621.038

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Centre Universitaire 8 Mai de Guelma
Institut de Génie Electrique

دائرة الهندسة الكهربائية

THESE DE MAGISTER

en

Electronique

Option : Informatique Industrielle et Imagerie.

THEME

Etude de Performances de Réseaux MMD de Transputers sur des Algorithmes de Traitement d'Images Bas niveau. Application à la reconnaissance automatique de pièces de monnaie algériennes.

Présentée par Mr: Debbi Aimad Eddine

Devant le jury composé de
Messieurs :

- A. YOUNSI	M.C	Centre universitaire de Guelma	Président.
- A. KHELLAF	Professeur	Université de Sétif	Rapporteur.
- A. BAKHTI	C.C	Université de Sétif	Rapporteur.
- A. BOUKERRAM	C.C	Université de Sétif	Examinateur.
- H. TEBBIKH	M.C	Centre universitaire de Guelma	Examinateur.

1998

Remerciements

Je tiens à remercier Monsieur A.KHELLAF Professeur à l'institut d'électronique de Setif pour le soutien de ce travail et pour la bienveillance qu'il a constamment témoignée.
Mes vifs remerciements vont également à Monsieur A.BAKHTI Chargé de cour à l'institut d'électronique de Setif pour sa contribution à l'élaboration de ce travail et son aide précieuse apportée à sa finalisation. Qu'il trouve ici l'expression de ma profonde reconnaissance.

Que Monsieur A .YOUNSI Maître de conférence à l'institut de Génie Electrique de Guelma trouve ici l'expression de mes remerciements les meilleurs d'avoir accepter de présider le jury.
Que Monsieur A. BOUKERRAM chargé de cour à l'institut d'informatique de Setif et Monsieur H.TABBIKH Maître de conférence à l'institut de Génie Electrique de Guelma , s'en trouvent de même remercier de bien vouloir juger le travail.

A tout ceux qui, de prés ou de loin ont porté d'aide, je dit merci.

Sommaire

Introduction.....1.
Chapitre I : systèmes parallèles et approche de parallélisme.....3.
I.1.1 Loi d'AMDAHL.....3.
I.1.2 Interprétation.....6.
I.1.3 Exemples.....6.
I.2 Classification de Flynn.....7.
I.3 Les multiprocesseurs (Structures et interconnexion).....9.
I.4 Paramètres d'évaluation de performances.....12.
I.5 Présentation de la stratégie de parallélisme.....14.
Chapitre II : Les Traitements d'images et leurs parallélismes...22.
II.1 Classement des traitements d'images.....22.
II.1 Restauration et rehaussement d'images.....22.
II.1.1 L'ajustement du contraste.....23.
II.1.2 La correction photométrique.....24.
II.1.3 Le filtrage spatial.....24.
II.1.4 La détection de contours.....27.
II.1.5 L'égalisation d'histogramme.....29.
II.2 La compression d'imagés.....30.
II.2.1 Les méthodes de compression sans pertes.....31.
II.2.2 Les méthodes de compression avec pertes.....31.
II.2.2.1 Le codage par troncature.....31.
II.2.2.2 Le codage par blocs.....31.
II.2.2.3 Le codage par transformé.....31.
II.3 L'analyse d'images.....34.
II.3.1 La segmentation.....34.
II.3.1.a la morphologie mathématique binaire.....35.
II.3.1.b la morphologie mathématique en échelle de gris.....38.
II.3.2 La caractérisation.....40.
II.3.3 La classification.....43.
Chapitre III : Aspects matériels et logiciels.....44.
III.1 Les transputers IMS T800.....44.
III.2.1 Caractéristiques.....44.
III.2.2 Architecture de base.....44.

III.2.3 Le codage des instructions . . . . . .45.
III.2.4 Les liens de communication . . . . . 45.
III.2.5 La configuration de réseaux de transputers . . . . . 46.
III.2.6 Les performances du transputer. . . . . .49.
III.3 Le langage parallèle OCCAM . . . . . 50.
III.3.a Les primitives . . . . . 50.
III.3.b La communication . . . . . 50.
III.3.c Les structures . . . . . 51.
III.3.d Les déclarations et les types . . . . . 51.
III.3.e Les procédures et les fonctions. . . . . 52.
III.3.f Les canaux et protocoles de communication. . . . . 53.
III.3.g Les boucles . . . . . 54.
III.3.h La configuration. . . . . 54.
III.4 Le système de développement pour transputers TDS . . 55.
III.4.1 L'éditeur. . . . . 55.
III.4.2 Le compilateur. . . . . 56.
Chapitre IV : Analyse de performance . . . . . 58.
IV.1 Configuration et stratégie de mesure. . . . . .58.
IV.2.1 Mesures prélevées pour la 1ère classe de traitements. . 62.
IV.2.2 Mesures prélevées pour la 2ème classe de traitements. .67.
IV.2.3 Mesures prélevées pour la 3ème classe de traitements. .71.
IV.2.4 Mesures prélevées pour la 4ème classe de traitements. .72.
IV.3 Simulation pour un grand nombre de transputers(6, 8, 9, 12) . . . . . .74.
IV.3.1 Mesures et courbes pour la 1ère classe. . . . . .77.
IV.3.2 Mesures et courbes pour la 2ère classe. . . . . .79.
IV.3.3 Mesures et courbes pour la 3ème et 4ème classe. . . . . .81.
IV.4 Interprétation et conclusions. . . . . .82.
Chapitre V : Application : tri de pièces de monnaie. . . . . 86.
V.1 Présentation du problème. . . . . 86.
V.2 La procédure de tri . . . . . 87.
V.3 *Mapping* de la procédure sur trois transputers . . . . . .95.
Exemples de programmes. . . . . .96.
Conclusion. . . . . 107.
Bibliographie. . . . . .108.

• Introduction

Pour répondre aux besoins nouveaux dans les secteurs industriels, médicales, et de recherche scientifique, les concepteurs de calculateurs travaillent sans cesse pour augmenter la rapidité de leurs produits. L'évolution des systèmes informatiques était due essentiellement aux progrès technologiques durant les dernières décades. Mais qui aujourd'hui approchent des barrières physiques au niveau des échelles d'intégration les plus élevées dans les semi-conducteurs(*VLSI*). Les tendances futurs comme proclamées dans les milieux informatiques s'orientent vers l'approche du parallélisme, qui constitue la préoccupation du jour.
Le présent travail rentre dans ce cadre. Les infrastructures et les systèmes parallèles commercialisés relèvent de choix architecturaux divers, et plusieurs aspects logiciels (algorithmique, programmation, compilation ...) interviennent pour la réalisation du parallélisme sur une plate-forme matérielle donnée. L'approche sur laquelle on s'est penchée dans cette étude se base sur le principe de l'adéquation du système physique avec l'ordonnancement de l'algorithme à paralléliser, ce qui est désignée encore " ***Adéquation algorithme architecture (AAA)*** ". La topologie définie pour un problème donnée est par conséquent une sorte de schéma d'occurrence algorithmique se rapportant à un graphe de ***dépendance de données***. D'une point de vue théorique les meilleurs performances seront d'autant plus approchées que les modèles sont plus raffinés et l'on parle ainsi de " ***fine grained hardware*** ".
L'analyse en cours a trait à ce qui est désigné par ***"co-désigne "*** et ce qui est dit ***add-on- board.*** Autrement dit le système parallèle utilisé est un réseaux de processeurs relativement de faible coût sur une carte additionnelle relogeable dans un PC. Parmi les systèmes qui se rapporte à ce modèle certains permettent des taux de raffinement importants car les éléments processeurs ***EP*** composants le système sont de plusieurs centaines (réseaux systoliques) [18]. D'autres utilisent pour le moment dans une étape expérimentale des réseaux avec un nombre de processeurs limité mais à puissance de calcul plus grande (DSP) [16]. Le matériel expérimenté ici vient comme intermédiaire entre ces deux tendances et est composé d'un certain nombre de processeurs dits *transputers* de la (série ***T800***) : comme réseau configurable. Chaque transputer est doté de 4 liens séries bidirectionnels. La configuration du réseau ainsi que l'allocation et la distribution des taches est établie à travers les primitives du langage ***OCCAM*** qui constitue la couche logicielle associée. Le travail se présente comme suite :

On commence dans le premier chapitre par passer en revue la loi d'amdahl. Celle ci met en rapport les performances maximales tolérables par un moyen de parallélisme quelconque et le degré du potentiel parallèle intrinsèque de l'application à paralléliser. Toujours dans ce chapitre serons exposés les différants modèles de calculs parallèles, les classifications des systèmes parallèles, leurs modes d'organisation et leurs modes d'interconnexion. Aussi on examinera la notion de performance en donnant les définitions de ses mesures les plus usuelles et des paramètres d'évaluation communément adoptés.
On présentera la structure PPF (*Pipeline Processor Farm*) comme schéma de parallélisme. On en discutera la performance et on en examinera le comportement au terme de deux facteurs essentiels (la latence et le *throughput*). Le raffinement de cette approche sera de même examiné de manière à explorer le maximum du potentiel parallèle intrinsèque.
Dans le deuxième chapitre seront rappelés les différents algorithmes de traitement d'images les plus répondues et leur potentiel parallèle sera discuter. On essayera de déduire les meilleurs rapports de performances acquittables dans les algorithmes à potentiel parallèle significatif et on développera les modèles de leur parallélisme.
A travers le troisième chapitre on présentera la série IMS T800 des transputers qui constitue le système sur lequel notre étude est expérimentée ainsi que la couche logicielle associée qui est le langage OCCAM et le système de développement pour transputers (TDS).
On décrira l'architecture de base de cette classe de processeurs et on insistera sur les caractéristiques essentielles concernant l'utilisation réseaux, à savoir les liens de communication et les modalités de configuration. Les primitives essentielles du langage OCCAM et leur utilisations dans les systèmes parallèles seront aussi présentées ainsi que les fonctions spéciales intégrées dans les utilitaires du système de développement TDS.
Au chapitre suivant seront présentés différentes implémentations parallèles des algorithmes déjà exposés au chapitre II et les mesures des temps d'exécution associées. Les performances requises avec ces différentes organisations sont modélisées par des courbes et des conclusions concernant l'impact des *overheads* et le rapport (performances/communication) sont établis.
Au dernier chapitre sera développé un exemple d'application industrielle temps réel consistant en un tri de pièces de monnaie algériennes de manière à mettre en évidence en particulier le modèle PPF, son raffinement, et la logique associée introduits tous au chapitre I, et aussi de façon à analyser les critères de principes intervenants lors d'une *AAA*. On terminera le mémoire par une conclusion.

Chapitre I

Systèmes parallèles et approche de parallélisme

I.1.1 La loi d'Amdahl :

La notion du parallélisme n'est pas toute à fait récente comme on peut le penser, mais, remonte plutôt aux premiers écrits sur les ordinateurs à programme enregistré. On avait trouvé dés 1967 déjà, certains travaux sur lesquels sont fondus quelques principes de base du parallélisme. Parmi ceux-ci nous vient la loi d'AMDAHL en premier plan.
AMDAHL [16]formula que l'accélération maximum possible par un dispositif de parallélisme pour l'exécution d'une tache est l'inverse de la fraction séquentielle inhérente de son algorithme. Soit : $S = \frac{1}{f}$ S : accélération (Speedup)

f : fraction séquentielle dans l'algorithme de l'application considérée.

Avec un nombre de processeurs N impliqué, la loi d'AMDAHL est donnée par :

$$S = \frac{1}{f + \frac{1-f}{N}}$$

N : nombre de processeurs

Autrement dit, si on se donne une application ayant une fraction séquentielle f, l'accélération S max qu'on pourra atteindre, en faisant exécuter la tache sur un nombre N de processeurs d'une manière concurrente, est donnée par la formule ci-dessus.

Exemple :

Si f=0.2 et en considérant un nombre de processeurs illimité $N \longrightarrow \infty$

On aura : $S_{max} = 5$

CALCULS AUX LIMITES :

Les calculs aux limites nous conduisent aux résultats suivants :

- Si le nombre de processeurs est illimité le Speedup est l'inverse de f.

$$\lim_{N \to \infty} S = \frac{1}{f}$$

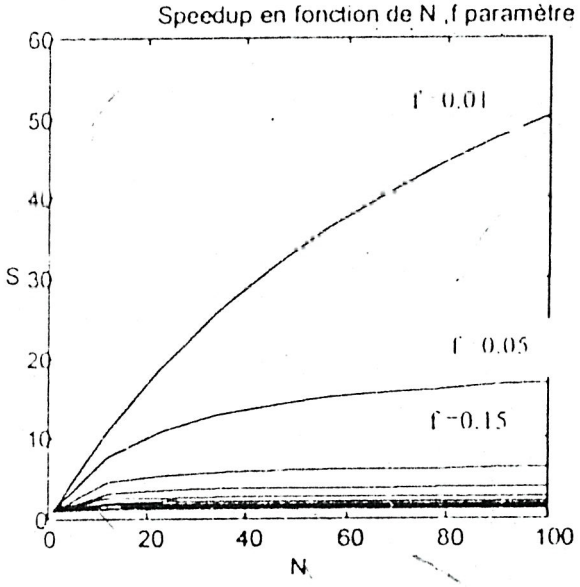
- Si l'algorithme est totalement parallèle (f=0) l'accélération est égale au nombre de processeurs utilisés. Ce qui est l'ultime recherché.

$$\lim_{f \to 0} S = N$$

- Si l'algorithme est totalement séquentiel (f=1) aucune accélération n'est possible quelque soit le nombre N de processeurs utilisés.

$$\lim_{f \to 1} S = 1 \quad \forall N$$

Le tracé de la loi d'AMDAHL est donné ci-dessous pour un échontillon de valeurs de f.



$$S = \frac{1}{f + \frac{1-f}{N}}$$

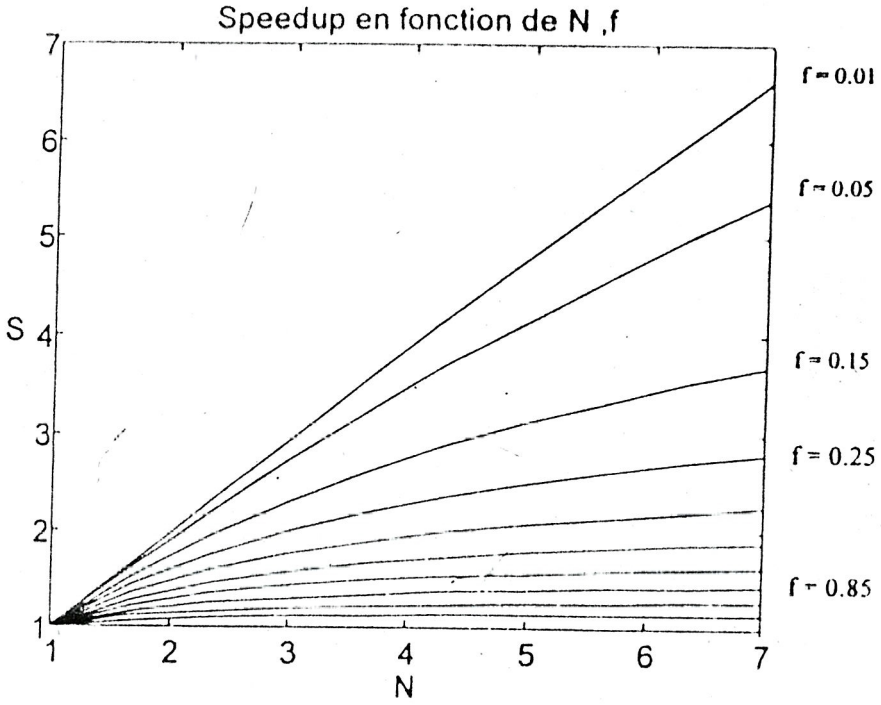
-Fig I.1.a : Speed-ups donnés par la loi d'Amdahl

Si N est de l'ordre de 100 :

Pour f = 0.01 $\longrightarrow$ S ≅ 50
Pour f = 0.05 $\longrightarrow$ S ≅ 17
Pour f = 0.1 $\longrightarrow$ S ≅ 9
Pour f > 0.5 $\longrightarrow$ S < 5

On peut déjà remarquer qu'avec même un algorithme d'une fraction de 1% seulement séquentielle sur 100 processeurs on n'atteint que la demi accélération idéale (100/2).
Et pour un algorithme de 10% de fraction séquentielle l'apport d'accélération n'est pas aussi significatif, S=10 contre 100 processeurs qui fait un rapport de 1 dixième.

De même, pour un nombre de processeurs limité, il semble que le Speedup n'est assez élevé que pour des algorithmes hautement parallèles. (voir Fig I.1.b).



-Fig I.1.b : Speed-ups pour un nombre de processeurs N limités.

La loi d'AMDAHL peut être aussi formulée sous forme d'un tableau similaire à celui représenté ci-dessous et qui peut servir de guide de prédiction des performances pour la parallélisation d'une application donnée.

Fraction séquentielle f	Nombre ressources N	Speedup S
0.01	50	33.5
	20	16.8
	7	6.6
0.05	10	6.9
0.09	5	3.6
0.1 0.2 0.3 0.5	5	3.5
	5	2.7
	4	2.2
	4	1.6
0.6 . .	∀ N	Non Significatif

I.1.2 Interprétation :

Nous pouvons donc conclure ce qui suit :

° On doit toujours avoir une estimation même approximative du potentiel parallèle intrinsèque de l'application avant toute tentative de parallélisme.

° L'accélération qu'on pourra atteindre n'accédera en aucun cas l'inverse de la fraction séquentielle dans l'algorithme considéré.

° L'idéal est d'arriver à assurer un speed-up S avec juste (N=S) ressources.

Aussi nous avons montré qu'un parallélisme est d'autant plus efficace que son potentiel parallèle intrinsèque est élevé. Alors que La quasi-totalité des algorithmes trouvés dans la littérature étaient écrite dans le contexte de la machine séquentielle sans accorder d'intérêt à l'aspect du parallélisme, le potentiel parallèle dans ces derniers restait très rudimentaire. Aujourd'hui alors comme les nouvelles plates-formes conçues autour des systèmes concurrents commencent sérieusement à s'imposer, la révision de certains algorithmes s'impose de même, et on pourra bien être amenés à réviser ce qui avait été développé comme algorithmes ou bien encore réécrire certains catégoriquement. Cela pourra bien entraîner aussi à de nouvelles sortes de discipline d'écriture d'algorithmes où la première motivation est l'augmentation du potentiel parallèle intrinsèque.

I.1.3 Exemples :

Afin de mettre en évidence l'aspect des potentiels séquentiels et parallèle dans une application donnée et les Speed-up associés, nous considérons les deux exemples suivants : le premier exemple porte sur évaluation d'une expression simple.
Le deuxième exemple concerne l'algorithme de Gauss-Jordan.

i. soit à évaluer l'expression :

$$V = (a + b)*(c + d) + (e + f)*(g + h)$$

Cette évaluation prendra 7 (ΔT) sur un système séquentiel, où ΔT est la durée d'exécution d'une instruction.
L'exécution d'opérations en parallèle nécessitera un temps qu'on peut calculer comme suit :

Durant le 1er ΔT : on effectue les 4 additions en même temps :

$(a + b)$, $(c + d)$, $(e + f)$, $(g + h)$

Avec V1=a+b V2 =c+d V3= e+f V4= g+h.

Durant le 2eme ΔT les deux multiplications : V1*V2 Et V3*V4.

Avec V1*V2 = V' et V3*V4 = V"

Durant le 3eme ΔT le calcul f : V = V' + V" .

Ainsi la fraction séquentielle **f** dans ce calcul est égale à **3/7 = 0.428**

ii. l'élimination de Gauss-Jordan est un algorithme qui renferme deux critères importants dans le problème du parallélisme : la densité de calcul – et la dépendance des données. Le fragment réalisant cette élimination dans une matrice A(N,N) est l'imbrication de trois boucles :

```
Pour k=1 à N
    Pour i=1 à N
        Fact ← A(i,k)
        Pour j=k à N+1
            A(i,j) ← A(i,j) – A(k,j)*fact
        Fpour
    Fpour
Fpour
```

La profondeur (*Depth*) ou la complexité de ce problème est dans l'ordre $O(\frac{N^3 + 3N^2}{2})$.

