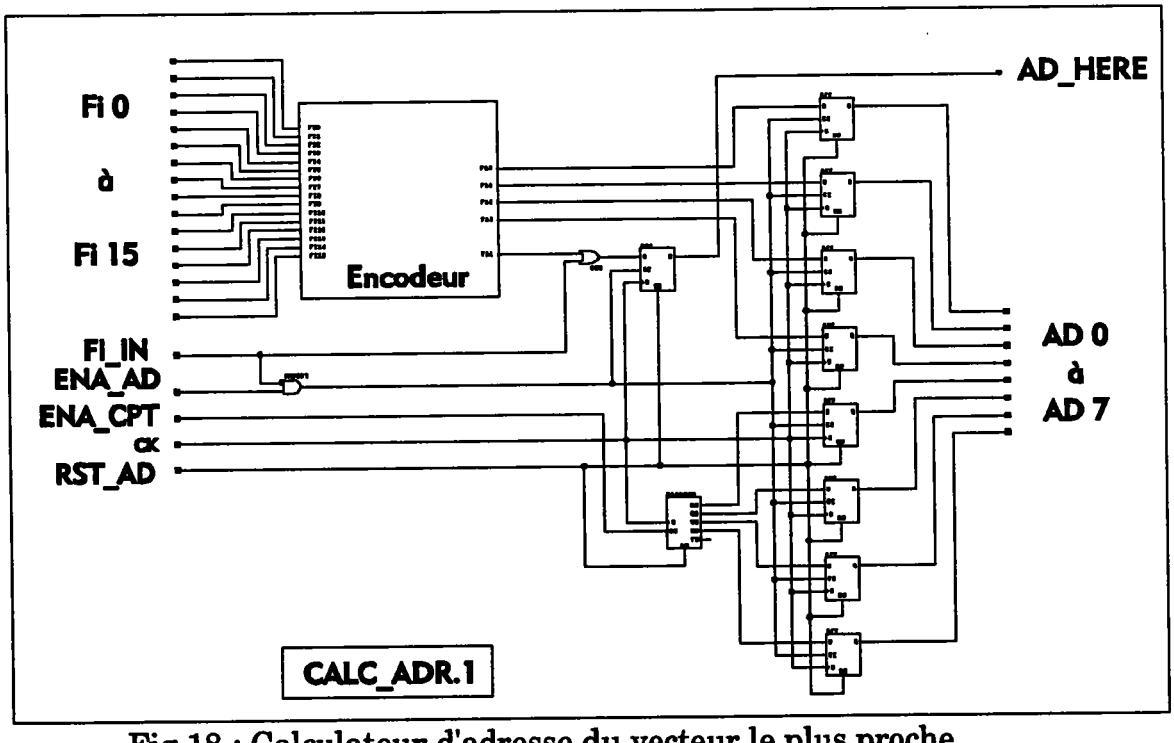


Fig 18 : Calculateur d'adresse du vecteur le plus proche.

Description des signaux:
- RST_AD remet à zéro les latches qui mémorisent l'adresse, ainsi que le compteur de vagues de vecteurs.
- CK est l'horloge.
- ENA_AD provoque l'écriture de l'adresse dans les "latches" (bascules) de sortie. Il doit être mis à 1 au moment où les flags sont valides.
- ENA_CPT sert à incrémenter le compteur de vagues. Il doit être mis à 1 pendant 1 période d'horloge après chaque vague.
- FI0 à FI15 sont les entrées des 16 flags.
- FI_IN est l'entrée du signal binaire ZINVD assigné au registre de la plus petite distance.
- AD0 à AD7 sont les 8 bits de l'adresse du vecteur du dictionnaire le plus pertinent.
- FI_HERE est un signal binaire qui indique qu'un vecteur proche a été trouvé. Il servira lors du fonctionnement de plusieurs modules de quantification en parallèle.

3.3.5 Traitement parallèle

Pour augmenter la vitesse de quantification, l'architecture permet l'utilisation de plusieurs quantificateurs en parallèle. Par exemple on peut scinder le dictionnaire en N parties dont on alimente N quantificateurs, qui fonctionneront simultanément. Si le nombre de vecteurs du dictionnaire est divisible par N, le gain de rapidité est pratiquement de N fois.
Chaque vecteur d'entrée est proposé simultanément aux N quantificateurs. Les signaux de contrôle sont les mêmes. La seule difficulté consiste à trouver la distance minimale parmi les quantificateurs, et donc le vecteur du dictionnaire le plus pertinent. Dans la suite nous proposerons deux méthodes de parallélisation.

3.3.5.1 Première Solution

On a vu qu'un composant quantificateur transmet par le signal ZOUT l'inverse logique de la distance minimale sur 12 bits qu'il a trouvé dans la quantification d'un vecteur. On peut utiliser cette information pour comparer les composants entre eux, de la même façon qu'à l'intérieur de chacun d'eux les modules quantificateurs ont vécu un processus d'élimination.

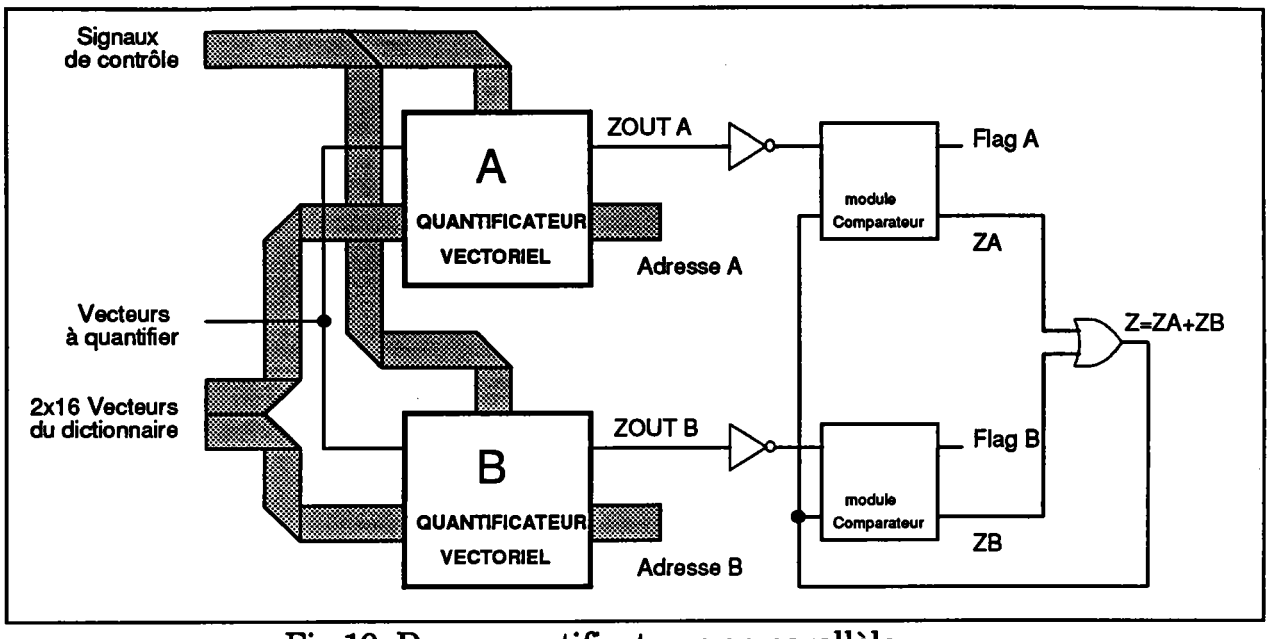


Fig 19: Deux quantificateurs en parallèle.

L'encodage final de l'adresse du vecteur, en fonction des Flags A et B, n'est pas représenté ici.
On généralisera aisément ce schéma à N quantificateurs.

3.3.5.1 Deuxième Solution

Une autre solution consiste à quantifier plusieurs vecteurs simultanément à l'aide du même dictionnaire.

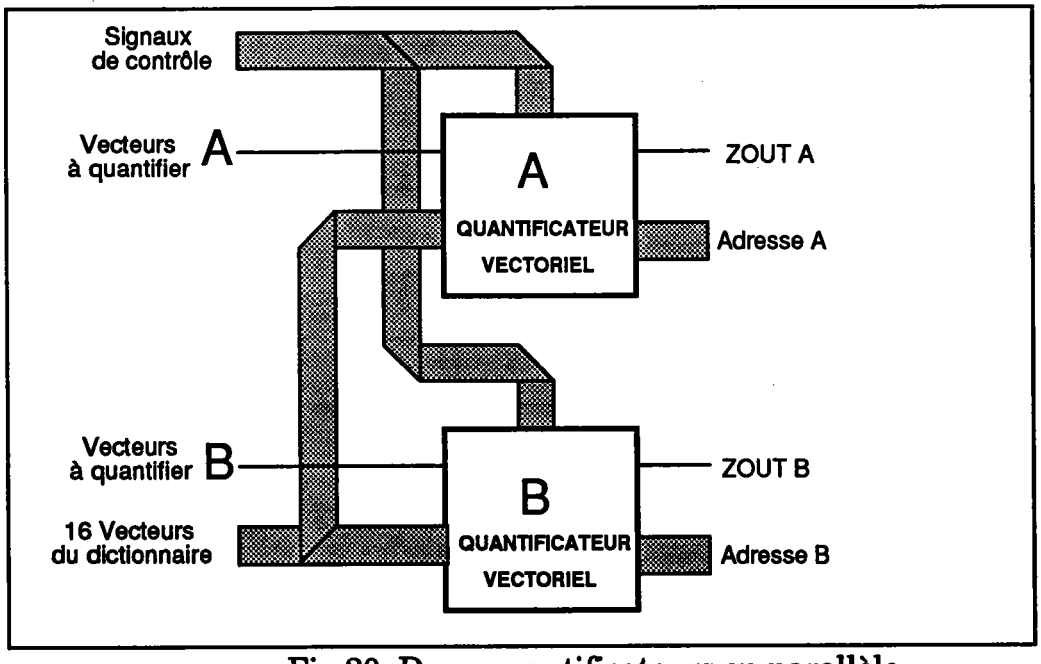


Fig 20: Deux quantificateurs en parallèle.

L'utilisation de N quantificateurs en même temps pour traiter N vecteurs différents permet de multiplier par N la vitesse globale de quantification. De

plus le câblage est simplifié puisque les signaux de contrôle peuvent être les mêmes, et le même dictionnaire est utilisé par les N composants.

3.3.6 Description et utilisation de la carte

Du point de vue de l'utilisateur, il faut voir le quantificateur comme un COPROCESSEUR spécialisé dans la quantification vectorielle. Il fonctionne de manière asynchrone ce qui permet de choisir librement sa vitesse de travail en fonction de la technologie dont on dispose.

3.3.6.1 Chronologie des opérations

Les simulations montrent que la quantification de vecteurs de dimension 3 prend 63 périodes d'horloge. En dimension 5 il faut 87 périodes d'horloge.

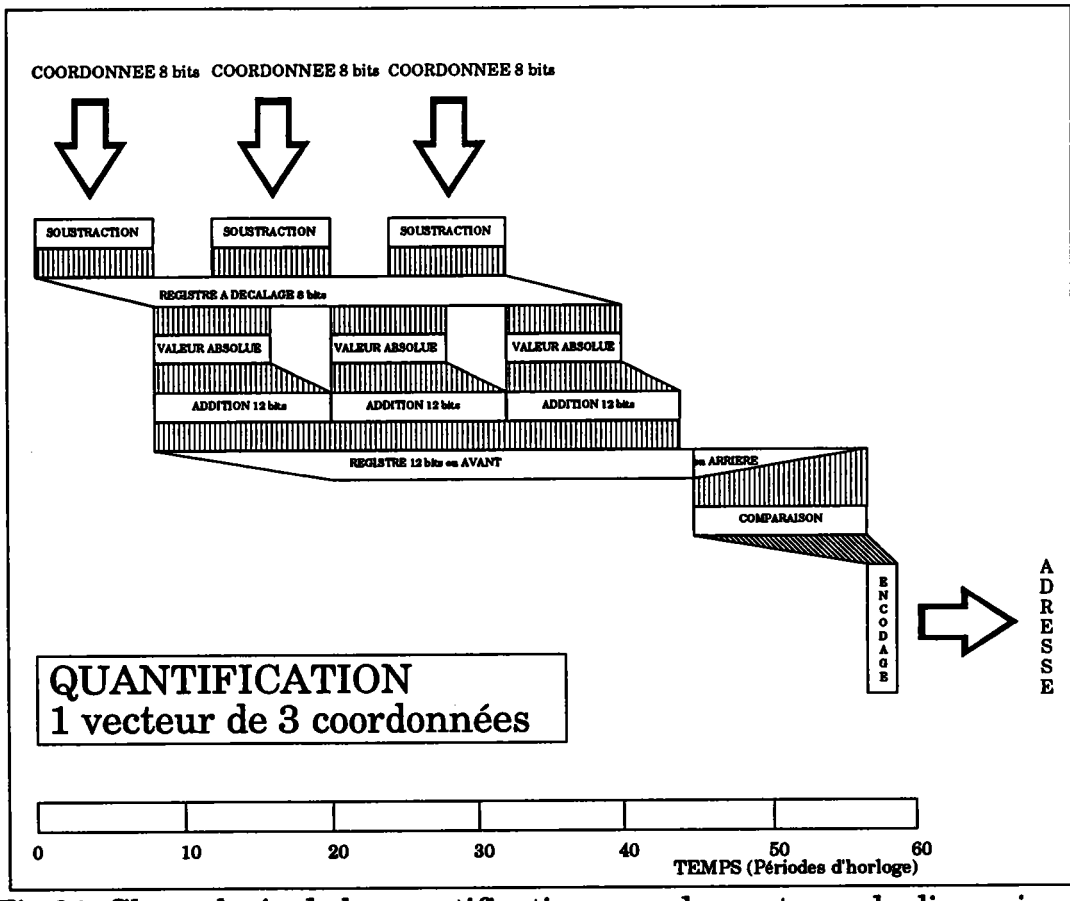


Fig 21: Chronologie de la quantification pour des vecteurs de dimension 3.

3.3.6.2 Technologie

La technologie utilisée est celle des FPGA (Field Programmable Gate Array) de XILINX (série 4010) pour la mise au point.
Une cellule FPGA est analogue à un petit PAL. On peut dire qu'un FPGA est constitué de beaucoup de petits PAL avec beaucoup d'interconnexions. Les blocs logiques, les plots d'entrées-sorties et les interconnexions sont

programmables en laboratoire.
XILINX est un outil de conception de réseaux logiques programmables (FPGA). C'est un logiciel qui fonctionne sur PC, muni d'une carte d'interface qui permet de programmer à partir du PC un circuit FPGA.