Nous pouvons remarquer qu'il n'y a pas de dépendances dans les deux boucles intérieures, donc peuvent être exécutées en parallèle, alors que la dépendance a lieu lors du passage de k d'une valeur à une autre (boucle extérieure).
Cette dépendance est mise en évidence en réécrivant le fragment avec l'utilisation des attribues SEQ et PAR qui désignent exécution séquentielle et parallèle respectivement :

```
SEQ k=1 à N
    PAR i=1 à N
        Fact ← A(i,k)
        PAR j=k à N+1
            A(i,j) ← A(i,j) – A(k,j)*fact
        Fp
    Fp
Fs
```

La fraction séquentielle est donc : $f \approx \frac{2N}{N^3 + 3N^2} \Rightarrow S_{Max} \approx \frac{N^2 + 3N}{2}$.

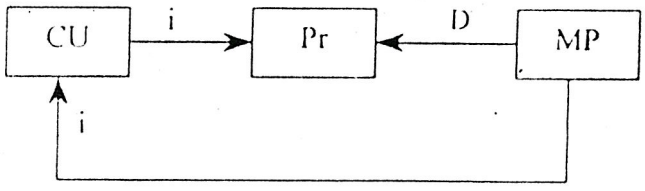
Et le problème est ramené à l'ordre O(N).

I.2 CLASSIFICATION DE FLYNN :

Pour caractériser les différents types de parallélisme, Les concepteurs d'ordinateurs ont proposé 4 classifications. La classifications de Flynn est la plus réputé.
Selon cette classification on peut distinguer les différentes architectures suivantes :

I.2.1 les systèmes à flot unique d'instruction et flot unique de données : (SISD)

A Cette classe appartient la machine séquentielle classique de Von Neumann qui ne renferme de parallélisme qu'à titre localisé.



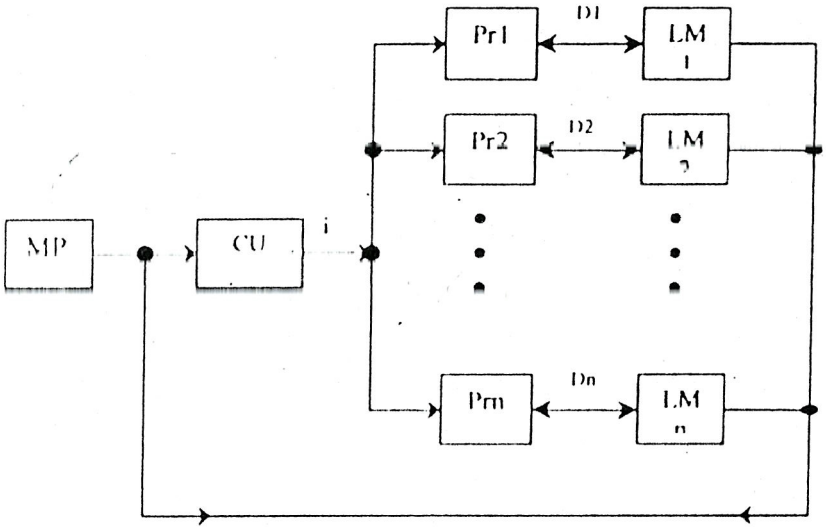
- Fig I.2.a : SISD

CU : unité de contrôle
Pr : processeur contenant l'ALU
MP: mémoire.

I.2.2 les systèmes à flot unique d'instruction flot multiple de données : **(SIMD)**

Plusieurs processeurs exécutent simultanément la même instruction d'un flot I, issu de l'UC.

Chaque instruction est exécutée sur un jeu différent de données Di présenté à chaque processeur Pri par une mémoire locale MLi. Le programme est rangé dans la mémoire principale MP et transmis à l'UC.

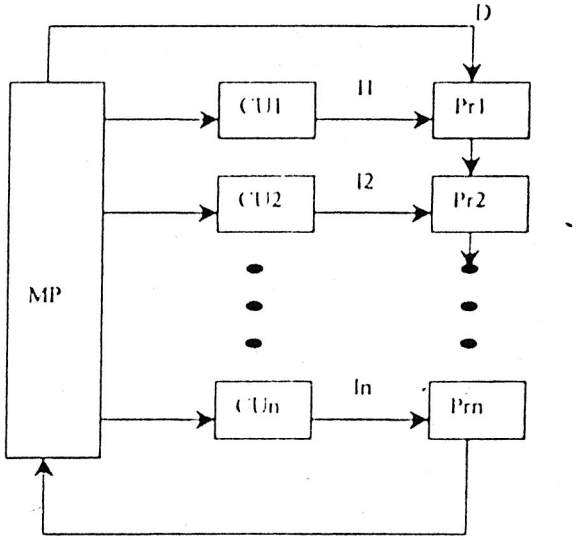


LM : mémoire locale

- Fig I.2.b : SIMD

I.2.3 les systèmes à flot multiples d'instruction flot unique de données : **(MISD)**

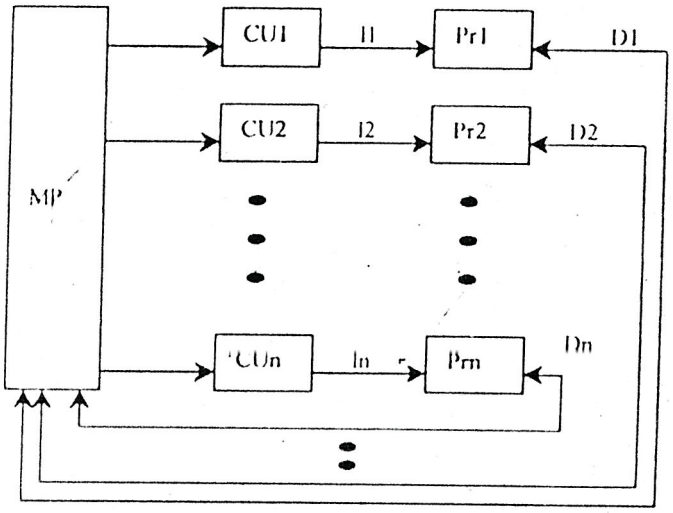
Dans un tel système une séquence de données est transmise à une séquence d'opérateurs, chacun d'eux est contrôlé par une UC séparée qui exécute une séquence d'instructions différentes. On remarque toutefois que la structure MISD n'a jamais été implantée réellement.



- Fig I.2.c : MISD

I.2.4 les systèmes à flot multiple d'instruction flot multiple de données : **(MIMD)**

Ce type de système est le plus répondu dans les conceptions multiprocesseurs. Il est caractérisé par le fait que plusieurs jeux d'instruction peuvent être exécutés sur un jeu de données différent simultanément. Ainsi on peut avoir sur un tel système un programme subdivisé en sous-programmes exécutés simultanément par plusieurs processeurs.



- Fig I.2.d : MIMD

I.3 Les multiprocesseurs(structures et interconnections) :

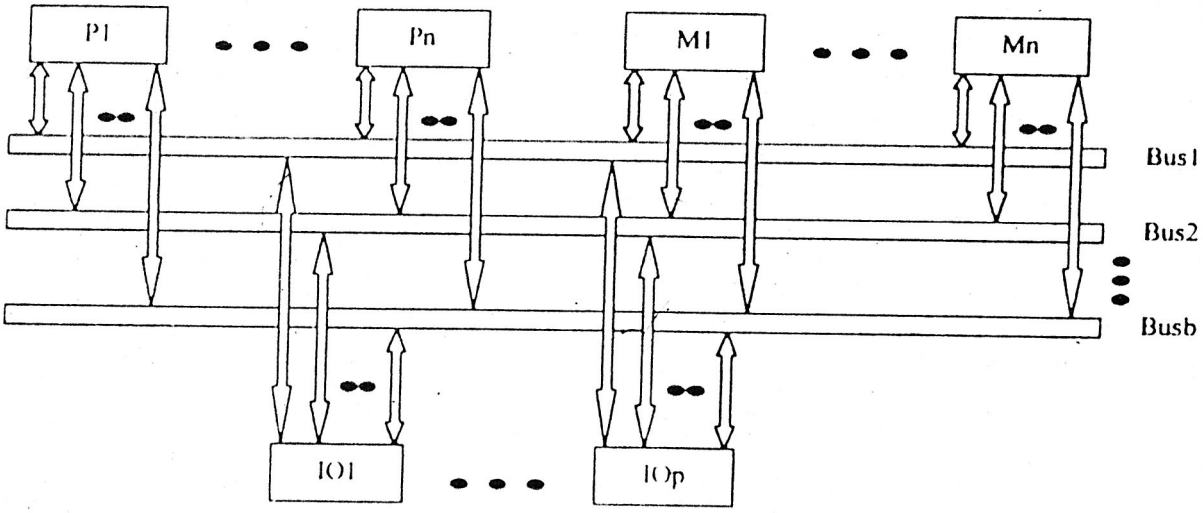
La classification de Flynn ne permet pas de faire des distinctions concrètes entre systèmes parallèles et en particulier dans les MIMD. En effet elle ne prend pas en compte de certaines particularités importantes qui sont : le couplage mémoires-processeurs ou processeurs-processeurs , la localisation mémoire (partagée - distribuée), et le synchronisme du système(synchrone - asynchrone).

En faisant compte du critère de la localisation mémoire par exemple, les MIMD selon la littérature sont classés en deux catégories : systèmes à ***mémoire distribuée*** et systèmes à ***mémoire partagée***. Les systèmes à mémoire partagée renferment une mémoire principale à laquelle peut accéder en totalité tous les processeurs du système. Alors que dans les systèmes à mémoire distribuée chaque processeur dispose d'une mémoire locale à laquelle il accède exclusivement seul. Ces deux catégories sont désignées parfois dans la littérature encore par les dénominations suivantes : ***multiprocesseurs***(systèmes à mémoire partagée) et ***multicomputers***(systèmes à mémoire distribuée). Elles sont considérées aussi comme un dual de systèmes ***fortement couplés*** et systèmes ***faiblement couplés***.

Pour caractériser encore mieux ces systèmes, on a eu recourt dans la littérature à prendre en considération le critère du mode d'interconnexion et de couplage. Les différentes catégories désignées à travers ce critère sont :

- Les systèmes orientés bus.
- Les systèmes hypercubes.
- Les systèmes à réseau de commutation.

- Les systèmes orientés bus sont bien populaires et considérés dans une certaine mesure comme les moins complexes de la part des constructeurs. Ils sont soit à bus unique soit à bus multiples. l'inconvénient majeur rencontré avec une configuration mono-bus est le goulot d'étranglement du au fait que les ressources (processeurs) ne peuvent pas accéder au bus en même temps. Avec une configuration multi-bus le problème d'étranglement est amoindri mais, au détriment d'une logique additionnelle complexe et coûteuse pour chaque module (voir FigI.3.a). Les systèmes commercialisés relevant de cette classe sont : ELXSI, Sequent et Encore.



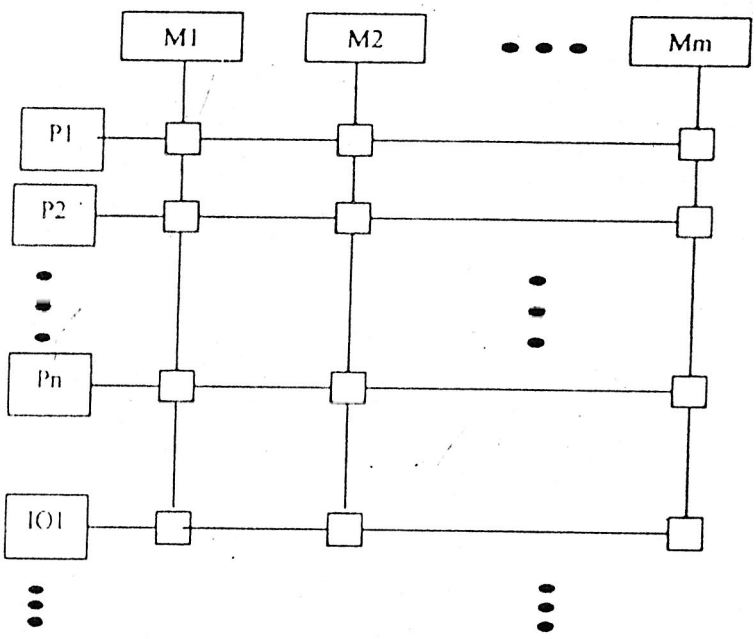
- Fig I.3.a : Système multibus

- Les multiprocesseurs hypercube sont caractérisés en particulier par une topologie en cube et l'utilisation d'un nombre puissance de 2 ($N=2^n$ processeurs). Ces systèmes sont conçus à base de cube imbriqué selon une logique appropriée(voir Fig I.3.b), la mémoire est distribuée, et chaque processeur dispose d'une mémoire locale. Le iPSC d'INTEL est un exemple d'hypercube.

Dimension	Nodes	Canaux	Topologie
0	1	0	
1	2	1	
2	4	4	
3	8	12	
4	16	32	

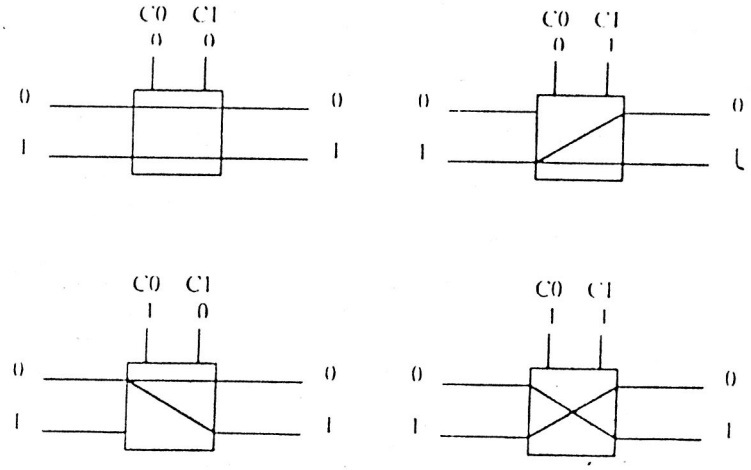
- Fig I.3.b : Topologie de l'hypercube

- Avec les systèmes à réseaux de commutation l'interconnexion est établie à l'aide de circuits d'aiguillage "CA" réalisant le multiplexage et l'arbitrage des lignes d'adresses, de données et de contrôle. Le Crossbar en est une variante assez répondue, sa configuration est donnée ci-dessous (Fig I.3.c).



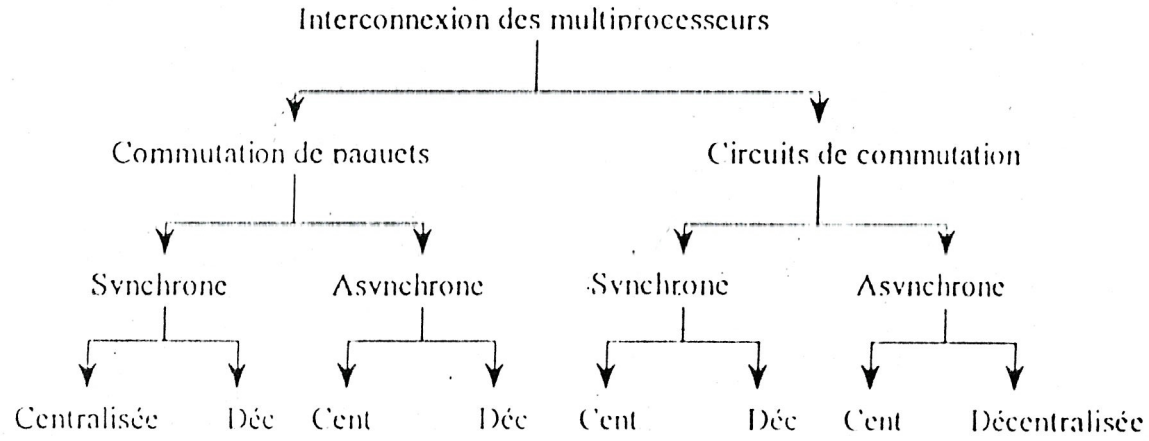
-Fig I.3.c : réseau à commutation Crossbar

Le réseau multicouche ou réseau cubique généralisé est une autre variante de réseaux à commutation moins coûteuse. Le composant élémentaire de ce réseau est une cellule d'échange à deux entrées et deux sorties avec deux lignes de contrôle (Fig I.3.d). Le couplage de N entrées et N sorties où N est un nombre puissance de deux $N=2^m$ est réalisé avec $(\frac{N}{2} * m)$ cellules organisées en m étages. Un système exemple conçu sur ce principe est le Butterfly de BBN.



-Fig I.3.d : Etats d'une cellule d'échange

Le mode de synchronisation de la communication est une caractéristique aussi importante des systèmes parallèles, on distingue deux modes de synchronisme :
le mode synchrone où le système est géré par une horloge centrale unique.
Et le mode asynchrone où les échanges s'opèrent principalement selon le principe d' un « hand shaking ».
La méthodologie de commutation « Switching methodology » et la stratégie de contrôle sont Aussi des paramètres qui peuvent parvenir à la caractérisation de la communication.
La commutation (des échanges) s'établit soit avec des circuits de commutation soit selon le mode commutation de paquets. La stratégie de contrôle, est soit centralisée ou décentralisée. Ces différentes caractéristiques sont groupées dans le schéma suivant :



- Fig I.3.e : caractéristiques de la communication dans les réseaux multiprocesseurs

I.4 Paramètres d'évaluation de performances :

Beaucoup de paramètres permettent l'évaluation des performance d'un réseau[23]. certains concernent uniquement les systèmes synchrones (Bande passante mémoire « *Bandwith* Bw», Probabilité d'acceptance « Probability of acceptance Pa ») ; d'autres concernent les systèmes asynchrones (le débit « Throughput Thr ») ; et en fin d'autres concernent les deux systèmes à la fois (utilisation processeur « Processor utilization » Pu).
Le throughput est définie comme étant la moyenne de nombre de paquets délivrés par le système par unité de temps.
L'utilisation processeur « Processor utilisation Pu » est défini comme le pourcentage de temps d'activité du processeur ($P_u = \frac{temps\ pendant\ lequel\ le\ processeur\ est\ actif}{temps\ total}$). Et le potentiel de calcul du réseau est donné comme le rapport : $\frac{P_u}{N}$ avec N est le nombre de processeurs utilisé dans le réseau.
Les paramètres déjà présentés servent en effet à l'évaluation des performances des différents modes d'interconnexion, alors que la performance globale du système est dépendante en premier lieu de la performances des processeurs utilisés.
Parmi les unités souvent adoptées pour la mesures des performances par les concepteurs de processeurs on cite : les MIPS, les MFLOPS. Ces mesures, cependant ne constituent pas toujours une approche quantitative tout à fait exacte et précise.

- Le MIPS : ou million d'instructions par seconde est donné par le rapport :

$$\frac{nombre\ d'instructins}{temps\ d'exécution * 10^6}$$

L'utilisation de cette mesure pour la comparaison des machines monoprocesseur doit tenir compte des aspects suivants:

° Le nombre de MIPS dépend du jeu d'instructions ; donc la comparaison de machines à jeux d'instructions différents est difficile.

° Ce paramètre dépend des programmes exécutés sur une même machine.

° Peut varier comme inversement proportionnel à la performance.

- le MFLOP : ou million d'instructions flottantes par seconde est donné par :

$$\frac{nombre\ d'instructins\ flottantes}{temps\ d'exécution * 10^6}$$

les problèmes rencontrés avec l'utilisation des MIPS se posent également au niveau desMFLOPs et en plus les MFLOPs ne sont pas une mesure utile pour tous les programmes.

- Les programmes de test de performance : une alternative pouvant être utilisée pour la prédiction des performances d'une machine est le recensement des temps d'exécution d'un même programme sur différentes machines. Quatre types de programme peuvent être utilisés à cette fin :

° Les programmes réels : tout programme communément utilisé et réputé peut être utilisé comme programme test, citons comme exemple le compilateur C, un traitement de texte, un logiciel CAO tel que SPICE... .

° Les noyaux : utilisés principalement pour prédire le comportement des machines en mode systèmes d'exploitation par exemple.

° Les programmes typiques : sont des programmes à code réduit(dizaine de lignes) Dont l'utilisateur connaît le résultat. Le crible d'érastosthenes et le tri en sont des exemples.

° Les programmes synthétiques d'évaluation des performances (Benchmarks): sont Des programmes conçus en vue de simuler un grand nombre de programmes réels. Leur code est créé synthétiquement de manière à ce qu'il corresponde à la moyenne des fréquences des opérations impliquées dans un bon nombre de programmes réels. Le Whetstone et le Dhrystone en sont des exemples.

I.5 Présentation de la stratégie adoptée pour le parallélisme :

Le parallélisme étudié ici concerne en particulier les applications de traitement d'images ou autres applications similaires caractérisées par un flux de données continues comme le traitement du signal. Par conséquent, on sera contraint à satisfaire deux principaux paramètres : Le débit (*THE THROUGHPUT*) – La latence (*THE LATENCY*).
On devra assurer au moins le même throughput de l'entrée vers la sortie, de sorte que le système en tout soit perçu comme étant un canal qui reçoit un flux de données avec une certaine cadence et émet un flux résultat avec au moins la même cadence. Cette contrainte s'impose notamment lorsqu'on a affaire à une visualisation, où on envoie des images vers un système de traitement qui doit à son tour envoyer les résultats vers un système d'affichage.

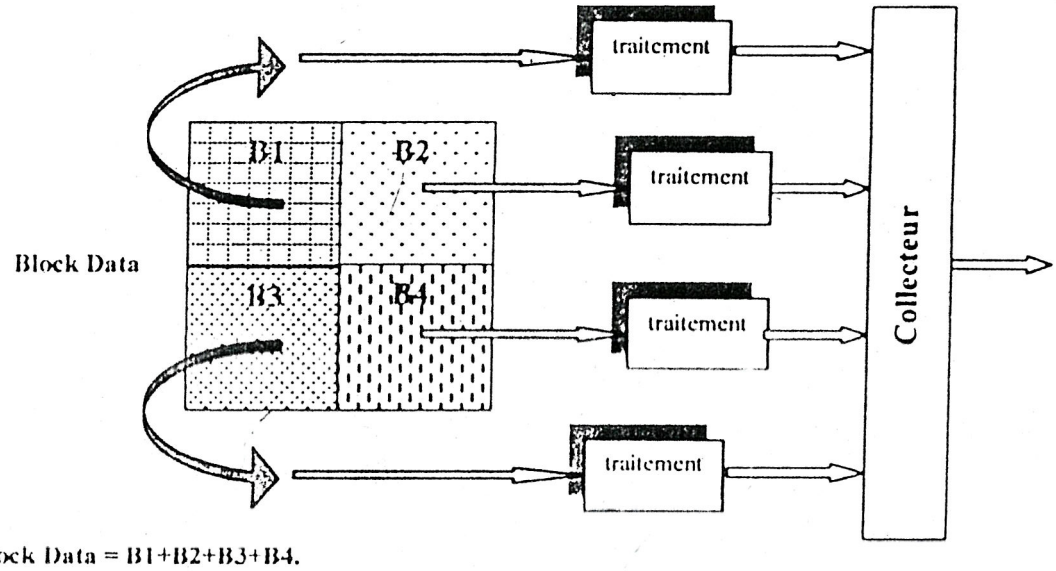
Dans d'autres situations on aura affaire à reproduire un flux en sortie avec non pas le même *throughput* d'entrée mais plutôt avec un *throughput* plus élevé. la décompression Vidéo en est un cas exemple. Par exemple, pour pouvoir transmettre la vidéo à travers un réseau (INTERNET), jusqu'à présent on a toujours recourt à une compression de manière à ce qu'on produit un débit « *Throughput* » qui s'adapte à la bande passante du canal de transit. A l'autre bout du canal on devra établir la décompression. le système de décompression, récupère ainsi le flux de la vidéo compressée à la cadence maximale tolérable par le canal (bande passante) et reproduit la vidéo à la cadence originale (celle de la caméra) qui est évidemment plus élevée.

La latence (*latency*) dans le contexte précédent (flux de données continues) c'est le temps que prend une donnée pour transiter dans le système de traitement. Ou d'une manière simple elle c'est le temps de réponse du système.
Elle est mesurée par l'inverse de l'accélération. Et c'est le paramètre qui est en liaisons directe avec le potentiel intrinsèque parallèle.

I.5.1 ARCHITECTURES PARALLELES:

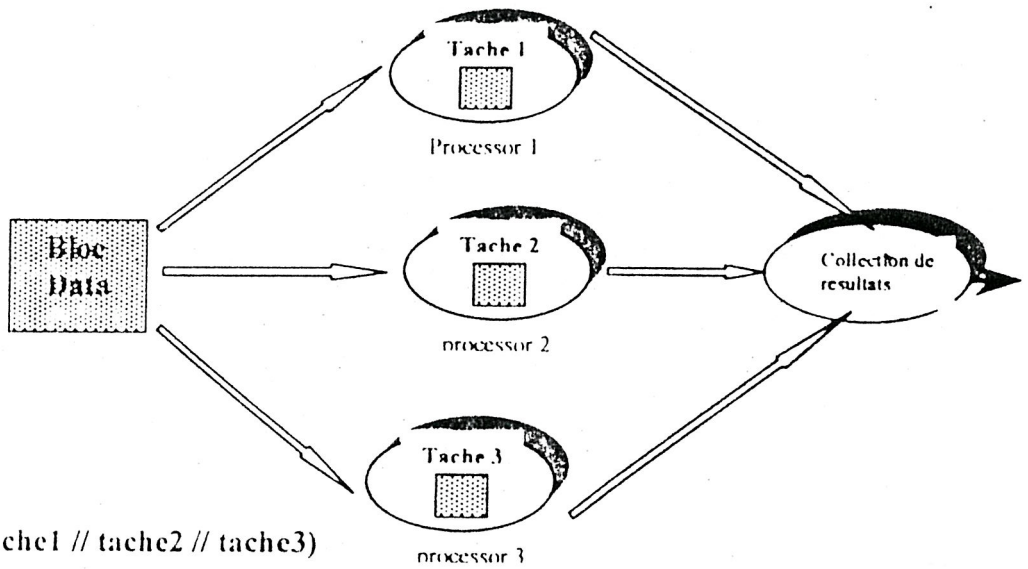
Le parallélisme revêt différentes formes :
a) le parallélisme spatial (de données): cette forme peut s'appliquer lorsqu'on a affaire à un ensemble de données qui doivent subir un même traitement. L'ensemble de données est fragmenté en partitions qui sont affectées chacune à une ressource puis traitées tous en concurrence.
Avec ce mode c'est la latence qui est améliorée proportionnellement au nombre de partitions.
Si le nombre de partitions est N, la latence théorique peut être ramenée à sa fraction Nième

$$latence_{améliorée} = \frac{latence}{N}$$



-Fig I.5.a : parallélisme spatial.

b) **le parallélisme algorithmique**: c'est dans cette forme qu'on peut vérifier le principe de la loi d'Amdahl. Ce parallélisme se base particulièrement sur le potentiel parallèle intrinsèque de l'application. On suspect les taches qui peuvent être exécutées en concurrence dans l'algorithme; on les affecte chacune à un processeur (ressource) qui opère sur l'intégrité des données. Comme dans le cas précédent c'est aussi la latence qui est améliorée, à la différence que dans ce cas il doit y avoir un étage qui collecte les résultats des taches.
La latence du système est égale alors à la latence de l'étage le plus lent augmentée de la latence de l'étage collecteur.
Sur la figure ci-dessous(Fig I.5.b), on a considéré une application P composée de trois taches ou processus concurrents : tache1 // tacche2 // tache3.



P = (tache1 // tache2 // tache3)

-Fig I.5.b : parallélisme algorithmique

c) Le pipeline: Cette forme est recommandée dans le cas où l'application est constituée d'un ensemble de taches séquentielles. Et où l'on a un flux de données continue. Les processeurs sont placés en cascade et les taches séquentielles leurs sont cédées dans l'ordre d'exécution tel que présenté sur la figure ci-dessous. La technique peut être assimilée à une chaîne d'assemblage où la pièce est manœuvrée dans un étage puis passée au suivant pour une autre manœuvre en livrant sa place à une autre similaire pour le même travail et le processus se poursuit jusqu'à la fin de la chaîne.
Cette méthode offre les avantages suivants :
- En considérant les *overheads* -communication inter-processeurs comme nuls et flux de données comme infini l'accélération est égale au nombre d'étages.
- Le débit (*throughput*) est amélioré proportionnellement au nombre d'étages.

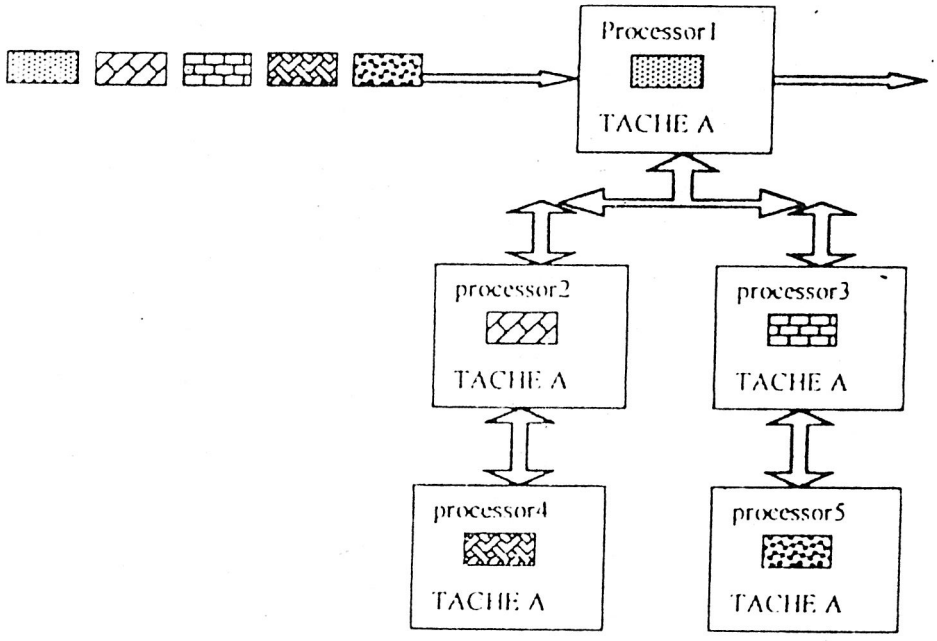
P = (tache1 + tache2 + tache3)

- Fig I.5.c : pipeline.

d) Multiplexage temporel :
En examinant cette technique on peut remarquer qu'elle améliore particulièrement le débit (*throughput*). Dans ce cas, les données sont distribuées à un ensemble de processeurs qui exécutent tous la même tache pour un bloc multiplexé de données. Le principe de cette approche pourra être plus visible lorsqu'on l'introduit plus loin dans le mode combiné.
Le débit (*throughput*) est amélioré ici proportionnellement au nombre de processeurs :

$$THROUGHPUT_{amélioré} = THROUGHPUT * N$$

Où N est le nombre de processeurs.

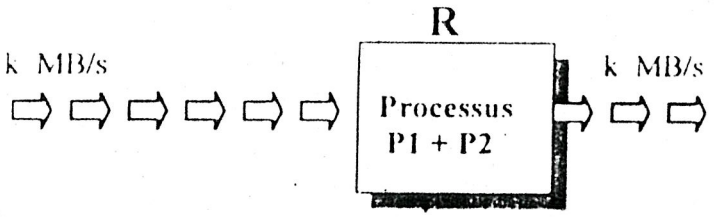


- Fig I.5.d : Multiplexage temporel.

En conclusion on peut dire que les deux premières méthodes contribuent particulièrement à l'amélioration de la latence alors que les deux dernières font augmenter particulièrement le débit.

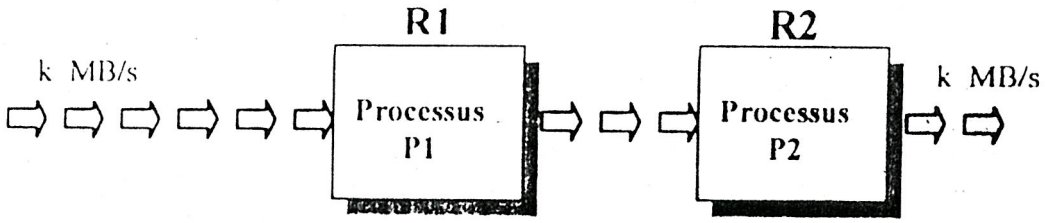
De la même manière une forme de parallélisme peut être aussi réalisée en combinant les approches présentées ci-dessus. Le premier schéma qu'on peut tiré est une structure articulée autour d'un pipeline. L'avantage du pipeline est montré à travers l'exemple suivant :
Considérant une application P composée d'une séquence de deux processus P1 et P2, constitués respectivement de **m1** et **m2** instructions. Et soit **k** (MByte/s) le débit du flux de données.

L'exécution du processus P sur un seul processeur R en gardant le débit de l'information à la valeur **k** nécessite une performance minimale de processeur égale à (m1+m2)*k MIPS(million instructions par seconde) en considérant l'instruction élémentaire opérant sur un seul byte. Ce ci ne peut être réalisé si le processeur R a une performance inférieure à cette valeur (m1+m2).k MIPS.



Performance de R >= (m1 + m2)*k MIPS

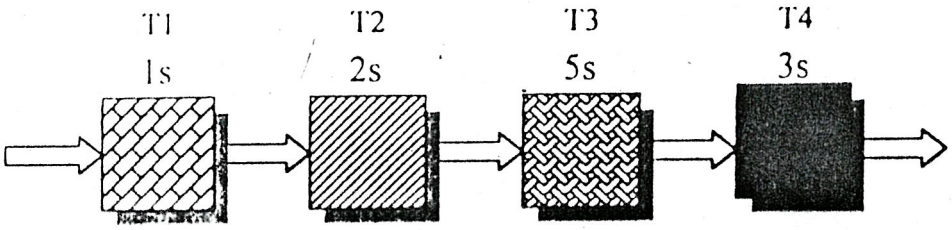
En utilisant un pipeline de processeurs R1, R2 qui exécutent dans l'ordre P1 puis P2 le débit est maintenu avec juste une performance de Max (m1,m2)*K MIPS pour les processeurs.



Performance de R >= Max(m1,m2) * k MIPS.

Le schéma dont on a parlé (schéma combinant différentes approches de parallélisme et articulé autour d'un pipeline) a été adopté dans **[16]** et était désigné sous le nom **PPF** (pipeline-processors-farme). Son avantage a été décrit par un exemple de calcul que nous reprenons dans ce qui suit :

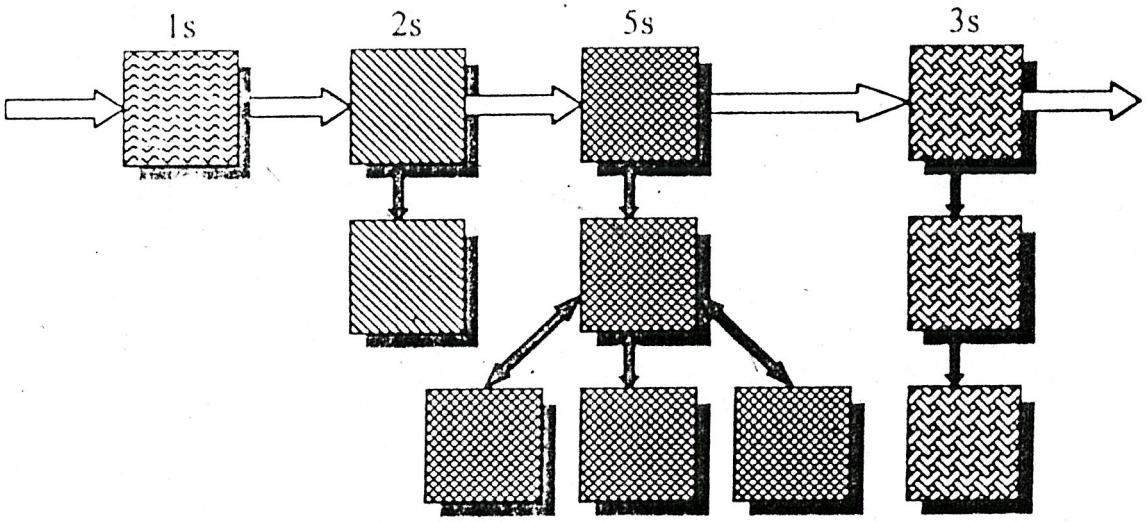
Considérons une application composée de 4 taches en séquence.
Les latence et débits requis avec les différentes structures sont les suivants :
1- Configuration en pipeline :



-Fig I.5.e : pipeline simple

La latence étant le temps que prend une donner pour transiter à travers le pipeline est égale à (3*5 + 3 = 18s). La latence = **18 s**. alors que le débit *throughput* = **0.2 tache/s**.

2- Configuration utilisant un multiplexage temporel :



- Fig I.5.f : PPF avec multiplexage temporel

La latence est de **11 s.** et Le débit est de **1 tache/s.**

3- Configuration utilisant une structure PPF :

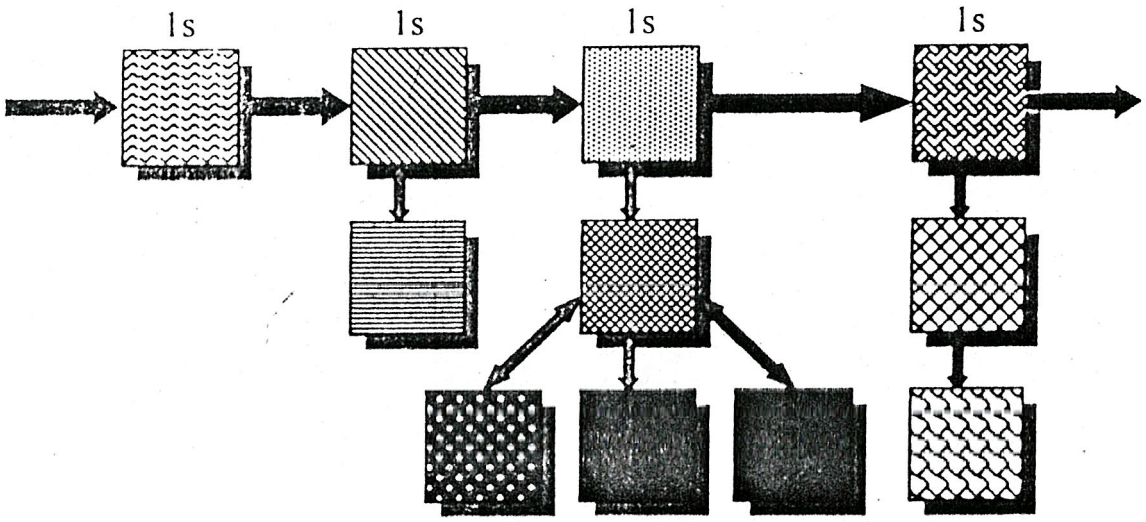


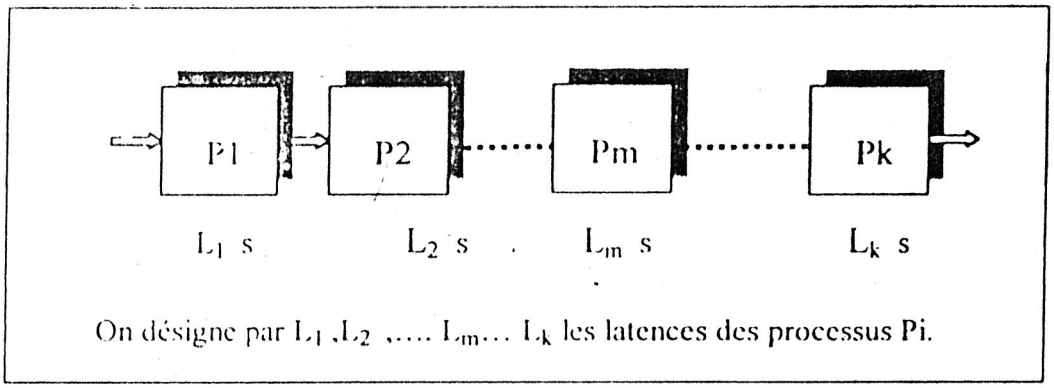
Fig I.5.g .PPF avec parallélisme spatial ou algorithmique .

La latence = **4 s.**
Le throughput = **1 tache/s.**

Ces calculs ont été faits sous les hypothèses suivantes :
- Les overheads de communications inter-processeurs sont considérés comme nuls.
- Le parallélisme intrinsèque est total au sein des étages 2,3,4 (c'est à dire que la fraction séquentielle f=0 dans chacun de ces étages).