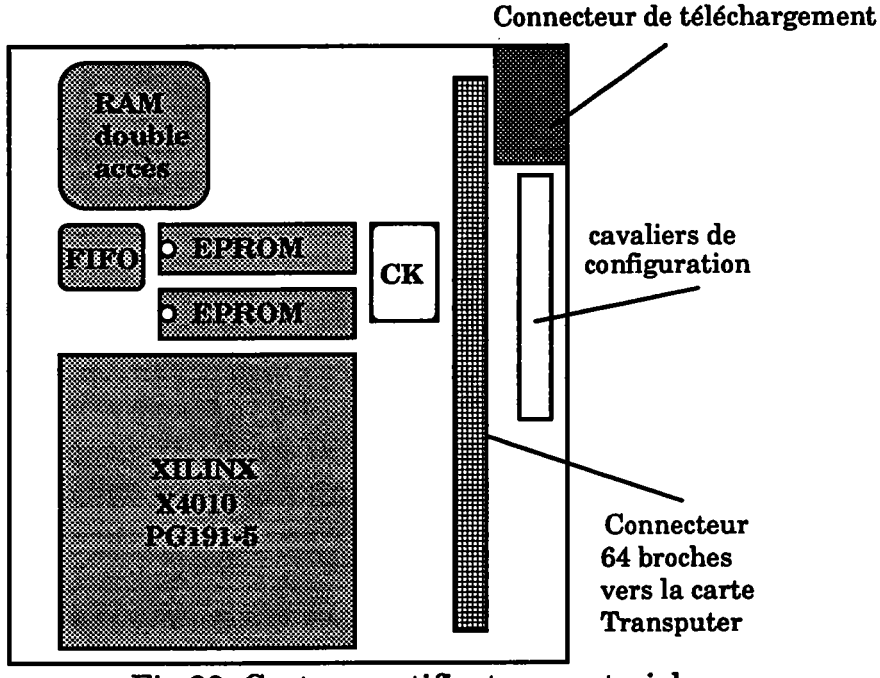


Fig 22: Carte quantificateur vectoriel

Le séquencement du quantificateur est contenu dans deux EPROM. On peut aussi le placer dans une RAM pour un fonctionnement dynamique du quantificateur. A noter que le séquencement dépend de la dimension des vecteurs et de la taille du dictionnaire.
Le dictionnaire est contenu dans une RAM double accès (accès par le Transputer + accès par le quantificateur). Ses vecteurs sont stockés bit par bit
Les coordonnées des vecteurs à quantifier sont placées dans une pile FIFO (First Input First Output) à entrée parallèle et à sortie série. La RAM double accès ainsi que les vecteurs du dictionnaire possèdent une adresse précise dans le champ mémoire du Transputer.
Le temps d'accès des RAM est de 55 ns. Cela impose une vitesse maximale d'horloge de 18 MHz.
Une horloge séparée pilote l'ensemble. Elle est indépendante de celle du Transputer. Le quantificateur n'a nul besoin d'être synchronisé sur le Transputer.

3.3.6.3 Principe de fonctionnement

Le principe de fonctionnement du quantificateur vectoriel est le suivant:

1- écrire le (s) dictionnaires dans la RAM à double accès à partir du transputer,

2 - envoyer le vecteur à quantifier vers le Q.V.,

3 - récupérer le résultat de la quantification (adresse du vecteur le plus proche).

3.3.6.4 Modèle d'utilisation

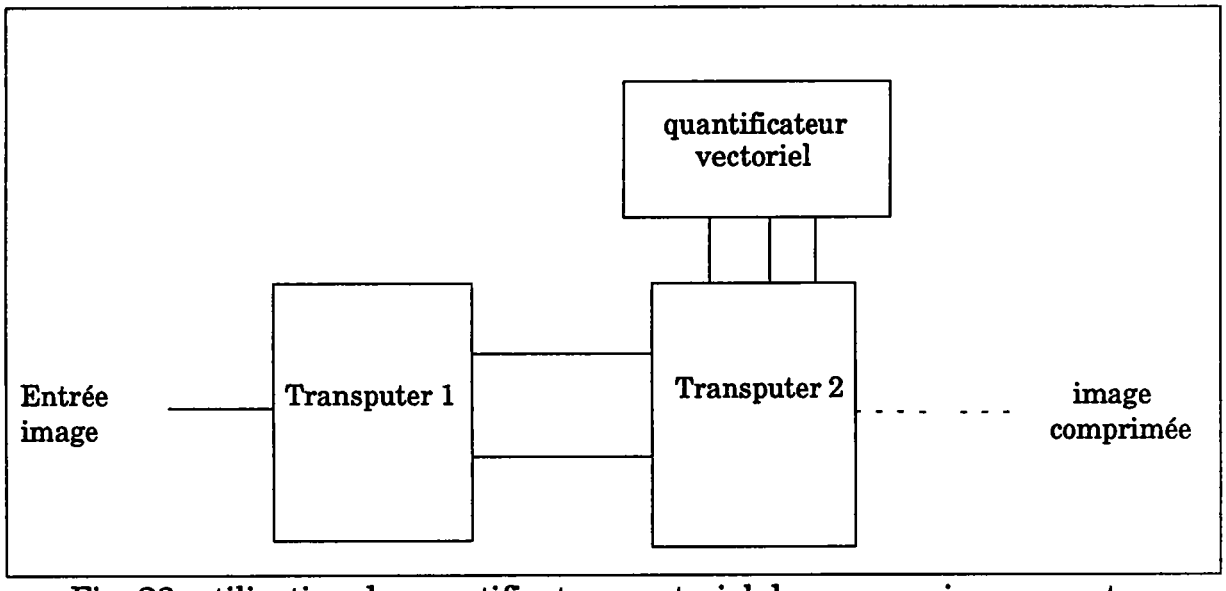


Fig. 23: utilisation du quantificateur vectoriel dans un environnement transputers.

Le quantificateur vectoriel est placé sur le transputer 2 (B430). Le transputer 1 (maître) envoie des vecteurs à quantifier au transputer 2 . Celui ci propose les coordonnées au quantificateur vectoriel, puis lit le résultat de la quantification et l'envoie au transputer 1 (ou à un autre processeur). Le dialogue entre le transputer 2 et le quantificateur vectoriel peut être schématisé comme suit:

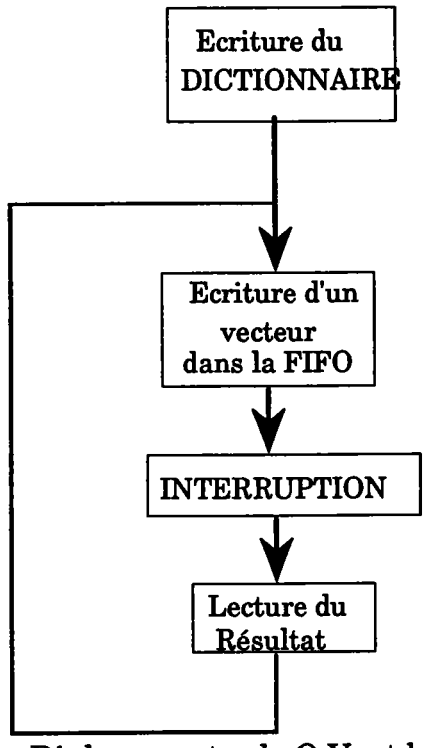


Fig 24 : Dialogue entre le Q.V. et le transputer

3.4 Performances comparées

Pour terminer voici quelques chiffres montrant l'efficacité du quantificateur vectoriel.
Dans la suite, on considérera les valeurs suivantes:

- Nd le nombre de vecteurs du dictionnaire,
- Tseq l'exécution séquentielle sur un seul transputer en seconde,
- Tps le temps de quantification par le Q.V. à 16 MHz en seconde,
- Tpar le temps total de traitement parallèle,
- Acc l'accélération.

Le découpage de l'image en blocs se fait au niveau du transputer T1 (maître) qui envoie les blocs un à un après avoir effectué la TCD sur le bloc. Et le transputer T2 se charge de l'extraction des vecteurs puis de leur quantification vectorielle ceci dans le cas où le quantificateur vectoriel n'est pas utilisé. Mais si on utilise le quantificateur vectoriel, T2 reçoit des vecteurs qui seront quantifiés par le composant puis envoie l'adresse du vecteur du dictionnaire le plus proche.

Nd	16	64	256
Tseq	20,87	39,56	112,19
Tpar	11,26	21,21	79,61
Acc	1,85	1,86	1,40

Tableau 2 : Performances sur 2 transputers sans le quantificateur vectoriel

Nd	16	64
Tseq (sec)	20,87	39,56
R (Q.V.,Tseq)	6,8	25,19
Tps(Q.V.)	0,309	0,379
Tpar	15,368	15,724

Tableau 3 : Performances sur 2 transputers avec le quantificateur vectoriel

Nd	16	64
Accélération sans la carte	1,85	1,89
Accélération avec la carte	1,35	2,51

Tableau 4: Mesures de l'Accélération

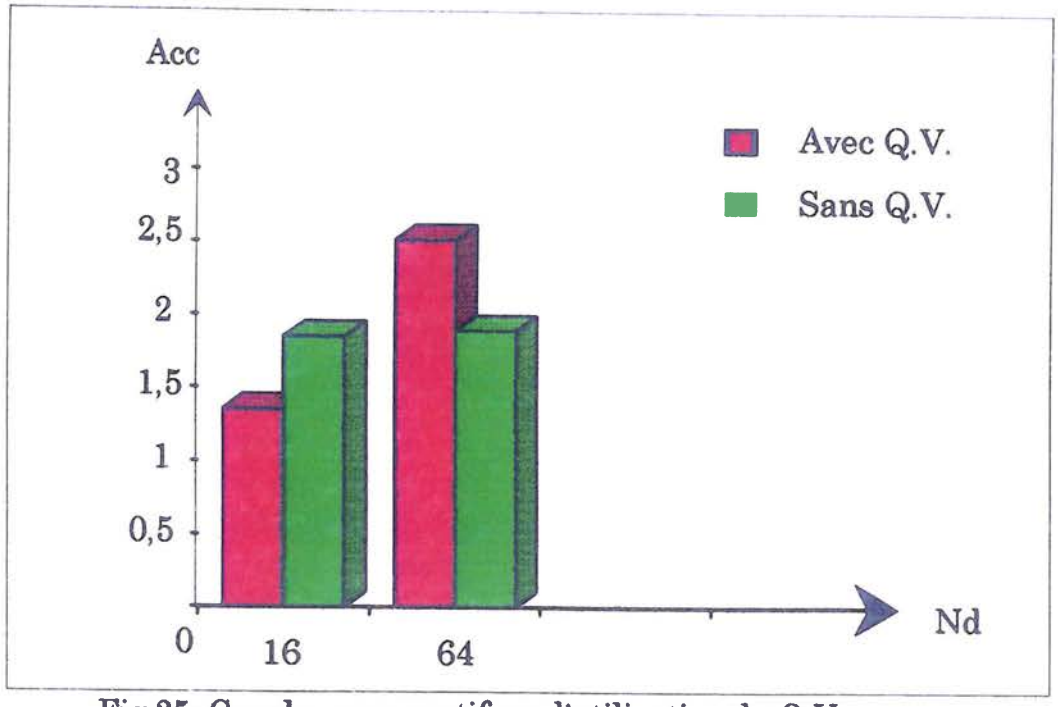


Fig 25: Graphe comparatif sur l'utilisation du Q.V.

L'utilisation de la carte donne des résultats très performants dans le cas d'un traitement avec un dictionnaire de taille élevée. C'est à dire pour des traitements pour lesquels le temps de traitement de la quantification vectoriel est significatif par rapport au temps de traitement total (dictionnaire de taille 64).

NB. Pour le moment nous n'avons pas encore mis au point un composant capable de traiter des dictionnaires de plus de 256 vecteurs mais le développement est en cours.

Aujourd'hui, bien qu'une partie importante de conception ait été réalisée, le quantificateur vectoriel est en mesure de fournir des résultats extrêmement performants. Pour ce faire, après avoir exploré et validé toutes les possibilités

ASIC. Cette évolution va permettre de gérer des dictionnaires de 64 vecteurs et de dimension 3, 5 , 8 ou 16. La fréquence d'utilisation sera elle aussi augmentée. Ce qui autorise soit la mise en parallèle de composants pour quantifier un vecteur avec 256 éléments du dictionnaire, pour des applications du type codage d'images, soit d'utiliser le mode vague pour des codages moins rapides.

4 Présentation du système final : codage - décodage

Nos recherches précédentes nous ont permis de mettre au point un dispositif de codage d'images à bas débit. Ce dispositif est une chaîne complète de traitement d'images basée sur une architecture parallèle. Elle exploite tous les avantages du parallélisme et des applications spécifiques y ont été implémentées. Notre objectif de transmettre deux images de qualité "télésurveillance" par seconde est atteint. Le processeur de base est le transputer. Mais, il existe d'autres processeurs spécialisés en traitement d'images ayant des performances supérieures à celles du transputer. Son utilisation dans notre système est justifiée par le modèle parallèle qu'il propose. Mais, il est tout à fait envisageable d'associer à chaque transputer un processeur spécialisé dans le traitement séquentiel pour former un processeur élémentaire plus puissant. Cette souplesse de l'architecture et cette puissance de calcul permettent son utilisation pour d'autres applications nécessitant le "temps réel".

4.1 Description du système

Ce système (Fig. 26) est une chaîne complète de traitement qui effectue la saisie de l'image à partir d'un capteur (caméra, rayons x,..), puis la traite avant de restituer l'image résultat sur un écran graphique de contrôle ou bien de la renvoyer à un calculateur ou à un processus automatique. Nous allons le décrire comme suit:

- 1) un Pu (Poste utilisateur) qui joue uniquement le rôle de serveur de fichier,

- 2) une carte d'acquisition vidéo (B429 d'INMOS), qui est placée sur le "root ". Elle joue le rôle de maître, elle est utilisée pour piloter les autres transputers. Elle fait l'acquisition et transmet l'image bloc par bloc au module coder_bloc,

- 3) une carte TCD transparente. Elle est placée entre deux transputers sur un lien. Elle permet de faire la TCD et la TCD inverse,

- 4) un module coder_bloc, qui peut être constitué d'un transputer ou de plusieurs transputers qui travaillent en parallèle pour déterminer les vecteurs à quantifier,

- 5) un module de quantification vectorielle constitué d'une carte B430 qui pilote un quantificateur vectoriel intégré,

- 6) un module de décodage et affichage, ce module est chargé du décodage et éventuellement de l'affichage. Il est constitué d'une