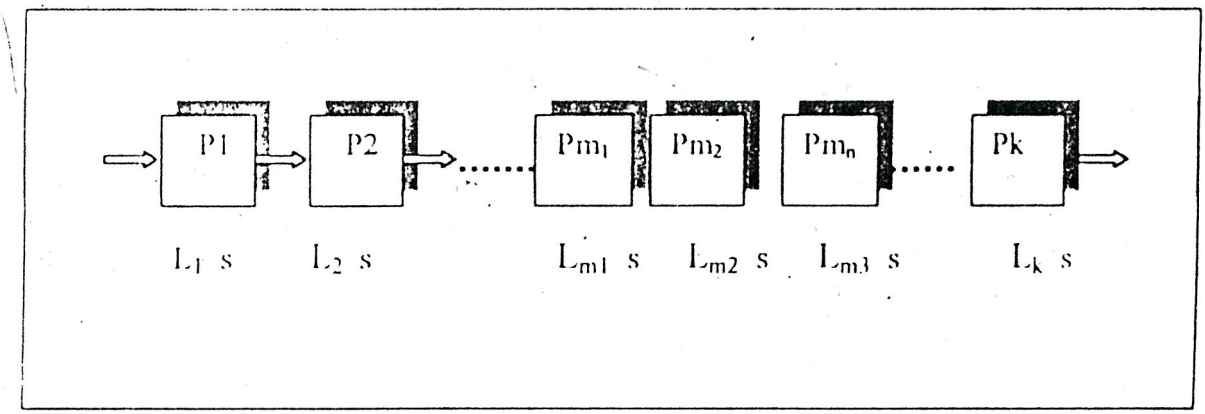
On voit que les meilleurs performances requises à la fois pour la latence et le débit *Throughput* sont réalisées au niveau de la troisième structure (Fig I.5.g) qui est un PPF à parallélisme algorithmique.
Théoriquement on peut continuer avec cette approche en affinant la granularité, autrement dit reprendre la démarche prise pour l'application P et l'appliquer au niveau de chaque tache composant P et ainsi de suite jusqu'à ce que la tache élémentaire exécutée sur un processeur soit réduite à une seule instruction. On arrive ainsi à une amélioration assez nette du débit, et de la latence également dans les limites tolérables par le potentiel intrinsèque parallèle(loi d'Amdahl).

La démarche qu'on suggère pour paralléliser une application à flot de données continues sera donc fondée sur l'approche PPF et l'approche d'affinement de granularité.
Les étapes du raffinement de la granularité peuvent être décrites comme suit :
Pour une application P composée d'un ensemble de taches séquentielles (P1+ P2 +...Pk),
On commence par un placement des processus : P1, P2, ...Pk en ordre d'occurrence dans un pipeline de nœuds (supposé des processeurs).



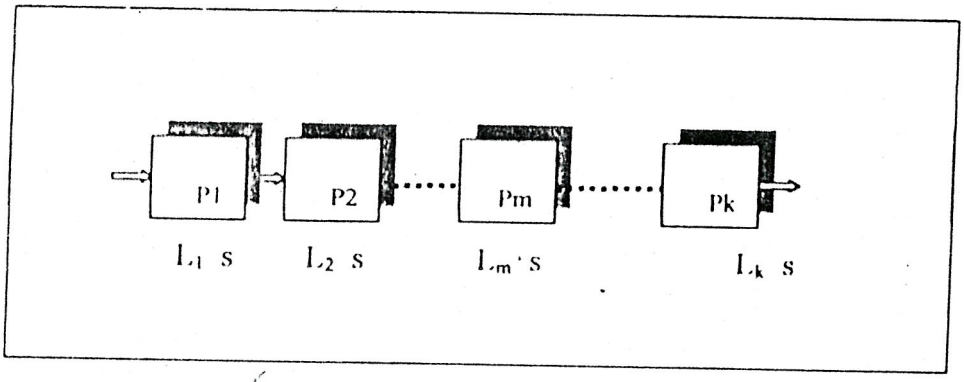
On désigne par $L_1, L_2, \ldots L_m \ldots L_k$ les latences des processus Pi.

Ensuit on repère l'étage le plus lent (c'est à dire celui dont la latence est la plus élevée) noté ici P_m. Si cet étage qui est supposé jusqu'ici un processeur ou élément processeur est en mesure d'assurer le débit fixé, alors le pipeline l'assure également, et le problème du débit est considéré comme réglé et on se consacre uniquement à l'amélioration de la latence. Si par contre ce n'est pas le cas, alors cet étage lui même devra être structuré en pipeline. Et Le processus P_m sera fragmenté en composantes(taches) séquentielles affectées à un nouveau pipeline. Le système aura l'allure suivante :



Cette opération va être répétée jusqu'à ce qu'on arrive à assurer le débit à travers le système.

Dans la prochaine étape c'est la latence qu'il faut améliorer. La latence à travers le pipeline (Fig ci-dessous) pourra être majorée par :



$$latence\ (P_{1\ldots k}) >= m * L_m + latence\ (P_{m+1\ldots k}) \quad \ldots\ldots\ldots\ldots\ldots\ldots(f)$$

Où latence($P_{1\ldots k}$) : désigne la latence du pipeline depuis le 1er étage jusqu'au dernier (le k ième). Et latence($P_{m+1\ldots k}$) : est la latence à travers les étages allant de m+1 à k. Avec P_m étant l'étage le plus lent.

La latence est une formule de récurrence donnée par la latence de l'étage le plus lent L_m multipliée par le nombre d'étages qui le précèdent (m) plus la latence du nouveau sous système formé des étages restants.

L'amélioration de la latence devra porter sur les étages lents dans la forme de récurrence, soit sur P_m , ou au niveau de l'étage le plus lent dans P(m+1..k) en utilisant un parallélisme algorithmique ou un parallélisme de données. Car si on parallélise un étage qui précède Pm la latence du système ne sera pas améliorée (d'après la formule (f)).

Il est à noter que le schéma de parallélisme présenté jusqu'ici est régi par trois approches essentielles :

- Premièrement on a essayé de définir une topologie en résonnant sur la logique algorithmique ce qui fait inscrire aussi le travail dans ce qui est appelé '' ADEQUATION ALGORITHME – ARCHITECTURE ''.
- Deuxièmement on a raisonner pour une implantation sur un système à transputers, donc l'utilisation d'un multiple de liens de communication et de mémoires locales.(Les transputers sont des produits VLSI dédiés à la conception de systèmes concurrents caractérisés en particulier par 4 liens séries de communication (voir §III).
- Troisièmement les oversheads ont été considérés comme nuls. Au fait leurs impacts ne sont pas tout à fait négligeables et seront examiner au § IV.

Chapitre II

Les traitements d'image et leurs parallélismes

Beaucoup de travaux ont traité le problème de la parallélisation des algorithmes de traitement d'images. On peut citer les travaux suivants :

- Parallélisation de la transformé de hough sur un réseau systolique[18].
- Parallélisation du filtre de canny-deriche avec l'outil *Syndex* [29].
- Parallèle décomposition en LU d'une matrice creuse[26].
- Nouvelle multiplication matrice-par-vecteur basée sur non recouvrement de decomposition de domaines[24].
- parallélisme total de transformation de matrice symétrique[25].

Dans ce chapitre nous présenterons quelques algorithmes de traitement d'images ' bas-niveau ' ; Nous rappellerons les principes fondamentaux de leur parallélisme. Et nous utiliserons l'approche graphique pour la description des méthodes de parallélisation.

Les traitements d'images développés et communément utilisés ces dernières décades peuvent être groupés en cinq (5) classes conformément à la figure II.1

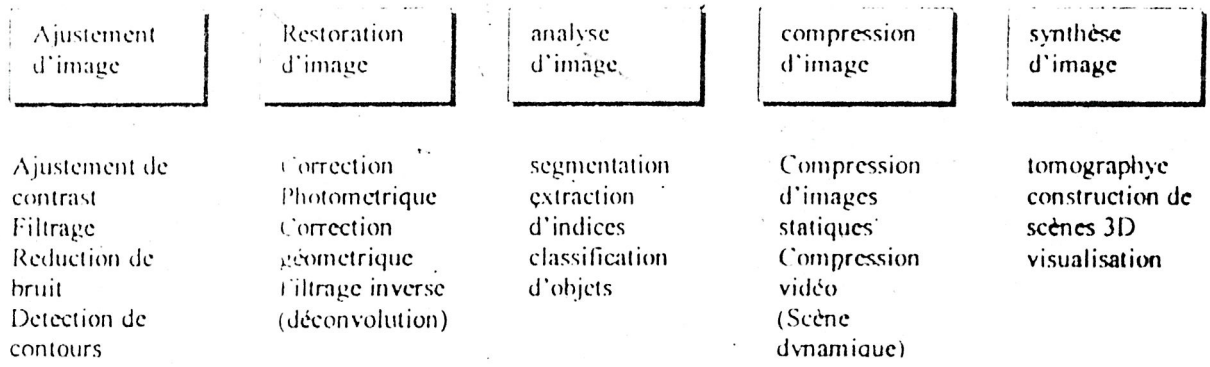


Fig II.1 : classement des traitements d'images

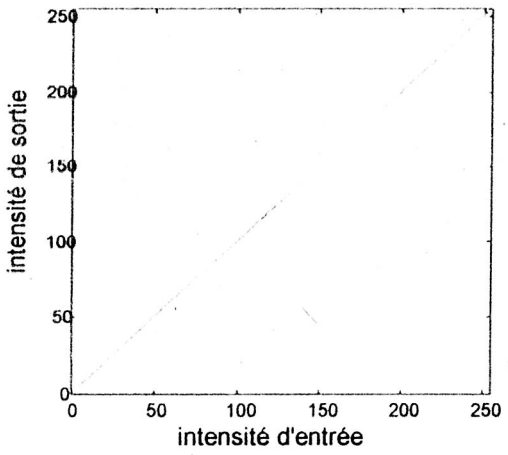
II.1 Restauration et rehaussement d'image : les traitements introduits ici sont souvent utilisés comme étape préliminaire ayant comme buts essentiels la réduction du bruit (filtrage), l'amélioration de l'apparence de l'image et la mise en évidence de certains détailles importants. Ces corrections sont soit à des fins subjectives ou à des fins objectives. Les corrections à des fins subjectives ont pour but de rendre plus plaisante l'image à la perception humaine. Cependant les corrections à des fins objectives préparent certains indices visuels dans l'image à un traitement ultérieur.

II.1.1 Ajustement du contraste : peut être défini par une transformation linéaire de la forme :

$$\mathbf{O}(x,y) = A * \mathbf{I}(x,y) + B .$$

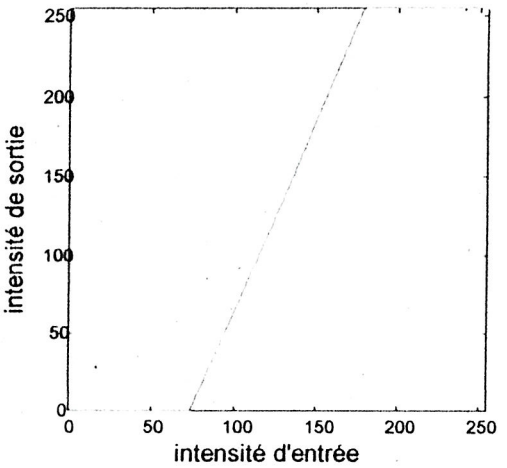
Avec $\mathbf{I}(x,y)$: pixel de coordonnées x,y de l'image d'entrée
$\mathbf{O}(x,y)$: pixel résultat de coordonnées x,y
A , B : coefficients d'ajustement.

Cette transformation a pour effet de rendre certaines zones ayant un niveau de gris donné selon le cas, plus sombres ou plus claires. Pour l'ajustement de contraste d'une image donnée on considère une courbe de contraste équilibrée :
(intensité de sortie = intensité d'entrée fig II.1.a), puis selon l'effet désiré on établie une courbe de contraste ajusté (fig II.1.b), Et en on déduit les coefficients d'ajustement correspondants.



- Fig II.1.a

- Image original -



- Fig II.1.b

- Image original à contraste ajusté -

L'algorithme d'ajustement peut s'écrire comme suite :

Pour toute pixel.in(i,j) :

Si (pixel.in(i,j) < Q)
Alors Pixel.out(i,j) = 0 ;
Sinon si pixel.in(i,j) > 255-Q
Alors / Pixel.out(i,j) = 255
Sinon
Pixel.out(i,j) = A*pixel.in(i,j) + B ;

Où A,B sont les coefficients d'ajustement déduits en fonction du quantum contraste Q.

II.1.2 la correction photométrique : l'acquisition d'images est le plus souvent sujette à une distorsion photométrique causée en premier lieu par les capteurs (*photos sensors*). La caractéristique du système d'acquisition qui, devait être linéaire, ne l'est pas en générale. La correction photométrique est alors utilisée pour compenser ces distorsions et linéariser la réponse du système. Un effet de compensation peut être aussi utilisé au niveau de l'affichage pour compenser les distorsions du tube cathodique (CRT).

La compensation peut être décrite par formule suivante :

$$O(i,j) = I(i,j) + \mathbf{index}[\, I(i,j)]$$

Où : I(i,j) est le pixel (i,j) de l'image d'entrée
O(i,j) est le pixel (i,j) correspondant dans l'image de sortie.
Index[ ]est le tableau de compensation.

On s'aperçoit que le traitement réalisant ces deux corrections se résume à une ou deux opérations ; Donc il n'y a pas lieu à la discussion du potentiel parallèle intrinsèque.

II.1.3 le filtrage spatial : Il a pour effet essentiel la réduction du bruit éventuel accompagnant l'image d'entrée. Certains filtres sont fiables mais impliquent une implémentation coûteuse. D'autres sont moins performants mais permettent une implémentation simple.
Nous citerons comme exemple de filtres simples, le filtre passe-bas et passe-haut, et nous discuterons leurs parallélisme.

° Les filtres passe-haut et passe-bas : Ils sont implantés couramment par un masque de convolution de dimensions variés (3,5,7...). L'expérience nous a appris qu'un bon compromis (fiabilité/taux de calcul) est réalisé avec des masques de dimension 3*3. Les masques communément utilisés pour le filtrage passe-bas et passe-haut sont les suivants :

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

. Masque pour filtre passe-bas .

-1	-1	-1
-1	9	-1
-1	-1	-1

. Masque pour filtre passe-haut .

Les résultats exemples obtenus avec ces deux masques sont donnés ci dessous :

- Image original -

- Image filtrée par passe-haut -

- Image avec bruit (Salt&pepper) -

- Image filtrée par passe-bas -

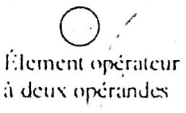
La convolution est donnée pour chaque pixel par :

$$O(i,j) = I(i,j)* M(1,1) + I(i-1,j-1)* M(0,0) + I(i-1,j)* M(0,1) + I(i-1,j+1)* M(0,2) + I(i,j-1)* M(1,0) + I(i,j+1)* M(1,2) + I(i+1,j-1)* M(2,0) + I(i+1,j)* M(2,1) + I(i+1,j+1)* M(2,2).$$

Avec **I**(i,j) représente le pixel (i,j) dans l'image d'entrée.
O(i,j) représente le pixel (i,j) dans l'image résultat.
M (comme matrice) est le masque de convolution.

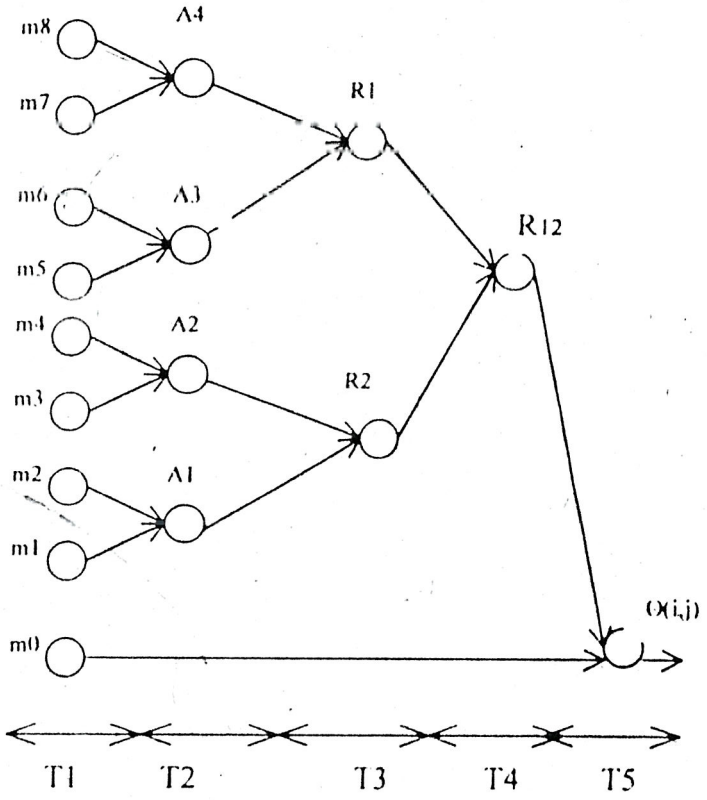
Le parallélisme des traitements que nous allons présenter est décris par un graphe de dépendance DG. Les éléments processeurs PEs (*processors elements*) sont présentés par des cercles et les acheminement des données sont présentés par des ares. L'abréviation PE a été inspirée de l'approche ***VLSI*** **[36]**; et de l'approche des systèmes distribués (réseaux systoliques et transputers **[18],[26]**).
Nous considérons en outre que le nombre de processeurs est toujours suffisant pour assurer les meilleurs performances.

la parallélisation de l'algorithme de filtrage en adoptant la description par graphe DG peut être décrite comme suit :



Le graphe traduit l'exécution des 9 premières multiplications en parallèle durant le premier cycle machine T1, les résultats sont de nouveau parallélisés au cycle T2 et ainsi de suite. On peut déjà voir que la fraction séquentielle est : $f = \frac{5}{17}$

Le déroulement des opérations se présente comme suit :



$$T1 : O(i,j) = \underbrace{I(i,j)*M(1,1)}_{m0} + \underbrace{I(i-1,j-1)*M(0,0)}_{m1} + \underbrace{I(i-1,j)*M(0,1)}_{m2} + \underbrace{I(i-1,j+1)*M(0,2)}_{m3}$$

$$+ \underbrace{I(i,j-1)*M(1,0)}_{m4} + \underbrace{I(i,j+1)*M(1,2)}_{m5} + \underbrace{I(i+1,j-1)*M(2,0)}_{m6} +$$

$$+ \underbrace{I(i+1,j)*M(2,1)}_{m7} + \underbrace{I(i+1,j+1)*M(2,2)}_{m8} .$$

$$T2 : O(i,j) = \underbrace{(m1+m2)}_{A1} + \underbrace{(m3+m4)}_{A2} + \underbrace{(m5+m6)}_{A3} + \underbrace{(m7+m8)}_{A4}$$

$$T3 : O(i,j) = \underbrace{A1+A2}_{R1} + \underbrace{A3+A4}_{R2}$$

T4 : O(i,j) = R1+R2
R12

T5 : O(i,j) = m0 + R12 .

En admettant que le flux de données est continu, et que les données sont codées sur un octet, nous pouvons déduire les résultats suivant :

Pour un débit (*throughput*) donné par : **Th = K** *MB/s*

L'exécution séquentielle du calcul précédent ne peut être réalisée en temps réel qu'avec un processeur dont la vitesse P est telle que : **P >= 17*K** *MIPS.*

Alors qu'avec le schéma déjà développé (utilisation de 17 éléments processeurs) l'exécution en temps réel est réalisée en utilisant des processeurs dont la vitesse est :

P = 1K MIPS.

En outre si la latence sur la version séquentielle est $L_{seq} = 1$.

L'accélération réalisée sera : S = 3.4 . et $L_{par} = \frac{1}{3.4}$. avec L_{par} est la latence pour l'exécution parallèle.

II.1.4 la détection de contours : la détection de contours est une opération nécessaire dans beaucoup d'applications de traitement d'images et en particulier pour la segmentation et l'extraction d'objets. Certains détecteurs s'implémentent comme des filtres en utilisant un masque de convolution(laplacien, Prewitt, Sobel). D'autres impliquent des transformations mathématiques et passent par des calculs complexes(Canny, Marr, Hueckel...).