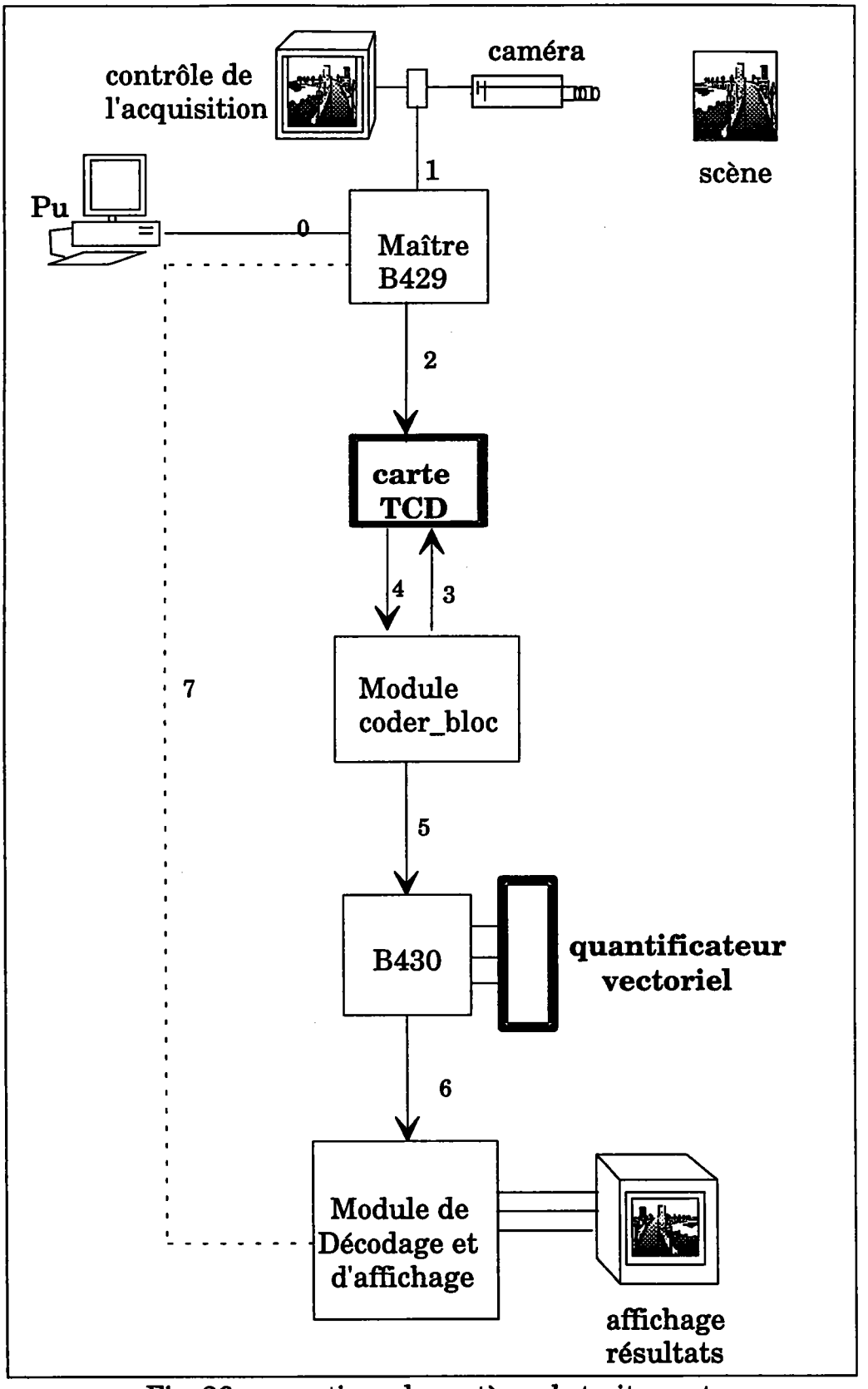


Fig. 26 : synoptique du système de traitement.

d'affichage graphique (B419 d'Inmos) qui peut être associée avec un ou plusieurs autres transputers en parallèle,

- 7) une caméra, un moniteur de visualisation des résultats et un moniteur de contrôle de l'acquisition (facultatif).

Le module Décodage et Affichage

Ce module effectue l'identification des codes et l'identification des vecteurs associés. En effet, pour chaque code reçu, l'identification permet de reconnaître la classe d'appartenance du bloc transformé codé, ainsi que la disposition de chaque vecteur transformé.
Il reconstruit le bloc image à partir de la moyenne et des blocs image élémentaires affectés aux vecteurs coefficients.
A noter qu'une TCD inverse est appliquée aux vecteurs du dictionnaire reçus pour obtenir des vecteurs de fréquences. Leurs adresses sont indiquées par le mot code du bloc.
A la sortie du module qui est directement relié à un moniteur pour la visualisation, l'image finale numérisée est ainsi reconstituée.

4.2 Sens des Communications (Fig. 26)

Par le fil 1 (lien que nous avons numéroté) passe :

- ❶ le signal vidéo provenant de la caméra, qui sera numérisé sur la carte d'acquisition (B429),

Par le lien 2 passent

- ❷ des blocs de pixels qui seront transformés par la carte TCD en blocs de coefficients TCD, (mode TCD)

Par le lien 3 passent:

- ❸ les ordres permettant de modifier l'état de la carte TCD,

Par le lien 4 passent :

- les coefficients TCD de ❷,

Par le lien 5 passent :

- ❹ la moyenne des coefficients TCD de ❷,
- ❺ les vecteurs à quantifier des coefficients TCD de ❷ provenant du module coder_bloc,

Par le lien 6 passent:

- les données ❹
- ❻ les adresses des vecteurs ❺,
- les dictionnaires chargés dans la RAM du quantificateur vectoriel,

Par le lien 7 passent :

- les dictionnaires (une possibilité supplémentaire d'envoi du

dictionnaire vers le module de décodage).

Utilisation

A l'initialisation il faut :

-1) transmettre les dictionnaires vers la carte B430 pour le quantificateur vectoriel. Les dictionnaires doivent être de taille multiple de 16. On transmet de mots de 16 bits contenant le ième bit de 16 vecteurs. 8 mots pour les 8 bits de chaque coordonnée avec le LSB en premier et le MSB à la fin,

-2) transmettre les dictionnaires au module de décodage soit à partir de la B430 ou du maître,

-3) puis le traitement commence. Les blocs pixels de l'image acquise sont envoyés un à un par le maître dans le réseau, ils subissent un traitement à chaque passage dans un module. Le module de décodage récupérera un fichier comprimé qu'il décodera et affichera sur le moniteur de visualisation résultats.

5 - Conclusion

Cette chaîne de traitement est conçue de manière souple que ce soit en ce qui concerne les applications (codages, analyse), son coût, ou sa vitesse de traitement.

L'utilisation d'une architecture parallèle lui confère une très grande puissance de calcul facilement modulable et adaptable à différents algorithmes.

Le codeur-décodeur est composé de modules indépendants, pouvant échanger des informations, conçues autour de transputers ou de processeurs de traitement du signal. Les différents processus du codage ou décodage sont exécutés indépendamment les uns des autres sur des modules capables d'échanger les informations.

Les différents modules de traitement peuvent être composés de transputers pouvant être configurés différemment suivant la complexité et la nature de l'application, ce qui fait qu'on est en présence d'un véritable outil de traitement et d'analyse d'images scientifiques. Le système peut utiliser tout autre processeur ayant des liens.

Le débit varie suivant la nature de l'application et la configuration des transputers dans les modules de traitement.