Parmi les masques de convolution usuels on cite :

1. le filtre Laplacien2 : . Les masques de Sobel :

-1	-1	-1
-1	8	-1
-1	-1	-1

1	2	1
0	0	0
-1	-2	-1

direction horizontale

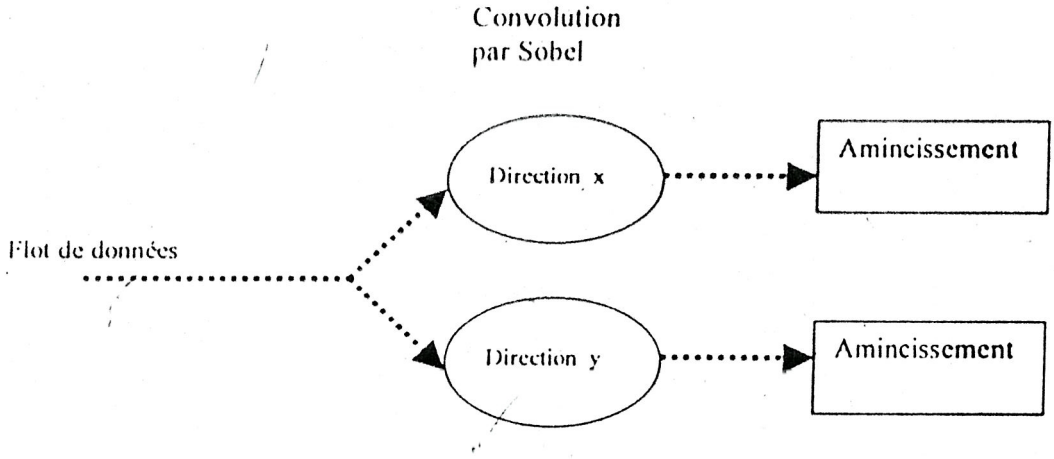
1	0	-1
2	0	-2
1	0	-1

Direction verticale

Les deux masques de Sobel réalisent la détection de contour chacun dans une direction. On peut par conséquent de manière concurrente, établir la convolution dans les deux directions.

La détection de contour est suivie couramment d'un cycle d'amincissement qui rend les résultats de la détection plus précis,

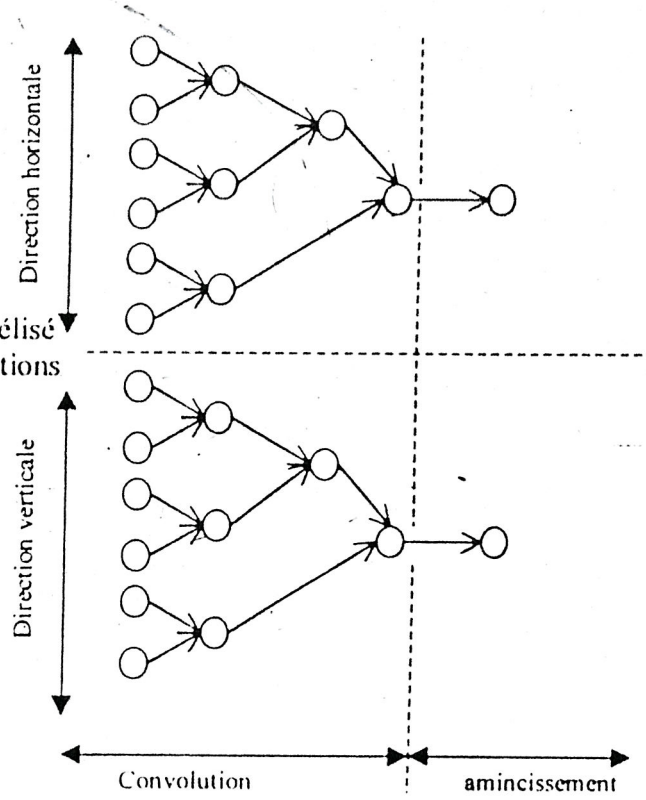
L'organigramme réalisant la détection par filtres de Sobel peut être décrit comme suit :



-Fig II.1.4 : . Convolution par les masques de Sobel (travaillant en parallèle), suivie d'une phase d'amincissement .

Et en reprenant l'approche utilisée en haut avec les filtres, Le graphe de flux de données DG pour l'exécution parallèle de la détection se présente comme suit:

Les masques de Sobel contiennent 6 éléments non nuls au lieu de 9 (ligne médian nulle dans le masque horizontale, colonne médiane nulle dans le masque vertical). Donc 6 noeuds sont utilisés pour les 6 multiplications dans chaque direction suivis des groupements d'additions séquenciels. L'amincissement est parallélisé de la même manière dans les deux directions et est supposé s'exécuter sur un seul nœud.

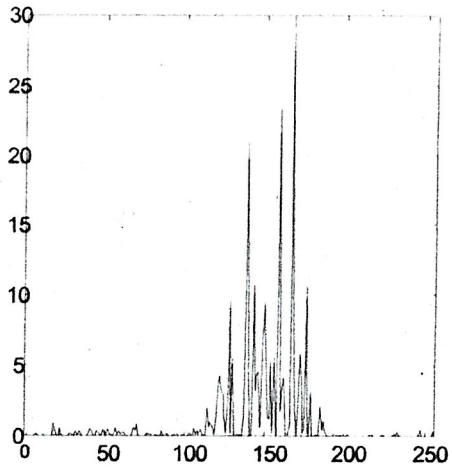


On déduit de même qu'avec ce modèle on réalise un Speed-up de $\frac{24}{4}$ et on arrive à assurer une exécution temps réel pour un flux de données de Throughput **Th** = K *MB/s* avec des éléments processeurs toujours de performance au moins égale à **K** *MIPS.*

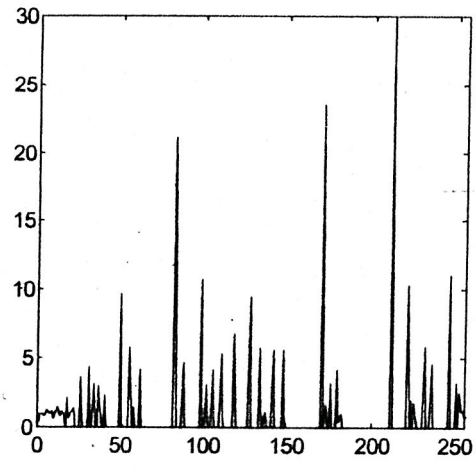
II.1.5 L'égalisation d'histogramme : on peut distinguer deux formes au moins d'égalisation d'histogramme. la prémière est dans un contexte subjectif, car elle améliore l'apparence de l'image à la perception humaine et consiste à rendre la distribution de l'histogramme uniforme (courbe plate).
La deuxième est dans un contexte objectif, et consiste à corriger l'image jusque ce que son histogramme ait une allure donnée.
Un exemple d'égalisation avec les histogrammes associés sont donnés ci-dessous :

- Image originale -

- Image à contraste égalisé -



- histogramme -



- histogramme égalisé -

-Fig II.1.5 -

L'égalisation d'histogramme d'une image passe par les étapes suivantes :
- le calcul de l'histogramme de l'image en question.
- Le calcul de la fonction cumule(histogramme).
- Et en fin la correction de l'image en fonction de la fonction intégrale (fonction cumule) déjà établie.

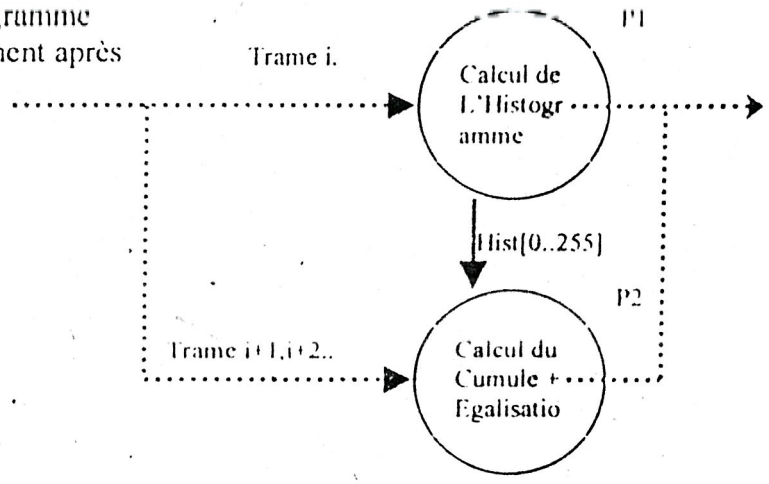
On discute dans la suite l'égalisation d'histogramme en temps réel :

Le calcule de l'histogramme de la première trame n'est conclu qu'à l'arrivé des dernières pixels de celle ci ce qui induit que l'histogramme n'est calculé qu'après au moins $N^2 * \frac{1}{K} \mu s$. Où la taille de l'image est (N*N). Et les pixels sont cadencé à **K** *MB/s.*

Ainsi l'égalisation n'ira pas s'opérer sur la trame dont on calculé l'histogramme mais plutôt sur une ou plus des trames suivantes. Cette manœuvre se prête à être valide si on admet qu'il n'y a pas de variations importantes et brusques depuis la première trame aux suivantes.

Le schéma proposé met en œuvre deux éléments processeurs :

Dans P1 se fait le calcul de l'histogramme Hist[0..255] de la trame i ,évidemment après que le dernier pixel y appartenant soit arrivé. L'histogramme calculé est retransmis à P2 où le cumule est calculé et utilisé immédiatement pour l'égalisation des trames suivantes i+1, i+2 ..... (nombre optionnel).



- Fig II.1.6 : schéma d'égalisation d'histogramme temps réel avec deux PEs

II.2 la compression d'images : la compression d'images est une opération qui fait ramener la taille des données représentant l'image à une taille plus réduite et plus compacte sans autant altérer la qualité de l'image. Ce ci est réalisé en faisant garder juste l'information utile de l'image et en écartant les informations redondantes éventuelles. La compression est bénéfique d'une manière immédiate pour deux utilisations essentielles : Le stockage et le transport (la transmission de l'information).

Le mot compression adresse en effet un dual (compression- décompression). Si ce dual se réfère à une technique unique car en principe l'un d'eux est l'inverse de l'autre, le plus souvent, ils s'opèrent avec des délais différants.

Les techniques utilisés pour la compression sont réparties en deux classes :
- Les méthodes de compression sans pertes.
- Les méthodes de compression avec pertes.

II.2.1 Les méthodes de compression sans pertes : On désigne par méthodes de compression sans pertes les méthodes qui à la décompression reproduisent l'original exact du document compressé. Elles sont utilisées principalement pour la compression des documents textes. Les techniques utilisées ici sont au fait une sorte de codage. Parmi les plus réputées on cite :
Le codage entropique RLC(Run Length Coding), le codage de HUFFMAN, le codage DPCM*(differential pulse code modulation)*, et le codage par principe du dictionnaire (Lempel-Ziv-Welch).
Le parallélisme de cette classe de méthodes n'est pas discuté car on peut s'apercevoir que le potentiel parallèle intrinsèque n'est pas aussi significatif.
II.2.2 Les méthodes de compression avec pertes : Sont les méthodes qui à la décompression ne reproduisent pas l'origine exact de la source compressée mais plutôt reproduise l'information utile y tenant. Elles sont utilisées en particulier pour la compression des images (de toute catégories simples ou séquences dynamiques), et pour la compression du son également. Parmi les techniques usuelles on cite :
Le codage par troncature – codage prédictive – codage par blocs - la compression par transformations.
II.2.2.1 *Le codage par troncature :* la compression est établie ici en effectuons une troncature franche d'un quantum de données représentons l'image. Et on envisage deux sortes de troncatures :
a)la troncature spatiale qui consiste à faire une sorte de réduction de la taille de l'image(un *Resize).* si l'image originale par exemple tient sur N X M pixels , la réduction de sa taille en la faisant ramener à une résolution spatiale $N'XM'$, établie un rapport de compression de $\frac{NXM}{N'XM'}$. La reconstruction de l'image source s'établit soit par interpolation soit plus simplement par agrandissement.
b)la troncature en résolution de luminance ou encore troncature en résolution de codage consiste à rétablir le codage des pixels sur un champs codes réduit. Une image originale qui tenait sur une résolution de 8bits par pixel (cas des images à échelle de gris courantes), sous sa forme compressée , devra tenir sur une résolution de longueur inférieur (6 ou 5bits,...par exemple).
Le taux de compression requis est alors : $\frac{longueur\ du\ champs\ code\ original}{longueur\ du\ champs\ code\ établi}$

II.2.2.2 *Le codage par blocs* : cette méthode de codage ressemble à la compression à base de dictionnaire(LZW). Toutefois, au lieu de considérer des chaînes de symboles comme étant des entités codées sur le dictionnaire, on considère ici des blocs de pixels de tailles variées (4*4 , 8*8, ...).
Beaucoup de recherches et de travaux ont été menées dans cet axe et se sont concentrés principalement autour de deux modèles connus par : la quantification vectorielle (*vector quantisation*)– et les fractales (*fractal compression*).
Le parallélisme de ce type de compression n'est pas analysé ici, car les tendances sont diversifiées.

II.2.2.3 *La compression par transformés :* ce mode de compression s'opère en appliquant une transformation mathématique qui ramène l'image d'une représentation spatiale en une représentation dans le domaine fréquenciel. La représentation fréquencielle se caractérise par le fait que la grande partie de l'information utile est concentrée autour des composantes spectrales les plus basses, au moment où les harmoniques les plus éloignées constituent les informations redondantes dans l'image et les détailles transitoires. Ainsi en affectant une

résolution appropriée à cette dernière gamme du spectre, ou encore en faisant une troncature franche(mise à zéro), l'image peut être codée sous une forme plus compacte. Les transformés éligibles à cette utilisation son celles héritées du domaine de traitement du signal : la FFT et la DCT. Toutefois, pour sa fiabilité, c'est la DCT qui était choisie et adoptée dans les standards de compression(JPEG, MPEG).

Le principe de la compression JPEG et qui est repris en partie dans MPEG est le suivant: L'image est fragmentée en blocs de pixels de dimensions 8x8, auxquels est opérée la DCT, puis dans chaque bloc transformé on procède aux éliminations(plutôt des mises à zéro) des éléments les plus éloignés de l'entrée du bloc (hautes fréquences) et qui sont souvent des petites valeurs. Et on passe enfin à l'étape de codage où l'on utilise le codage DPCM suivi du codage de HUFFMAN.

On peut se contenter à travers le modèle JPEG de deux arguments :
- le taux le plus important de calcul est consacré à la transformation DCT.
- La phase où le potentiel parallèle s'avère plus important est la phase de la transformation DCT également.

Nous essayons donc, dans ce qui suit, de donner un modèle de parallélisme de la DCT :

L'expression de la DCT et son inverse sont données respectivement par :

$$D(m,n) = \alpha(m)\alpha(n)\sum_{i=0}^{N-1}\sum_{k=0}^{N-1} d(i,k)\cos\left[\frac{\pi(2i+1)m}{2N}\right]\cos\left[\frac{\pi(2k+1)n}{2N}\right]$$

$$d(i,k) = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} \alpha(m)\alpha(n)D(m,n)\cos\left[\frac{\pi(2i+1)m}{2N}\right]\cos\left[\frac{\pi(2k+1)n}{2N}\right]$$

$$avec \quad \alpha(0) = \frac{1}{\sqrt{N}} \qquad et \quad \alpha(m) = \sqrt{\frac{2}{N}} \qquad 1 \le m \le N$$

On peut bien vérifier que le calcul des formules précédentes se rapporte plus simplement à des multiplications de matrices. En considérant des blocs de 8*8, la DCT sera évaluée comme suit :
[dct] = [bloc]*[cosin]*[bloc].
Et de même la IDCT :
[IDCT] = [DCT]*[cosin]*[DCT].

Nous commençons dans un premier temps par la multiplication de deux matrices. Nous essayons ensuite de définir le potentiel parallèle intrinsèque, d'en déduire les accélérations possibles et d'établir un modèle théorique pour le parallélisme. Puis dans un second tour nous extrapolons la manœuvre à la DCT.
La profondeur ***(depth)*** du problème de la multiplication de deux matrices de même dimensions N*N est de $O((2.N-1) * N^2)$. On conçoit qu'en résonnant sur la dépendance des variables (éléments des matrices), la profondeur du problème peut se ramener à l'évaluation d'un seul élément du résultat produit soit O(2.N-1). Car le calcul de chaque élément de la matrice produit est indépendant des autres (N multiplications + (N-1) additions).

On voit qu'on peut réaliser jusqu'ici un speed-up de $\frac{O((2.N-1)*N^2)}{O(2.N-1)} = N^2$.

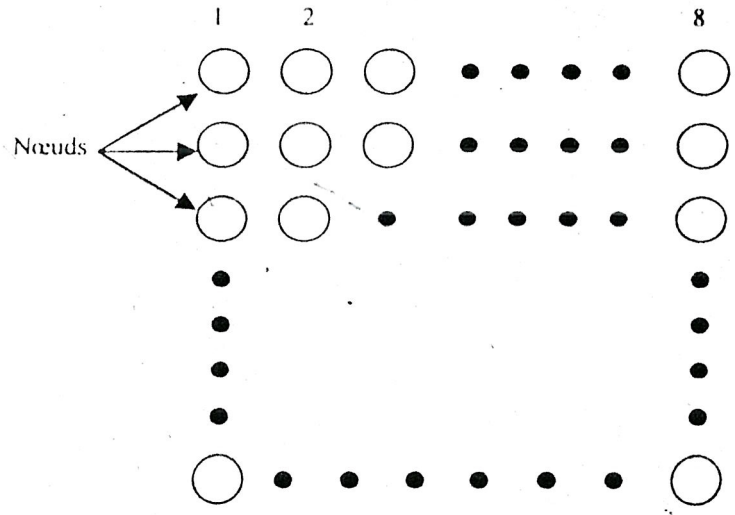
Avec l'approche graphique établie en §II.1.3, le calcul d'un élément peut se ramener également à $O(\log_2(N)+1)$; ce qui donne un speed-up de $\frac{O(2.N-1)}{O(\log N+1)}$ (pour l'évaluation d'un seul élément).

Et le Speed-up total qu'on pourra atteindre est donc :

$$S = \frac{O((2.N-1)*N^2)}{O(\log_2.N+1)}$$

Dans le cas de la DCT les matrices sont de dimensions N=8. Le Speed-up de ($N^2 = 64$) pour la multiplication de deux matrices est réalisé avec une grille de 64 nœuds, où on suppose effectuer le calcul de chaque élément de la matrice résultat dans un nœud séparé.

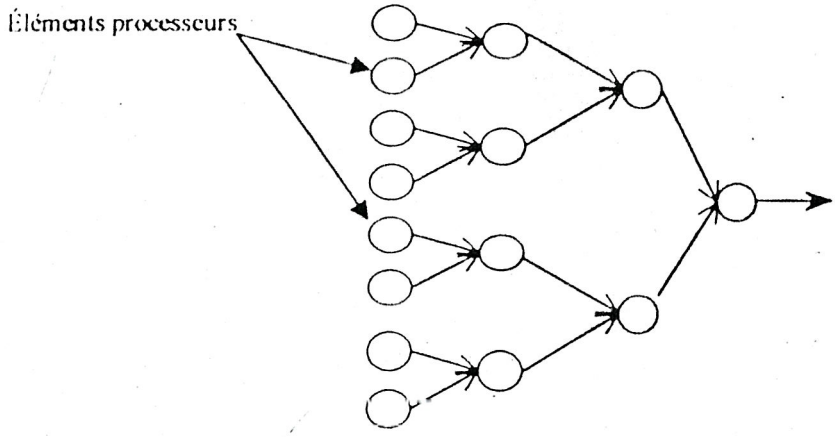
La machine Systola 1024 par exemple (réseaux systoliques) [18] est constituée d'une grille de 4x4 blocs de processeurs. Chaque bloc est constitué de 64 processeurs arrangés en grille de 8x8. Cette machine peut constituer un support adéquat à une telle décomposition.



-Fig II.2.a : Grille de 64 nœuds

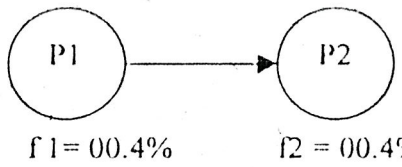
Ensuite, le calcul de chaque élément est lui même parallélisé selon la structure donnée sur figure II.2.b :

Ceci induit donc un speed-up de $(\frac{O(2.N-1)}{O(\log N+1)} = \frac{15}{4})$ au sein d'un seul nœud, et le speedup total sera $S = \frac{O(2.N-1)}{O(\log N+1)}.N^2 = \frac{15}{4}*64 = 240$.



-Fig II.2.b : Organisation des EPs au sein d'un nœud

La DCT est une séquence de deux processus P1 et P2. chacun est un produit matriciel d'une fraction séquentielle $f = \frac{1}{240} = 00.4\ \%$:



Le Speedup Maximal qu'on pourra estimer pour la DCT est donc: $S = \frac{2}{f1+f2} = \frac{1}{f} = 240.$

II.3 l'analyse d'images : l'analyse d'image comprend l'ensemble des traitements opérant soit une partition de régions dans l'image(segmentation) , soit une extraction d'indices visuels (caractérisation), soit une classification des éléments déjà caractérisées(classification). Elle ne produit pas, comme c'est le cas des classes de traitements précédentes un résultat pictoriel (***pictorial***),mais plutôt fournit des décisions.

II.3.1 la segmentation d'image : est l'opération de discrimination des éléments composant une image donnée. Son résultat est une représentation adéquate de ces éléments dans un arrière plan(***background***), permettant de faire des localisations, des mesures ou d'une manière générale des caractérisations. Parmi les techniques utilisées pour la segmentation, on peut citer la morphologie mathématique.
On distingue deux sortes de traitements morphologiques :
- morphologie binaire.
- morphologie à échelle de gris.

II.3.1.a. *la morphologie binaire* : la morphologie binaire a une grande similarité avec le filtrage spatial par convolution. L'image est scrutée pixel par pixel avec un masque mettant en fonction chaque pixel avec les pixels adjacents (*neighbors*). Dans ce cas les images traitées sont des images binaires et par conséquent l'image résultat est également binaire.
Les opérations morphologiques sont nombreuses. Nous nous limitons dans ce qui suit à deux opérations fondamentales qui sont (l'érosion – la dilatation) ainsi qu'à quelques opérateurs conçus sur la base de ces deux dernières.

- *L'érosion* : consiste à la suppression des pixels formant le périmètre des objets figurant dans l'image. Elle s'implémente, soit en utilisant un élément structurant, soit avec la technique des (LUT) *look-up-table*. L'élément structurant est défini par un masque de dimension (3*3), le plus souvent chargé d'une manière appropriée de valeurs binaires. Un pixel résultat est évalué en mettant en fonction les pixels adjacents et l'élément structurant avec des opérateurs logiques. Soit M un masque donné par la matrice suivante:

M

M1	M2	M3
M4	M5	M6
M7	M8	M9

L'érosion au niveau d'un pixel I(x,y) sera donnée par :

Si I(x,y)=M5 AND
I(x-1,y-1)=M1 AND I(x-1,y)=M2 AND
I(x-1,y+1)=M3 AND I(x,y-1)=M4 AND
I(x,y+1)=M6 AND I(x+1,y-1)=M7 AND
I(x+1,y)=M8 AND I(x+1,y+1)=M9
Alors O(x,y) = TRUE
Sinon O(x,y) = FALSE. Où I(x,y), O(x,y) sont le pixel d'entrée et le pixel résultat respectivement.

Dans la technique des LUT on considère plutôt un tableau de valeurs binaires prédéfinis à partir d'un élément structurant donné, et de longueur :

$$l = 2^{N^2}.$$

où N est la dimension de l'élément structurant.
Si N=3, comme c'est le cas le plus courant, $l = 2^9 = 512$.

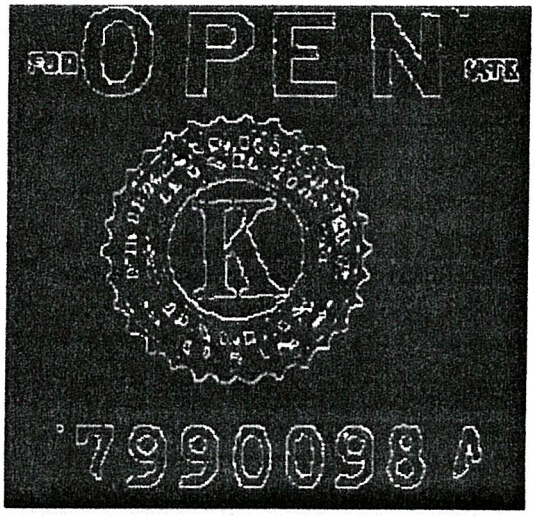
L'évaluation du pixel résultat d'érosion sera donné par l'élément indexé dans ce tableau(LUT) par les pixels adjacents(neighbors) mis en équation.

- *La dilatation* : est l'opération inverse de l'érosion et consiste en une extension de la taille des objets figurant dans l'image. les périmètres résultats sont les périmètres qui entouraient immédiatement les périmètres d'objets avant dilatation.
Au même titre que l'érosion, la dilatation s'implémente soit en utilisant un élément structurant soit avec des LUT.
En utilisant un élément structurant les pixels de l'image résultat seront donnés par :

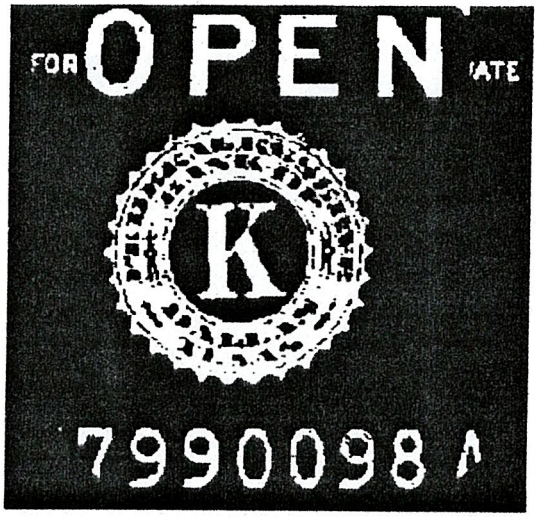
Si I(x,y)=M5 AND
I(x-1,y-1)=M1 AND I(x-1,y)=M2 AND
I(x-1,y+1)=M3 AND I(x,y-1)=M4 AND
I(x,y+1)=M6 AND I(x+1,y-1)=M7 AND
I(x+1,y)=M8 AND I(x+1,y+1)=M9
Alors O(x,y) = TRUE

Sinon O(x,y) = FALSE.

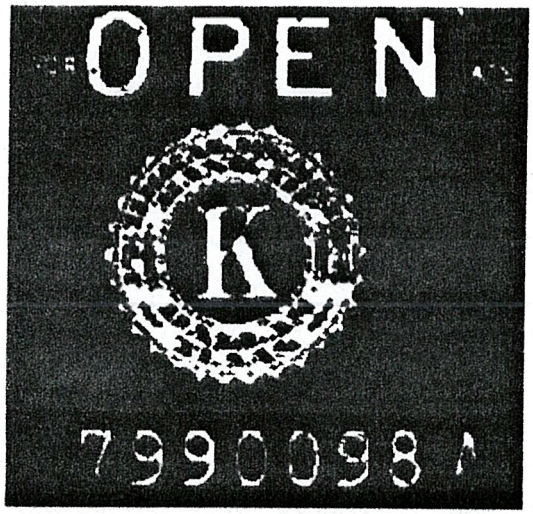
Les figures ci-dessous sont des illustrations d'érosion, de dilatation et (d'*outlining*) :



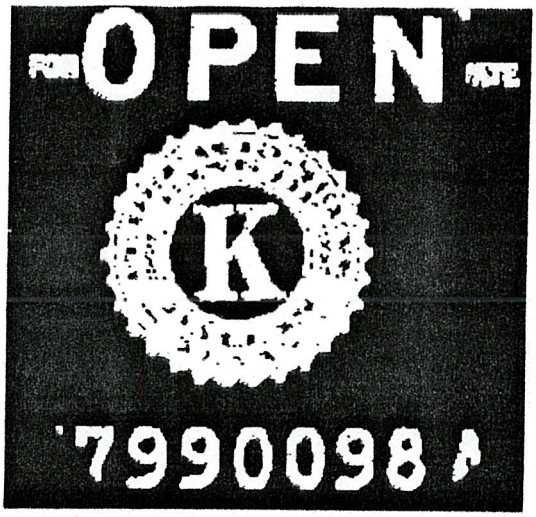
-détection de contour par morphologie-



- Image binarisée -

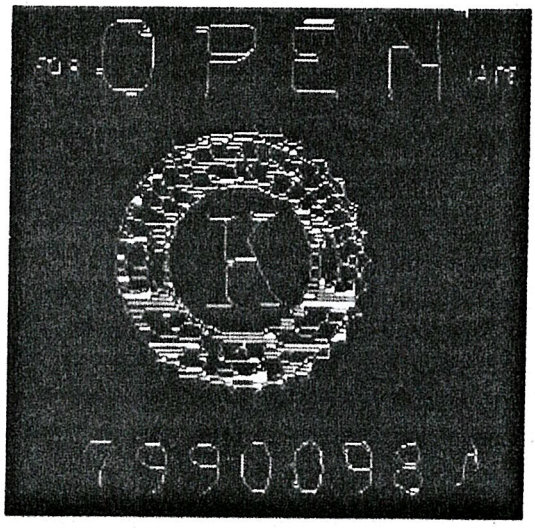


- Image érodée -



- Image dilatée -

- Image squelètte -



- *L'ouverture et la fermeture* : sont des opérations dérivées de l'érosion et de la dilatation. L'ouverture est une érosion suivie d'une dilatation. Son effet consiste en la suppression des points ou des taches isolés en maintenant les objets fictifs à leurs tailles et formes originales. Elle peut être réalisée aussi en effectuant un multiple d'érosions suivis du même multiple de dilatations. La fermeture quand à elle est une dilatation suivie d'une érosion. Son effet consiste principalement en la suppression des discontinuités éventuelles dans les corps d'objets.

- *La détection de contour par morphologie* ***(outlining)***: s'opère en faisant soustraire l'érodée d'une image de la même image dilatée.

- La skelétisation : s'opère en effectuant une séquence d'érosions reprises dans plusieurs directions jusqu'à ce que les objets soient réduits à leurs squelettes (soit uniquement des lignes représentants la forme d'objet). son utilité est de donner une représentation compacte renseignant en particulier sur la forme des objets.

II.3.1.b. ***la morphologie à échelle de gris :*** les opérations discutées ici sont une version étendue de ceux utilisées dans la morphologie binaire. L'image d'entrée et l'image résultat ne sont pas binaires mais des images en niveau de gris. A l'instar de la morphologie binaire, c'est un masque de dimension (3*3 le plus souvent) qui est utilisé pour produire le résultat. Les valeurs du masque sont dans le rang [-255 ...0] pour l'érosion et dans le rang [0... 255] pour la dilatation.
Chaque pixel résultat est évalué comme le Minimum(érosion) ou le Maximum(dilatation) des valeurs de pixels adjacents augmentées des valeurs de masque correspondantes. Ou donné formellement encore par :

$$O(x,y) = \text{Min / Max } \{ I(x,y) + M5,\ I(x-1,y-1) + M1,\ I(x-1,y) + M2,\ I(x-1,y+1) + M3,\ I(x,y-1) + M4,\ I(x,y+1) + M6,\ I(x+1,y-1) + M7,\ I(x+1,y) + M8,\ I(x+1,y+1) + M9 \}$$

ou M est un masque de valeurs M_i : [0.. 255]

M1	M2	M3
M4	M5	M6
M7	M8	M9

- L'érosion et la dilatation : sont à la base aussi de beaucoup d'opérateurs morphologiques relevant à cette classe de traitement et sont implémentés le plus souvent en utilisant un masque de valeurs nuls. Soit :

M:

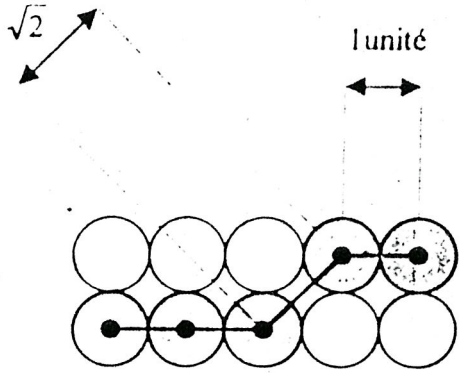
0	0	0
0	0	0
0	0	0

- *Détection par lignes de partage des eaux (watersheds)*: la technique de segmentation par lignes de partage des eaux est largement traitée dans les œuvres de segmentation d'images [39]. Elle concernait en premier temps les images binaires[45] ; puis s'est étendue aux images en niveaux de gris [46]. Son rendement est plus appréciable par rapport aux autres techniques, en particulier pour des images ayant des composantes à niveaux de gris à peu prés confondues (par exemple, le cas où les objets se touchent ou encore sont à un niveau de gris proche de celui de l'arrière plan). Son principe est le suivant :
  Pour une image donnée, on commence par établir le complément de celle ci de sorte que les objets qui représentaient un niveau de gris supérieur constituent dans l'image complément des zones " profondes ". Ces zones sont faites inondées "d'eaux" d'une manière progressive jusqu'à ce que les eaux évacuées dans différentes zones arrivent à se toucher. Les points où se rencontrent les "lacs" immergés constituent les lignes à détecter.

II.3.2 la caractérisation (*Features extraction*) : l'image une fois segmentée, ses composantes sont supposées bien discriminées et ainsi l'extraction de leurs caractéristiques est bien possible. Ces caractéristiques sont de différents genres : caractéristiques concernants l'intensité et la couleur – caractéristiques concernant les textures – caractéristiques concernant la forme et la taille ; Ce sont ces dernières qui seront discutées dans ce qui suit :

Caractérisation des formes et des tailles : les attributs qui reflètent la forme et la taille sont les caractéristiques les plus utilisées pour des fins de classification, car avec un nombre réduit (d'attributs), on peut acquérir des résultats immédiats et non ambiguës.

- Mesure du périmètre : le périmètre d'un composant est donné par la longueur du contour. La mesure est établie en considérant que la distance entre un pixel et le pixel suivant est d'une unité si ce dernier est sur le même axe ou l'axe orthogonal et est de $\sqrt{2}$ s'ils sont sur une diagonale.



- Ligne de contour de longueur $l = 1 + \sqrt{2} + 1 + 1 = 3 + \sqrt{2}$ -

- Mesure d'aire : l'aire d'un élément est évaluée comme le nombre de pixels formant l'objet.

- L'axe majeur : est défini comme la ligne qui passe par les deux points ((x1,y1) (x2,y2)) les plus distant entre eux de l'objet.
  La longueur de cet axe est donnée par :

$$l_{AM}=\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$$

  Son angle, par rapport à l'axe des abscisses est donné par :

$$\varphi_{AM} = arctg(\frac{y_2 - y_1}{x_2 - x_1})$$

- L'axe mineur : est la ligne orthogonale à l'axe majeur et qui passe par deux points de l'objet les plus éloignés. Sa longueur est de la même manière évaluée par :

$$l_{Am}=\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$$

- Mesure de l'aire du cadre renfermant l'objet : est une mesure qui est déduite des axes mineurs et majeurs. Elle renseigne sur le volume de l'objet.
  Elle est donnée par : $A = l_{AM} * l_{Am}$

- Moments spatial : On peut distinguer deux moments : le moment spatial d'ordre zéro qui est défini comme la somme des niveaux de gris des pixels formants l'objet. Et le moment spatial d'ordre un qui est formé de deux composantes : moment d'ordre 1 par rapport à x et moment d'ordre 1 par rapport à y. ces deux mesures sont nécessaires pour la définition d'une autre mesure importante qui est le centre de gravité.

- Le centre de gravité : les coordonnées du centre de gravité sont déduites à partir des moments spatiaux d'ordre zéro et d'ordre un comme suit :

$$x_c = \frac{M_x^1}{M^0}$$

$$y_c = \frac{M_y^1}{M^0}$$

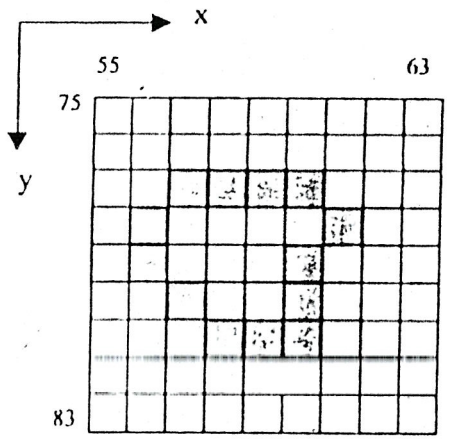
  où x_c, y_c sont les coordonnées du centre de gravité, M^0 le moment d'ordre zéro et M_x^1, M_y^1 les moments d'ordre 1 par rapport à x et y respectivement.
  Selon qu'il s'agit, d'une image binaire ou d'une image à niveau de gris, le centre de gravité calculé est soit un centre de gravité de masse(classique) , soit un centroide d'intensité.

- Codage des lignes de contour : l'approche de codage des contours est adoptée aussi comme un outil essentiel à la classification. Parmi les modes de codage on distingue le codage direct ou explicite et le codage par chaîne (code de Freimann) :
  Codage direct : les contours en question sont codés explicitement par la liste des coordonnées des pixels qui les forment.

Codage de Freimann : produit une représentation plus compacte car les points de contours(pixels) ne sont pas codés par leurs positions absolues dans un repère Oxy, mais plutôt chaque point du contour est codé par sa position par rapport au point précédant.
les esquisses ci-dessous montrent les deux formes de codage.

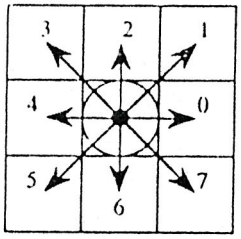
Le codage direct du contour donné ci contre produit la liste :

L : ((56 , 78) - (56 , 79) - (57 , 78) -
(57 , 80) - (58 , 77) - (58 , 81)
(59 , 77) - (59 , 81) - (60 , 77)
(60 , 79) - (60 , 80) - (60 , 81)
(61 , 78)).

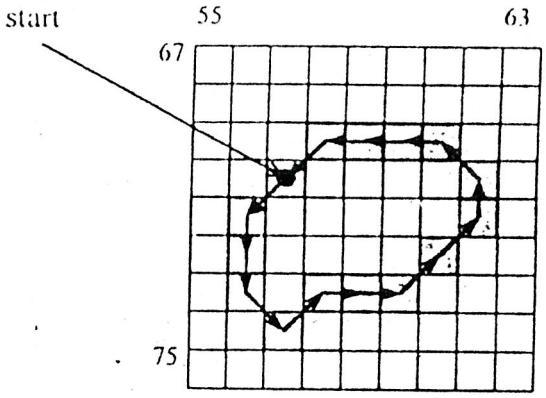


- codage direct des lignes de contours -

Pour la codage par chaîne(Freimann), le code d'un pixel contour est le numéro de la direction dans laquelle il est repéré. Comme il y a au plus 8 directions le code est une valeur dans l'intervalle [1..8].



- Numéros de directions -



- allignement des points de contour -

Le codage du contour est donné par la séquence :

5, 6, 6, 7, 1, 0, 0, 1, 1, 2, 3, 4, 4, 4, 5 .

II.3.3 la classification : la classification est la collection des résultats et des mesures de segmentation et de caractérisation de manière à établir l'identification des objets présents dans l'image. On utilise généralement une base de données *(database)* dans laquelle sont spécifiées les différentes caractérisations de l'ensemble d'objets auxquels l'élément à identifier est supposé appartenir. Une fois le module dans l'image caractérisé, il est assigné à l'objet avec lequel il a plus de similitudes en caractéristiques (forme, taille,.....).
Le rendement de la classification est donc fonction du choix du type de caractérisations. Une caractérisation est bonne si à la fois elle implique un nombre de mesures réduit et conduit à une identification rapide et correcte.

Chapitre III

Aspect matériel et logiciel

Dans ce chapitre nous allons décrire l'architecture de la machine sur laquelle nous allons implanter nos algorithmes. Différentes architectures parallèles peuvent être utilisées pour l'implantation des algorithmes de traitement d'image:

- Machines multiprocesseurs.
- Machines à *Array* processreurs.
- Machines à architecture pipeline.

Nous avons adopté une architecture multiprocesseurs MIMD à base de transputers. Le choix a été essentiellement dicté par la disponibilité du matériel et du logiciel (carte à transputers, système de développement pour transputers *TDS*).
Les algorithmes implantés au niveau de notre machine (PC/AT + carte à transputers) peuvent être implantés sur des systèmes similaires caractérisés particulièrement par une multitude de liens de communication et de mémoire locales:- Cartes multi-DSP (32C40) avec 6 liens de communication par processeur.
- Cartes multi-DSP (32C80 avec 8 liens de communication par processeur).

III.1 Les transputers IMS T800 :

La famille IMS des transputers est une gamme de produit INMOS dédiée principalement aux applications concurrentes. INMOS qualifie son produit comme étant le futur standard pour l'exploration du potentiel parallèle. Le premier membre de la gamme, le transputer 32-bit IMS T414 était introduit en 1985 et a permis de réaliser le parallélisme dans une large variété d'applications telles que la robotique, la simulation, et les traitements des signaux et d'images. Le IMS T800 est le successeur du T414 introduit en 1987. Il comporte (en plus par rapport au précédent) une unité de traitement en virgule flottante lui confiant l'exécution de 1.5 million d'instruction par seconde (1.5 MFLOP) à la fréquence 20 MHZ. Sa puissance a été ramené à 2.5 MFLOP en 1988 avec une fréquence horloge de 30 MHZ.
Le développement de l'IMS T800 fait partie du projet européen ESPRIT avec comme but la conception d'un super calculateur à base d'un réseau de nœuds de transputers excédant des performances de 1géga-flop destiné à des applications en physique, CAO et le traitement d'images.