Son architecture permet encore d'améliorer le parallélisme de fonctionnement

ce qui augmente la vitesse de traitement, en prévoyant plusieurs cartes montées en parallèle (cartes TCD, quantificateurs vectoriels). Et chaque carte opérera sur un bloc de pixels par exemple.

Il permet également le développement aisé d'applications car les processeurs qu'il peut utiliser ont le mérite de pouvoir être facilement programmables, puisqu'il existe des bibliothèques écrites en C ou OCCAM qui permettent d'exploiter leurs possibilités. Ils fonctionnent sur un PC (ou station SUN) à l'aide de cartes enfichables, et on peut directement les programmer depuis un PC (ou depuis une station de travail).

BIBLIOGRAPHIE

[BRE-86] Brent E. Nelson et Chistopher J. Read "A Bit Serial VLSI Vector Quantizer ". IEEE Transactions , pp. 2211-2214, 1986

[BAS-90] Paul Basehore, Joseph Yestrebsky, Geral Reed, "Fuzzy Data comparator with neural network postprocessor a hardware implementation", Micro Devices, spécial products Division of Chip supply, May 1990.

[GRA-88] Grant A. Davidson, Peter R. Capello and Allen Gersho, "Systolic Architectures for Vector quantization", IEEE Transactions and acoustics speech. and Signal processing, Vol. 36, N° 10, pp. 1651-1664, October 1988.

[IPA-93] F.Idris et S. Panchanathan " Associative Memory Architecture for Video Compression " et " Adaptive Vector Quantizer for Image Coding " Departement of Electrical Engineering, University of Ottawa, Ontario, Canada, Kin 6N5, 1993

[KOP-90] T. Komarek et P. Pirsh " Array Architecture for Block Matching Algorithms "

[MID-90] Micro Devices " Fuzzy Set Comparator ", " Fuzzy Data Correlator " , Notes Techniques Micro Devices, May 1990.

[RBT-89] P. A. Ramamoorthy, Brahmaji Potu et Tien Tran " Bit-Serial VLSI Implementation of Vector Quantizer for Real-Time Image Coding " IEEE Transactions on circuits and systems, Vol. 36, N° 10, pp. 1281-1289, October 1989.

[SET-91] Sethuraman Panchanathan, Morris Goldberg, "A Content-Adressable Memory Architecture for Image Coding using Vector Quantization", IEEE Transactions Signal processing, Vol. 39, N° 10, pp. 1991-2066, Septembre 1991.

Chapitre IV

Conclusions et Perspectives

L'étude menée tout au long de cette thèse nous a permis de dégager une série de principes fondamentaux liés au parallélisme et au traitement d'images. Leur exploitation nous a conduit à la mise en oeuvre d'une machine de codage d'images à bas débit destinée dans un premier temps à la télésurveillance.

En effet, à travers la parallélisation d'un algorithme de codage d'images à bas débit, on a pu montrer que:

- le traitement d'images est un domaine privilégié pour la mise en oeuvre du parallélisme car il fait souvent appel à une grande puissance de calcul,
- la conception des algorithmes gagne à être pensée en terme de parallélisme,
- le parallélisme apporte des solutions efficaces au traitement d'images.

Cette étude nous a permis d'expérimenter plusieurs techniques de parallélisation sur un algorithme de codage d'images à bas débit sur machines MIMD. Malgré la séquentialité de l'algorithme, nous avons pu montrer que ce type d'algorithme pouvait s'implanter de façon très efficace dans un environnement parallèle. En effet, nous avons introduit un modèle de programmation par les données SPMD (Single Program Multiple Data). L'efficacité logicielle de l'algorithme fut cependant limitée par l'absence d'équilibrage de la charge des données sur les différents processeurs, ce qui nous a conduit à réaliser les parties logicielles de l'algorithme les plus gourmandes en temps de calcul par des cartes spécialisées. Pour ce faire , nous avons développé une carte TCD et un quantificateur vectoriel intégré. Cette machine de codage incorpore quelques concepts architecturaux très intéressants dont plusieurs ont fait l'objet d'une demande de brevet.

Inscrite dans un cadre technologique précis, le champ d'applications de cette machine est cependant plus vaste que le codage d'images (contrôle de processus industriel, détection de panne, biomédical, ... etc). Elle peut aussi être utilisée en traitement d'images animées.

Relativement prometteuse, elle nécessite toutefois une remise en question de certains choix technologiques pour être en mesure de fournir des résultats plus performants à savoir:

- l'interface de quantification devra être implanté dans un ASIC (Application Specific Integrated Circuit),
- le processeur de base à savoir le transputer est technologiquement dépassé car il existe d'autres processeurs qui sont nettement plus performants que le transputer.

En revanche, l'approche de l'architecture multiprocesseurs lui confère une très grande puissance et une très grande flexibilité.

Enfin, pour terminer, nous allons aborder l'évolution de cette machine. Une évolution ultime de la partie codeur, sera un ASIC .

En conclusion, nous espérons avoir montré que le parallélisme est un outil puissant pour accélérer le traitement des algorithmes en traitement d'image,

et que la symbiose de ces deux domaines complémentaires permet d'apporter non seulement des solutions efficaces au problème posé.

ANNEXE

LE TRANSPUTER

1- Présentation

Le transputer est un circuit VLSI réalisé en technologie CMOS, comprenant un processeur, une mémoire locale propre et quatre liens de communication série bidirectionnels à haute vitesse (20 Mbits par seconde) pour se connecter avec d'autres transputers. L'architecture du CPU est de type RISC. Le transputer est un com**PUTER** (ordinateur) monoboitier et un composant silicium comme un **TRANS**istor. Il est aux machines parallèles ce que le transistor est à l'électronique. Il peut d'ailleurs accéder à des ressources propres ou partagées à l'aide d'un bus parallèle conventionnel configurable.

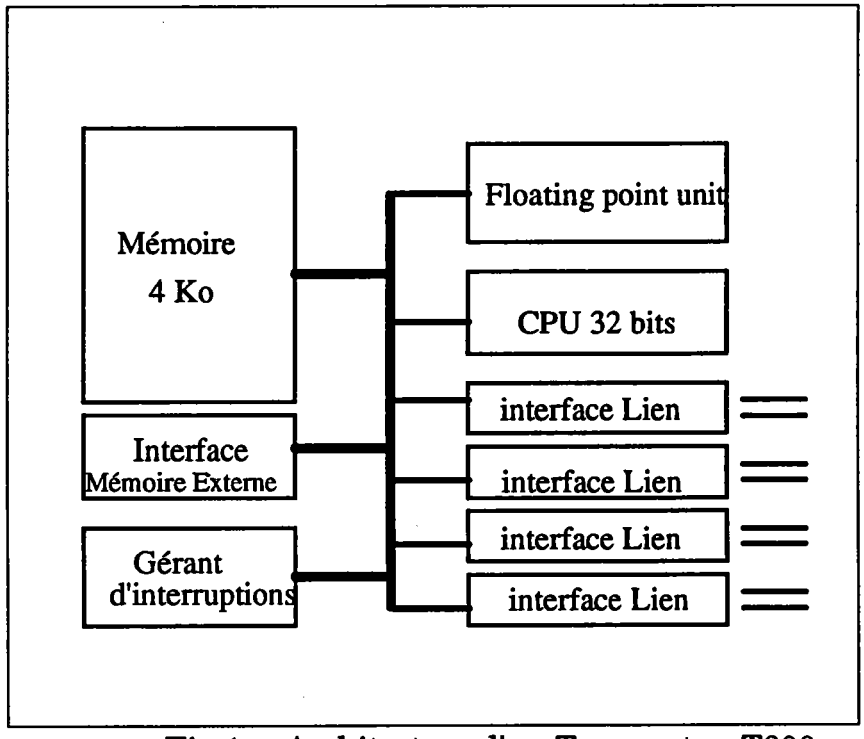


Fig 1 : Architecture d'un Transputer T800

Le T800 est un microcalculateur entièrement intégré sur un composant. Il comprend:

- un processeur central 32 bits (CPU) d'une puissance de 10 Mips avec un temps de cycle de 50 ns,
- une mémoire interne rapide. C'est une RAM statique avec un débit de 80 Moctet/sec et cela grâce à l'absence de tampon d'entrée/sortie, avec un temps d'accés de 50 ns,
- une interface rapide pour mémoire externe, permettant d'adresser 4 Goctetsde mémoire avec un débit de 26,6 Mo/sec.
- 4 liens indépendants de communication série bidirectionnels, permettant un débit maximum de 20 Mbits/sec,

- une unité de gestion des signaux externes (horloges, signaux d'erreurs, etc ...),

- une unité de calcul flottant capable d'effectuer 4,3 Mflops (millions d'opérations en virgule flottante par seconde) et compatible avec la norme IEEE 754.

2 - Avantages

C'est le premier microcalculateur spécialement destiné aux applications parallèles. Son architecture a été optimisée pour l'exécution de processus concurrents soit au sein d'un même transputer, soit sur plusieurs d'entre eux. Son architecture est basée sur la communication; les tâches sont considérées comme des blocs séquentiels dialoguant entre eux par l'intermédiaire de canaux de communication.

2.1 Processus

Dans le transputer, la notion de tâche est l'équivalent de processus(petit programme avec début et fin). Les processus communiquent entre eux au moyen d'un canal. Si les deux processus sont situés sur le même transputer, le canal est matérialisé par une case mémoire; sinon il est matérialisé par un lien de communication série.

2.2 Les Communications

La communication est l'échange d'informations entre deux processus. Il est possible d'établir une communication entre deux processus que ceux-ci soient traités par le même transputer ou par des transputers différents.
Un lien est constitué de deux liaisons unidirectionnelles de type série transmettant à la fois les données et les bits de contrôle. Le lien est bidirectionnel. Les transferts (en mode DMA) entre deux transputers, automatiquement synchronisés, s'effectuent par séquence d'octets. Les quatre liens peuvent assurer des transferts simultanés, ce qui porte le débit global à 8 Mo par seconde. La simultanéité du traitement entre l'unité centrale et les liens est totale. Les communications se font suivant le modèle CSP (Communicating Sequential Processes).

Dans le formalisme CSP, une communication a lieu quand un processus émetteur désigne un autre processus comme destinataire d'une sortie et que ce destinataire désigne le premier comme source d'information. De plus, il n'y a aucune possibilité pour l'émetteur de continuer son calcul tant que la valeur émise n'a pas été transmise au destinataire. Sur le plan général, l'exécution d'une commande d'entrée ou de sortie est retardée tant que le

processus correspondant n'est pas prêt à exécuter la sortie ou l'entrée correspondante .

Les liens du transputer sont l'équivalent matériel du concept logiciel de canal.

2.3 Les timers

Le transputer contient deux timers 32 bits intégrés. Le premier est accessible par les processus de priorité haute et incrémenté chaque micro-seconde. Le second pour les priorités basses est incrémenté toutes les 64 micro-secondes. Le "scheduller" intégré dans le CPU, attribue à chaque processus d'un transputer, le temps nécessaire à son exécution entre deux communications ou le bloque pendant un certain temps pour laisser la place à des tâches plus urgentes.

3- Langages de programmation

Le transputer a été conçu pour rendre la programmation parallèle aussi simple que possible. Il accepte cependant n'importe quel langage de haut niveau de modèle CSP, comme Pascal, Fortran, C, etc ... Mais son langage de prédilection est Occam.

3.1 Occam

C'est un langage de haut niveau CSP, adapté au transputer. Il a été developpé antérieurement au transputer à l'université d'Oxford. Occam manipule des processus, c'est à dire des petits programmes avec un début et une fin. Ces processus échangent des messages entre eux par l'intermédiaire de canaux lorsqu'ils appartiennent au même transputer, ou bien par des liens de communication lorsqu'ils n'appartiennent pas au même transputer.

Le couple occam-transputer présente plusieurs caractéristiques intéressantes:

- processeur adapté à la conception des machines MIMD avec mémoire locale et communication par messages,
- transparence de la gestion des envois et de réception des messages grâce à l'unité de lien,
- ajustement de la puissance de calcul aux besoins de l'application (traitement d'image) par ajout de microprocesseurs,
- facilité de construction et d'utilisation d'une machine parallèle.

3.2 le langage C

Sa flexibilité inhérente , sa capacité de mixer du code provenant de langages différents, et sa capacité à exploiter la topologie concurrente du Transputer font du langage C d'Inmos compatible avec la norme ANSI un outil puissant pour la programmation des systèmes concurrents.

- exécution du compilateur plus rapide,
- nouveaux outils de débogage incluant un débogueur symbolique interactif avec des points d'arrêt, un débogage post-mortem amélioré et un simulateur de transputer sur la machine hôte,
- langage de configuration mis à jour pour supporter les futures générations de transputers,
- librairie améliorée de fonctions parallèles,
- mixage amélioré du C et du langage OCCAM.

4 Modèle de programmation

Le modèle de programmation parallèle consiste en un nombre de processus indépendants s'exécutant simultanément et communiquant à travers des canaux. Ces canaux sont les chemins de communication par lesquels les processus s'échangent des données.

Un processus peut être généré à partir d'un nombre quelconque d'autres processus parallèles, de telle sorte qu'un système software entier peut être décrit comme une hiérarchie de processus parallèles communiquant entre eux. Ce modèle est conforme à plusieurs approches modernes de programmation parallèle. Par exemple la programmation objets avec passage de messages.

La communication entre processus est synchronisée. Lorsqu'une information est transmise d'un processus à un autre, le processus émetteur ne peut continuer tant que le processus récepteur n'est pas prêt.

Des processus parallèles peuvent être exécutés indépendamment sur un ou plusieurs processeurs connectés en réseau. Un langage spécial de configuration est utilisé pour distribuer les processus à travers le réseau de Transputers et peut ainsi être utilisé pour programmer des systèmes multiprocesseurs complexes.

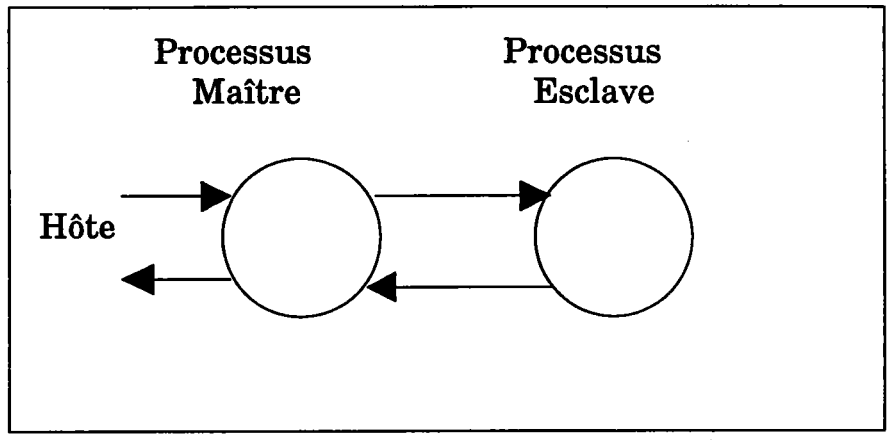


Fig 2 : Implémentation de 2 processus parallèles sur un seul Transputer.