III.2 Caractéristiques et architecture de base des transputers :

III.2.1 Caractéristiques:

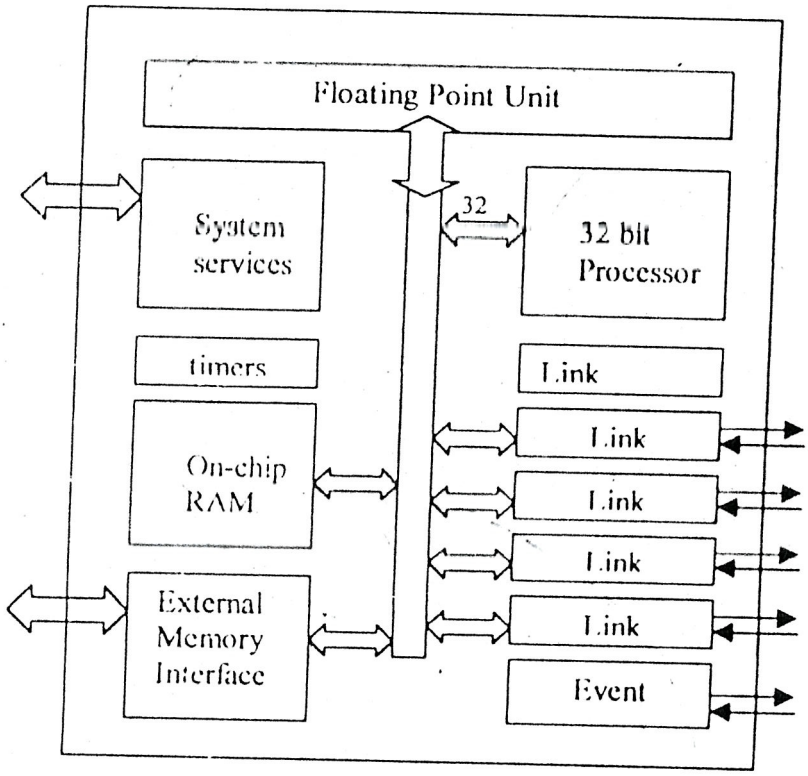
III.2.1.a Circuit programmable : la programmation des réseaux de transputers est facilitée par le langage OCCAM (langage évolué) qui constitue l'interface la plus appropriée supportant l'approche de processus communicant et permettant l'exploration efficace des liens physiques de communications.

III.2.1.b Processeur avec mémoire intégrée : Au sein du processeur s'intègre une RAM permettant la réduction des temps d'accès donc l'accélération de l'exécution programme.

III.2.1.c Liens de communication sériels : dotés de liens de communications les transputers permettent d'implanter efficacement l'approche de processus communicant.

III.2.2 **architecture** : le IMS T800 intègre sur la même puce une unité centrale de traitement CPU 32 bits,une unité de traitement flottant (FPU) ,4kbytes de RAM, 4 liens sériels, et une interface mémoire 32 bits permettant l'adressage de 4 Gbyte d'espace mémoire.

Les liens sériels jouissent d'une autonomie de fonctionnement permettant d'effectuer les traitements et la communication simultanément. De même la FPU peut opérer des traitements en concurrence avec la CPU et exécute des opérations arithmétiques flottantes en simple et double longueur (32 et 64 bits) conformément à IEEE 754. Le bloc diagramme du processeur est donné ci-dessous :



- Fig III.1 : - IMS T800 bloc diagramme -

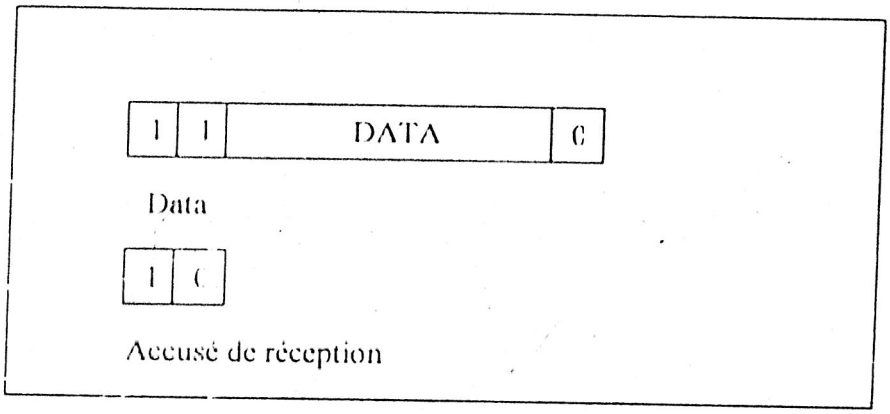
III.2.3 **codage des instructions** : toute la gamme des transputers partage un même jeu d'instructions de base comprenant un nombre réduit d'instruction avec un même format choisi pour donner une représentation compacte aux opérations les plus fréquentes.

Chaque instruction est codée sur un byte dans lequel les premiers 4 bits significatifs représentent le code d'opération et les 4 derniers l'opérande. Le format de l'opérande peut être étendue en le signalant dans le code opération comme étant positivement ou négativement préfixée.

Les instructions de base se répartissent en trois classes :

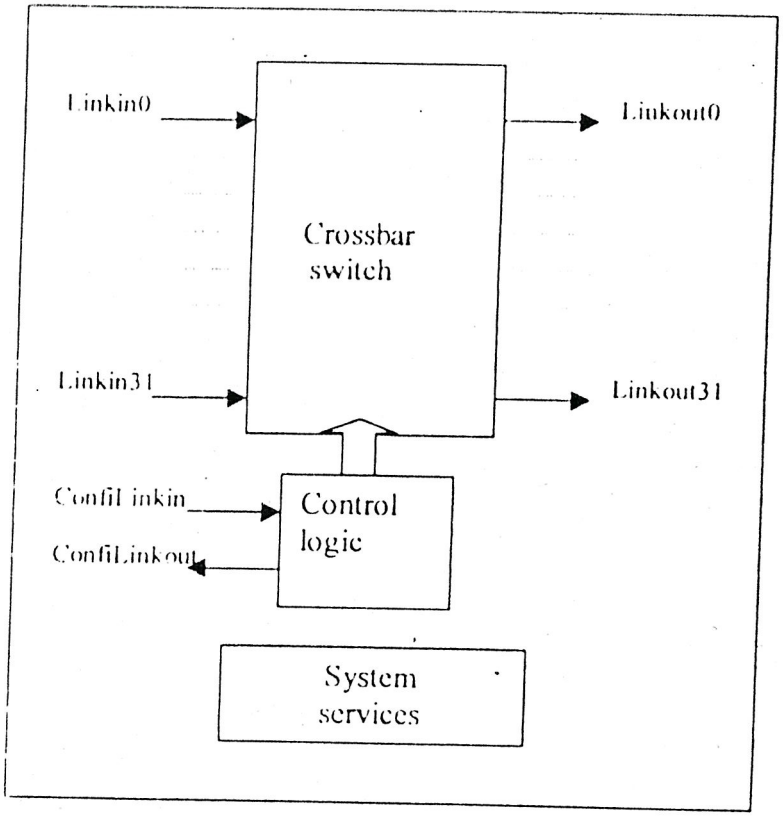
- les fonctions directes au nombre de treize qui agissent sur des opérandes appartenant à l'intervalle 0 – 15.
- Les fonctions préfixées (préfixe et préfixe négatif) utilisées avec les opérandes à format étendu.
- Les fonctions indirectes, obtenues au moyen du mot clé *operate* et interprétant l'opération comme une fonction agissant sur les valeurs de la pile de registres.

III.2.4 les liens de communications : la communication entre deux transputers s'établit avec un simple protocole de communication : le message est transmis octet par octet avec un bit de start, suivi d'un bit à 1 et un bit de stop. Le message doit être accusé de réception depuis l'autre bout du canal de communication. Le message et le signal accusé de réception sont de la forme suivante :



III.2.5 Configuration de réseaux de transputers avec le IMS C004 :

L'approche du commutateur crossbar a été introduite (§ I.3) et exposée comme un moyen d'interconnexion de réseau de processeurs. En effet avec une conception VLSI, les crossbars rendent la configuration des réseaux simple et font sorte que la topologie soit flexible. INMOS propose son (Crossbar switch), le IMS C400, avec 32 voies permettant l'interconnexion d'une manière programmable 32 liens de communication.



- Fig III.2 : -Block diagramme du IMS C004 -

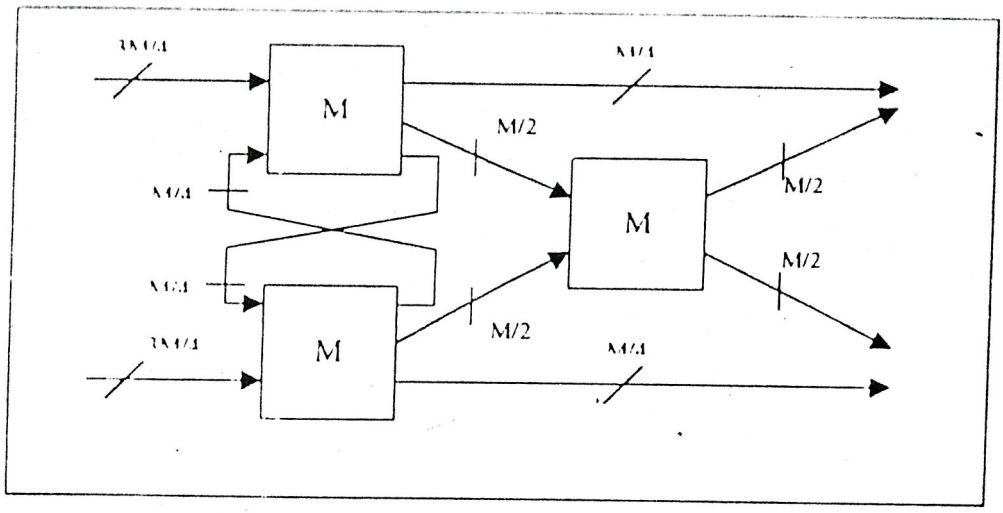
Le IMS C004 est constitué de trente-deux (32*1)multiplexeurs. A chaque multiplexeur est associé un latch 6 bits, dont cinq bits sont utilisés pour la sélection d'une entrée parmi 32 et le sixième est utilisé pour coupler ou découpler la sortie. Sa programmation s'effectue via des liens séries séparés (liens de configuration) Configlinkin et Configlinkout. Il peut être cascadé pour construire des réseaux plus larges.

Les messages de configuration sont donnés dans le tableau suivant :

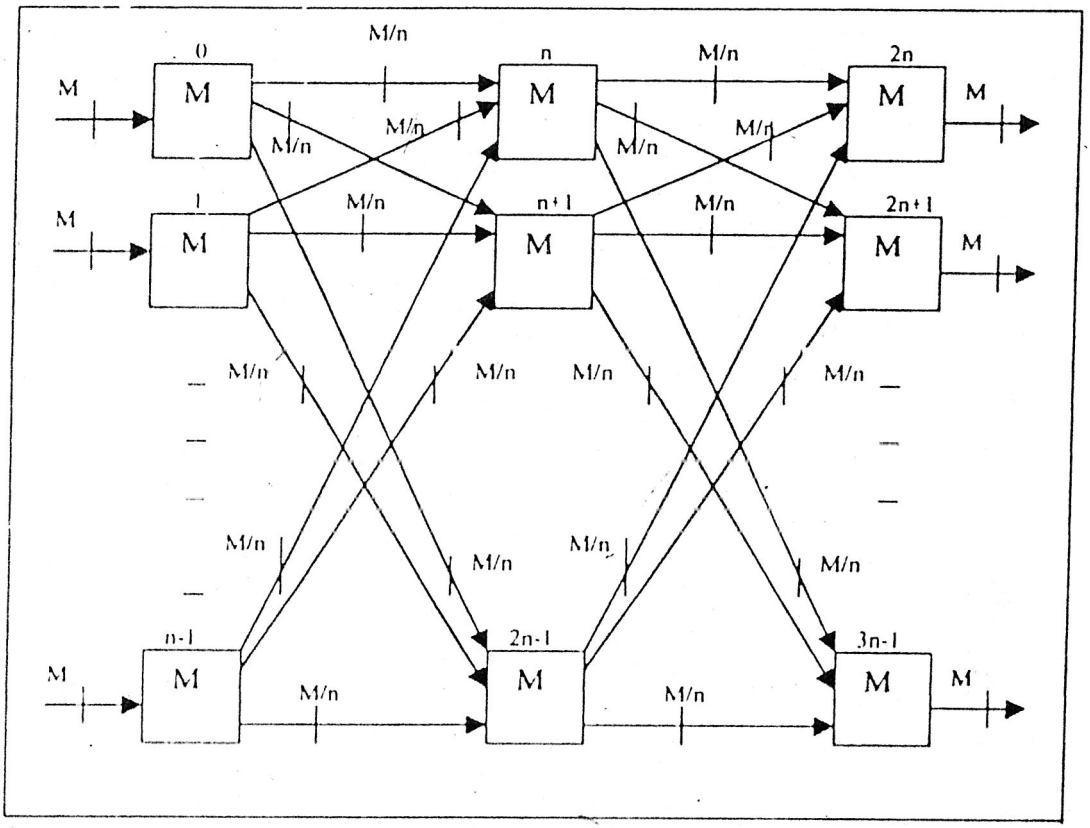
Message de configuration	Fonction
[0] [input] [output]	Connecter **input** avec **output**
[1] [link1] [link2]	Connecter input de **link1** avec output de **link2** et input de **link2** avec output de **link1**.
[2] [output]	Suspecter quelle input l'**output** est lui connecter.
[3]	Cette commande doit etre transmise à la fin de chaque phase de configuration, ce qui rend le IMS C004 prêt à établir la connexion
[4]	reset le commutateur.
[5] [output]	Découpler **output**.
[6] [link1] [link2]	Découpler output de **link1** et de **link2**.

Le commutateur introduit des délais supplémentaires dans la communication (environ 2 bits par crossbar). Ces délais croissent avec l'augmentation de la taille du réseau.
L'interconnexion de réseaux dépassant la capacité d'un crossbar est réalisée par combinaison d'une multitude de crossbars selon un schéma approprié. Deux schémas sont exposés dans ce qui suit :

Le premier établit la connexion de **3M/2** liens (entées – sorties) avec 3 Crossbars de capacité **M** [11]:



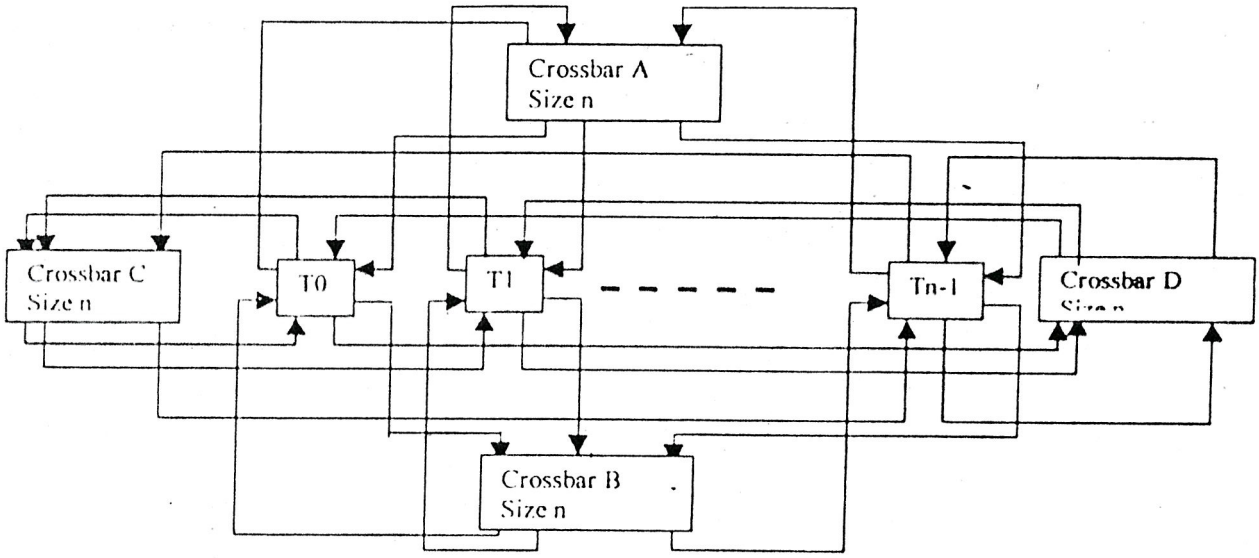
Le deuxième réalise le couplage de **n*M** entrées à **n*M** sorties en utilisant **3*n** Crossbar de capacité **M** [11]:



- Fig III.3 : - Crossbar **n*M** voies avec **3*n** Crossbar **M** voies -

La configuration complète d'un réseau de transputers peut être établi alors par 4 Crossbars de capacité au moins égale au nombre de nœuds. (Avec 4 Crossbars, 32 voies, on peut réaliser la connexion de 32 nœuds et avec 12 crossbars 32 voies combinés comme 4 Crossbars 48 voies on établit la connexion de 48 nœuds)[11].

Le schéma d'organisation est donné ci-dessous :



- Fig III.4 : - Connexion complète de n nœuds avec 4 Crossbars (n voies) -

III.2.6 Performance des transputers :

Les performances des transputers élaborées par INMOS sont comparées à d'autres processeurs dans le listing donnée ci–dessous. Les programmes de test *(the benchmarks)* utilisés pour l'évaluation de la performance sont :Le Whetstone et le Dhrystone.

Le whetestone benchmrak : est le programme test fréquemment utilisé et simule en particulier les applications scientifiques. Il inclut, en plus des traitements flottants, des opérations arithmétiques d'entiers, des opérations sur les tableaux, des appels de procédures et de fonctions, et des sauts. Le programme est considéré comme l'équivalent de l'exécution d'un million Whetestone instructions.

Le dhrystone benchmark : Sa version originale écrite en ADA est dédiée en particulier pour la mesure des performances des programmes systèmes. Son inconvénient est l'absence d'opérations de boucles et la surcharge des traitements de chaînes de caractères.

Système	Millions whetestones par seconde
IMS T800-30	6800
IMS T800-20	4548
INTEL 80386+80387	1860
VAX 11/780	1083
SUN-3	860
NS 32332/32081	728
IMS T414-20	704
NS 32032/32081	390
INTEL 286/287	300
IBM RT-PC + FPA	200

Système	Dhrystones par seconde
IBM 3090/200	31250
IMS T800-30	13400
IMS T800-20	8956
VAX 8600	6423
Gould PN 9080 Custom ECL	4992
INTEL 386-16	4300
MC 68020-17	3977
INTEL 80286-9	1976
VAX 11/780	1650
MC68000-8	1136

III.3 Le langage de programmation parallèle OCCAM : OCCAM est un langage concurrent issu du modèle de programmation parallèle CSP (communicating Sequential Processes) développé initialement à l'université d'Oxford .

OCCAM est parfaitement cohérent avec les systèmes à transputers, les liens physiques de communication, les ressources et certains paramètres matériels sont accessibles directement par le langage. La gestion des processus, leur synchronisme, et l'allocation des ressources sont confiés au programmeur à travers les primitives OCCAM.

Les principaux éléments du langage sont résumés dans les paragraphes suivant :

III.3.a Les Processus primitifs :

Les programmes OCCAM sont conçus avec trois primitives essentielles :

- l'affectation :

  variable := expression . primitive connue dans tous les langages.

- l'entrée depuis un canal :

  channel ? variable . Représente l'attente d'une donnée provenant du canal **channel** puis son affectation à **variable.**

- La sortie à travers un canal :

  channel ! expression . Représente la sortie ou l'expédition d'une valeur (**expression**) à travers **channel.**

III.3.b La communication : la communication à travers un lien ne s'établit que si les processus sur les deux extrémités du lien sont à la fois prêts. Si durant l'exécution d'un processus, une primitive de sortie est atteinte, la transmission ne s'opère qu'au moment où le processus d'autre bout du lien atteint une primitive d'entrée. la communication est donc synchronisée.

III.3.c Les structures : les processus OCCAM sont combinés pour former trois types de structures :

- structure séquentielle :

```
SEQ
  P1
  P2
  P3
```

Les processus P1, P2, P3 sont exécutés en séquence.