Supposons par exemple que ces deux processus vont être implémentés sur deux Transputers.l Le processus Maître sur le processeur Maître et le processus Esclave sur le processeur Esclave.
On va procéder de la manière suivante:

- écriture des deux programmes correspondant à chaque transputer,

- compilation et édition de liens de chacun de ces deux programmes,

- puis création d'un programme de supervision des deux processus.

Ce programme va donc permettre au système de connaître quelles sont les différentes tâches qui vont s'exécuter, et avec quels canaux les communications vont s'établir.

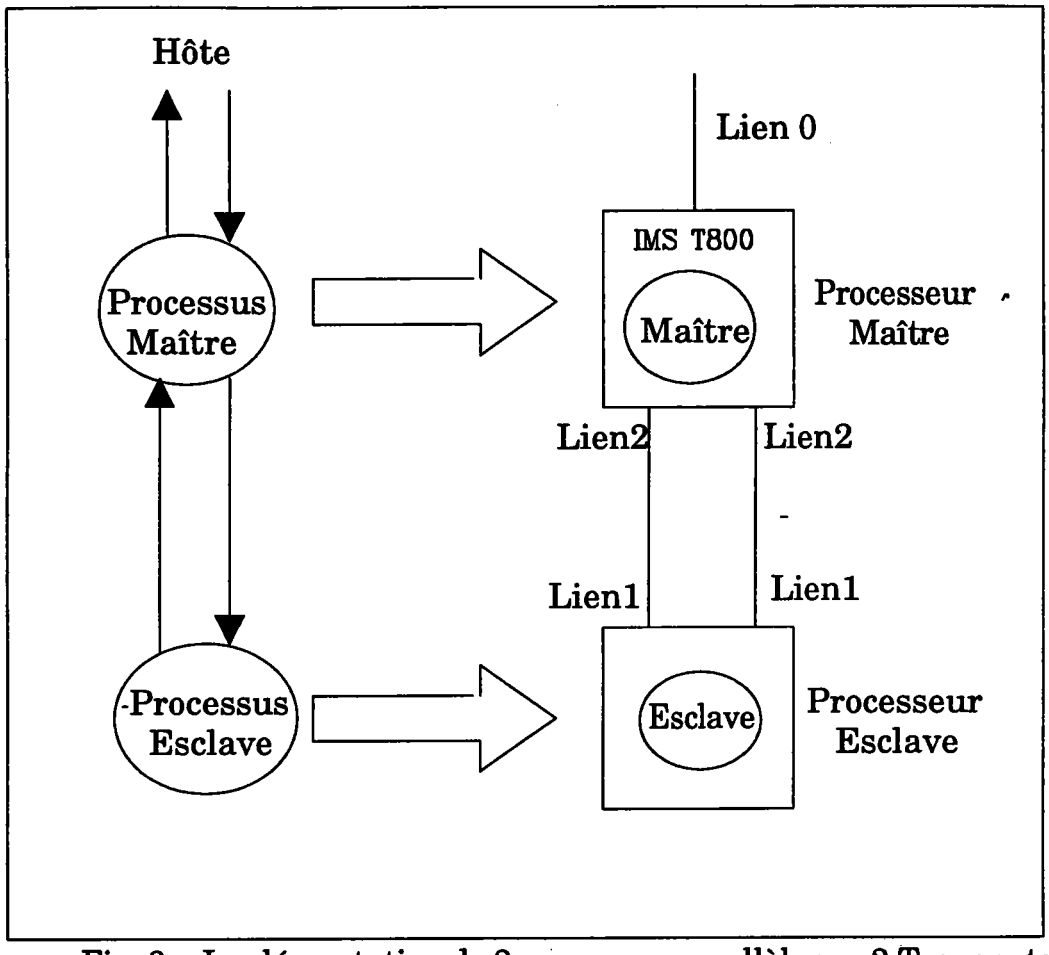


Fig 3 : Implémentation de 2 processus parallèle sur 2 Transputers.

4.1 Programmation temps réel

La topologie concurrente du transputer est particulièrement adaptée à la programmation temps réel. Ses caractéristiques sont décrites ci-dessous :

- implémentation directe et efficace de processus parallèles dans l'architecture du transputer,

- mise en oeuvre de priorités entre plusieurs processus concurrents de manière à optimiser le temps d'exécution de séquences critiques,

- possibilité d'implémenter des interruptions software comme processus de haute priorité,

- programmation facile de l'horloge interne du transputer (timer) de façon à pouvoir contrôler le temps alloué aux différents processus constituants du programme.

5- Utilisation

On peut distinguer trois catégories de besoins satisfaits par des solutions à base de transputers, auxquelles correspondent trois classes de produits:

- pour un supplément de puissance, une carte additionnelle ne comportant qu'un seul transputer peut apporter à des applications précises de bonnes performances. En effet la carte va jouer le rôle d'une carte coprocesseur programmable. Et le gain dépendra autant de la performance des outils logiciels utilisés que de la manière dont a été posé le problème à résoudre,

-nécessité de traitements parallèles pour résoudre des problèmes de type "temps réel". Problèmes qui se posent souvent dans les applications graphiques ou industrielles. Alors le transputer peut être utilisé soit sous forme de carte additionnelle (applications d'imagerie en temps réel, animations graphiques ou contrôle de processus), soit incorporé dans des cartes offrant un service particulier,

- le transputer apporte une réponse de choix dans la satisfaction des besoins en calcul des laboratoires et des grandes sociétés (usage de grands calculateurs spécialisés).

D'un point de vue architecture matérielle des systèmes temps réel, les systèmes à processeurs multiples s'orientent vers la solution répartie en raison de sa relative simplicité de mise en oeuvre sans dégradation significative de performances; la solution distribuée se prête à priori plus aux transputers qu'aux processeurs classiques pour lesquels l'accroissement des performances reste dans ce cas borné.

Cette thèse est une contribution à la parallélisation d'algorithmes de traitement du signal. Elle justifie la nécessité du parallélisme en traitement du signal et en traitement d'images en particulier. En s'appuyant sur un exemple précis, il nous est montré que le traitement d'images est un domaine privilégié pour l'étude et la mise en oeuvre du parallélisme.

En effet au travers d'un algorithme de codage d'images à bas débit, plusieurs techniques de parallélisation ont été mises au point. Les parties logicielles critiques ont été remplacées par des cartes prototypes spécialisées. C'est ainsi qu'une carte TCD (Transformée en Cosinus Discrète) et un quantificateur vectoriel intégré ont été développé pour réaliser une machine parallèle basée sur un réseau de transputers, destinée dans un premier temps au codage d'images à bas débit. Cette machine incorpore quelques concepts architecturaux très intéressants dont plusieurs ont fait l'objet d'une demande de brevet.

Inscrite dans un cadre technologique précis, le champ d'application de cette machine est plus vaste que le codage d'images (contrôle de processus industriel, détection de panne, biomédical, ... etc.).

Mots clefs: parallélisme, SPMD, transputer, traitement d'images, codage, TCD, quantificateur vectoriel, Xilinx, FPGA