- Structure parallèle :

```
PAR
  P1
  P2
  P3
```

Les processus P1, P2, P3 s'exécutent tous à la fois. Ils commencent au même moment, et la structure est considérée terminée lorsque tous les processus se terminent.

- structure alternée :

```
ALT
  Chan1 ? input
    P1
  Chan2 ? input
    P2
  Chan3 ? input
    P3
```

ALT peut être aussi considérée comme une primitive de synchronisme. Le canal possédant une entrée valide le premier fait activer le processus qui lui est associé.
Dans le cas où des arrivées simultanées sur les différentes entrées sont tolérées, on doit faire appel à la variante **PRI ALT,** qui établit un ordre de priorité pour prévenir toute sorte de blocage *'deadlook'*.

```
PRI ALT
  Chan1 ? input
    P1
  Chan2 ? input
    P2
  Chan3 ? input
    P3
```

Le corps de la structure est défini par une indentation de deux espaces par rapport au mot réservé et la combinaison d'un ensemble de structures constitue également un processus.

III.3.d Les déclarations et les types :

En plus des déclarations des types de base qui sont les entiers, les flottants et les booléens, OCCAM permet la déclaration de liens de communications qui se rapportent directement à des liens physiques ou à des liens logiques (logiciels). Toute variable

n'est connue que dans le processus où elle est déclarée. Les identificateurs peuvent être aussi longs que l'on désire et l'utilisation du point(.) dans les dénominations est valable.

Exemple :

```
INT i,j :
REAL32 x,y :
CHAN OF INT canal.in , canal.out :
VAL N IS 256 :
[N][N]BYTE image :
```

III.3.e les procédures et les fonctions :

La déclaration d'une procédure est effectuée par le mot réservé **PROC** et sa fin est marquée par le signe ' : ' . la procédure comme les variables, n'est connue qu'au sein de la structure où elle est déclarée.

Exemple :

```
PROC delay (VAL INT interval)
  INT n :
  SEQ
    n : = interval
    WHILE n > 0
      n := n - 1
:
INT y :
SEQ
  input ? y
  delay (1000)
  output ! ? y
  delay (2000)
```

la définition d'une fonction prend la forme suivante :

type **FUNCTION** nom (paramètres formels)
  spécification :
  VALOF
    Processsus
    RESULT *expression*
:

Exemple :

```
INT FUNCTION max (VAL INT a, b)
  INT answer :
  VALOF
    SEQ
      IF
        a > b
          answer : = a
        a < b
          answer : = b
        a = b
          answer : = a
    RESULT answer
:
```

III.3.f les canaux et protocoles de communication :

la communication entre processus est établie à travers les canaux (variables type CHAN) qui se rapportent à des liens physiques lorsque les processus émetteur et récepteur sont sur des processeurs différents ou se rapportent à des liens logiciels si ces processus sont sur un même processeur. Les messages migrant à travers le réseau doivent être spécifiés par un protocole dans lequel est défini leur composition. Un protocole peut être une variante des quatre formes suivantes :

i) *protocole simple* : protocole contenant un seul type de données.

```
PROTOCOL word IS INT :
...
CHAN OF word canal.out :
```

Le canal *canal.out* ne peut véhiculer que les messages du type entier (INT).

ii) *protocole séquentiel* : ce protocole spécifie qu'une séquence de valeurs de types différents compose le message.

```
PROTOCOL séries.val IS BYTE ;INT ;INT :
...
CHAN OF series.val canal.in :
```

Le message canalisé à travers canal.in doit être composé d'un byte et deux entier.

iii) *protocoles pour tableaux* :utilisé pour le transfert de blocs de données (vecteurs et tableaux) .

```
PROTOCOL string IS BYTE : : [ ]BYTE :
CHAN OF string canal.out, canal.in :
PAR
  SEQ
    ...
    VAL salutation IS " salue " :
    Canal ! BYTE(SIZE salutation) : : salutation
  [20]BYTE salutation :
  BYTE size :
  SEQ
    canal.in ? size : : salutation
```

d) *protocoles variants* : avec ce protocole, différants types de données peuvent être envoyées à travers un même canal. Le message envoyé doit être une structure parmi les structures définies dans un CASE :

```
PROTOCOL messg
  CASE
    Etiqu1 ; INT
    Etiqu2 ; INT ; BYTE : :[ ]BYTE
    Etiqu3 ; BYTE ; INT
:
```

Ici, la structure est un type ou une séquence de type ordonné, qui est repérée par un étiquetage.

Exemple :

```
PROTOCOL identité
  CASE
    nom ; BYTE : :[ ]BYTE
    adresse ; INT : :[ ]BYTE
    age ; BYTE
:
```

```
CHAN OF identité info :
[50]BYTE personne :
[350]BYTE personne.adresse :
BYTE personne.age :
INT size :
SEQ
   Canal.in ? CASE
      nom ; size : : personne
        Enregistre.personne( )
      adresse ; size : : personne.adresse
        Enregistre.adresse( )
      Age ; personne.age
        Enregistre.age( )
```

III.3.g **Les boucles :**

En OCCAM la duplication peut être opérée sur les deux sortes de structures : SEQ et PAR :

La boucle :

```
SEQ i=0 FOR n
   P
```

Réalise l'exécution d'une manière séquentielle le processus (P) n fois.

Alors que la boucle :

```
PAR i=0 FOR n
   P(i)
```

Réalise l'exécution des processus P(i) tous en concurrence.

III.3.h **La configuration** : nous avons déjà fait remarquer qu'en OCCAM le programmeur est appelé à réaliser lui même la distribution et l'allocation des taches et la gestion de la communication. Les primitives OCCAM utilisées à cette fin sont essentiellement :

PLACE et PLACED PAR .

- **PLACE :** fait spécifier un lien donné par une adresse unique parmi les huit adresses disponibles (4 liens bidirectionnels).

Syntaxe :

```
PLACE link.in.num AT adress :
```

- **PLACED PAR :** décrit la distribution et l'allocation des processus et leur séquencement.

Exemple :

```
PLACED PAR
  PROCESSOR 0
     Processus0
     Processus1
  PROCESSOR 1
     Processus3
     Processus4
  PROCESSOR 2
     Processus2
```

III.4 Le système de développement pour transputers TDS d'INMOS :

Le TDS est un environnement intégré conçu par INMOS pour permettre la programmation en OCCAM des réseaux de transputers. Il comprend un éditeur intégré, un gestionnaire de fichiers, un compilateur et un débuggueur. Le TDS tourne sur des cartes à transputers connectées à un IBM PC/AT et compatibles ,permettant ainsi la gestion du clavier, de l'écran et des unités de stockage.
Des programmes Occam peuvent être écrits, compilés et exécutés à partir du système de développement. Ces programmes peuvent être configurés pour être exécutés aussi bien sur un transputer unique que sur un réseau de plusieurs centaines de transputers.

III.4.1 L'éditeur :

En plus des fonctions les plus utiles qu'un éditeur classique peut intégrer, l'interface d'édition du TDS est basée sur un concept très particulier appelé *folding* (mettre en plis). Cette opération permet de donner aux textes et aux programmes édités une apparence plus compacte rendant leur parcours, leur examen et leur manœuvre plus aisé et simple. Elle consiste à faire une sorte de plis de bloc de lignes ou de paragraphes et de les faire repérer avec une ligne unique d'un format particulier appelé *fold*. Plusieurs plis peuvent être aussi compacté dans un fold (pli).
Un *fold* est marqué lorsqu'il est fermé (plié) par trois points (...) suivi éventuellement d'un titre ou commentaire résumant le contenu. Lorsque le fold est ouvert tout le contenu est visible et délimité par des accolades.
Exemple :

```
... Déclarations
SEQ
    ... initialisation
    WHILE go.on
        Process(caracters, go.on)
```

Le fragment ci-dessus comprend deux folds (plis) fermés : (... déclarations , ...initialisations).

L'apparence du fragment avec les plis ouverts devient :

```
{{{ Déclarations
[ ]BYTE caracters :
CHAN OF cara.proto input :
BOOL go.on :
}}}

SEQ
   {{{ initialisation
    go.on := TRUE
    input ? caracters
    }}}
    WHILE go.on
        Process(caracters, go.on)
```

III.4.2 Le compilateur :

Le développement de programmes sous TDS s'effectue avec des utilitaires(*file handler utility, compiler utility, debugguer utility*) . Après l'édition du programme et pour passer à sa compilation on doit charger l'utilitaire de compilation.

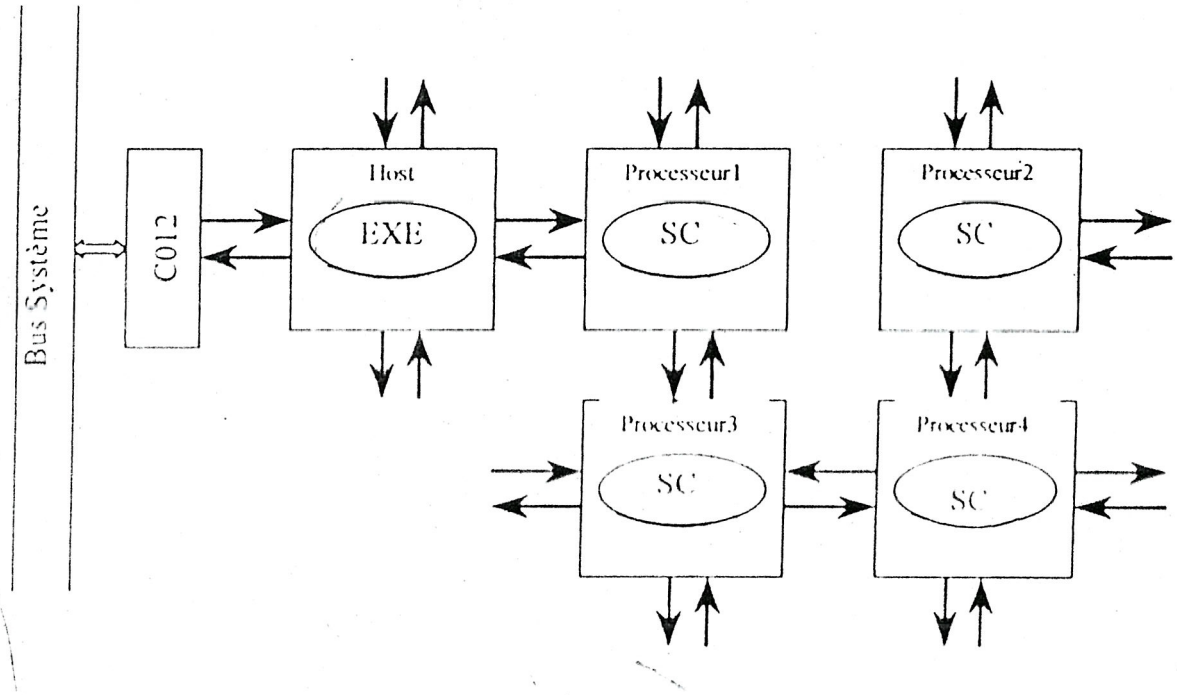
La compilation s'opère aussi bien pour un programme s'exécutant sur un seul transputer, que pour une exécution sur un réseau de transputers. Dans le dernier cas et sous l'utilitaire de compilation le *chekker* examine en outre la validité de la configuration de manière à ce qu'elle coïncide parfaitement avec l'interconnexion physique. Si ce n'est pas le cas un message d'erreur est renvoyé.

Une autre particularité du TDS est que la compilation s'effectue sous différentes catégories de modules et qui sont : SC - EXE - LIB - PROGRAM . avec :

- SC (module compilé séparément) : un module désigné par SC est un fragment comprenant un ou plusieurs processus destiné à être chargé dans un processeur (transputer) du réseau ou plus.
- EXE (module exécutable) : un module exécutable est exclusivement destiné à être chargé au niveau du transputer maître(*the host*) qui constitue l'interface entre le TDS et le réseau. Généralement un module EXE est une routine qui gère les transferts de données depuis et vers le système hôte et le réseau.
- LIB (librairie compilée) : le module LIB est une option pour la conception modulaire. Son dual, dans les langages classiques c'est l'unité comme c'est le cas dans le pascal. Les routines qui sont d'une utilisation fréquentes sont écrites et compilées dans un module LIB, et pourront ainsi être appelées depuis le programme en incluant la directive d'appel : #use nomlib.
- PROGRAM(programme réseau) : les programmes écrits pour une exécution réseau(de transputers) doivent être spécifiés comme module PROGRAM. Un module PROGRAM est constitué de : - l'ensemble des modules SC devant être alloués aux processeurs du réseau. – et d'une partie programme spécifiant la configuration du réseau et la distribution des taches et leurs allocations. Les étapes dans lesquelles un programme réseau est développé se résument comme suit :

- on commence par définir un module EXE dans lequel sont désignées les routines de gestion des entrées/sorties programme. Le module EXE étant chargé exclusivement au niveau du *host(Master)*, ces routines géreront les transferts de données depuis et vers le PC(entrées/sorties) et le réseau (*workers*).
- Chaque tache devant être attribuée à un processeur est désignée comme étant un module SC séparé. Elle est écrite comme étant une procédure puis foldée(mise en pli), et en fin spécifiée par un *MakeFoldSet* comme module SC.
- L'ensemble des taches composant l'application et qui sont foldées comme des SC sont groupés dans un fold, et sont suivis par une partie déclaration décrivant la configuration du réseau, l'allocation et la distribution des taches.
- Ce dernier fold (renfermant les procédures SC et la configuration) est spécifié par un *MakeFoldSet* comme étant un module PROGRAM.
- Le module PROGRAM est chargé ensuite dans le réseau par la commande *LoadNetwork*, et l'exécution peut être ainsi lancée.

L'allocation des modules EXE et SC sur un réseau de 5 transputers dont un est le *host* est illustrée par le schéma ci-dessous :



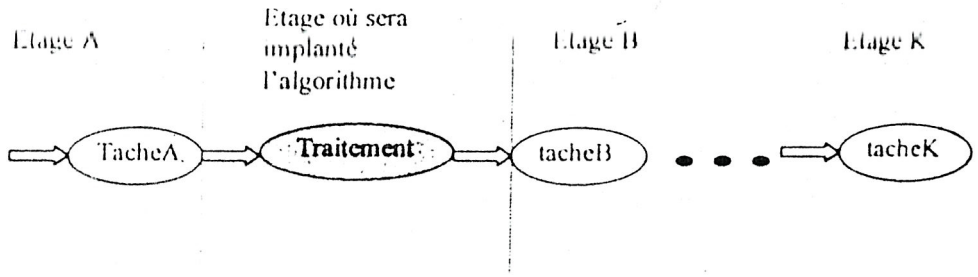
- Fig III.5 : allocation des modules EXE et SC

L'aspect de la configuration, l'allocation, la distribution des taches et la gestion des liens de communication avec le langage OCCAM est montré à travers le programme exemple donné à la fin du mémoire (page 101).

Chapitre IV

Analyse de performance

Dans ce chapitre nous présentons la parallèlisation d'algorithmes de traitement d'images bas niveau effectuée au niveau d'un étage PPF , et nous essayons à partir des mesures prélevées de définir les rapports d'accélération au respect du nombre de transputers utilisés et de voir l'impact des overheads.
Chaque algorithme est considéré comme étant une tache venant s'intercaler dans une séquence de processus (tache A, .., tache B,....,tache K)composant une application donnée. (voir Fig IV.1).
Et L'étage où sera implanté l'algorithme sera composé d'un nombre variable de transputers (1,2,3,6,8,9,12) organisés en une structure d'arbre.



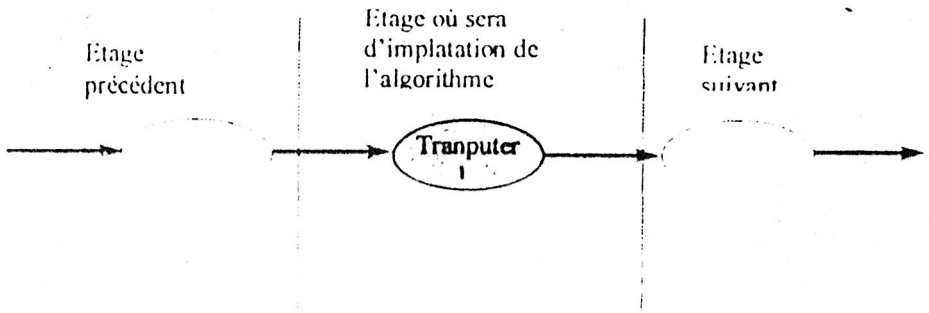
Le parallélisme adopté ici est un parallélisme de données portant sur une image en niveau de gris de 256x256 de résolution, et les mesures sont prélevées au respect du nombre de transputers utilisés.

IV.1 Configuration et stratégie de mesure :

Dans ce qui suit sont données les différentes organisations correspondantes à l'utilisation de 1,2,3 transputers :

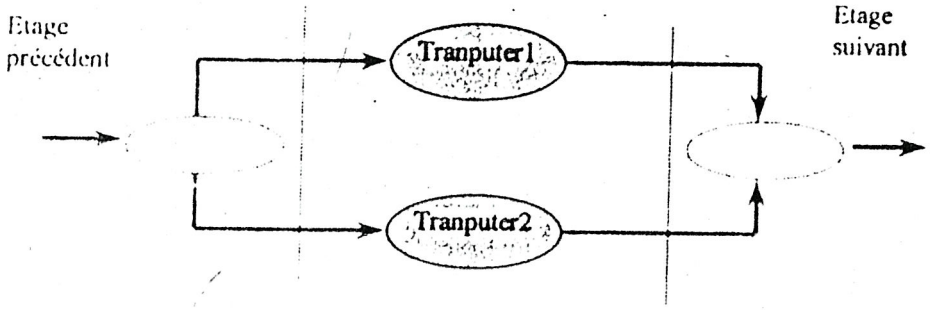
1.avec un transputer :

- Etages adjacents en pointillées.
- Etage de traitement en grisé.



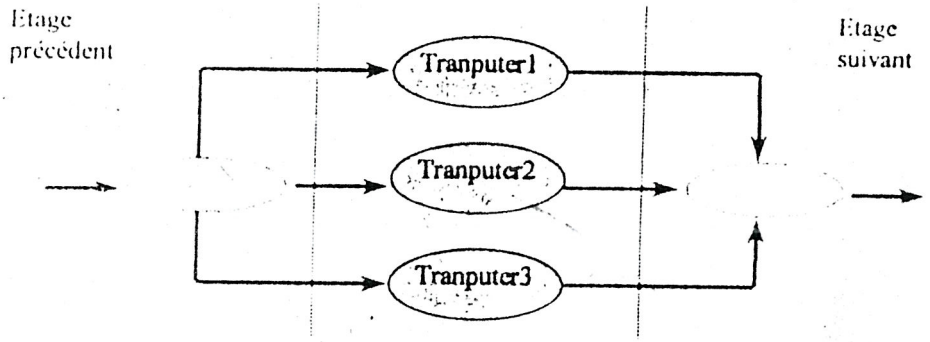
- Fig IV.1 : étage avec 1 tranputer

2.avec deux transputer :



- Fig IV.2 : étage avec 2 tranputers

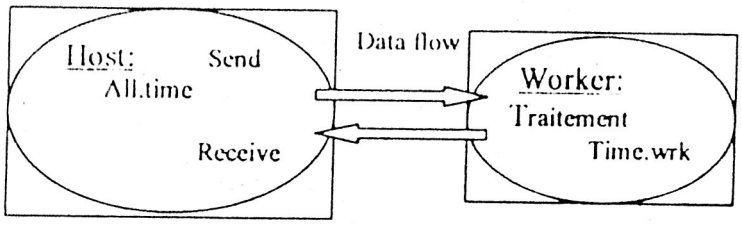
3.Avec trois transputers :



- Fig IV.3 : étage avec 3 tranputers

Les mesures prélevées pour les différentes configurations sont : le temps total désigné par ***All.time*** incluant les temps de passage de données (toute l'image) depuis l'étage précédent vers l'étage de traitement puis de l'étage de traitement vers l'étage suivant plus le temps de traitement. Ainsi que le temps de traitement au niveau du transputer *worker* désigné ici par ***Time.wrk.***

L'organisation physique utilisée et simulant l'utilisation d'un seul transputer comme étage dans un pipeline est la suivante :



- Fig IV.4 configuration physique et mesures avec un worker

On utilise donc le tranputer maître (*Host*) et 1 tranputer du réseau (*worker*). Au début, toute l'image se trouve au niveau du host sur (la mémoire locale), ensuite elle est transférée au tranputer worker où elle subit le traitement et est en fin retournée au niveau du Host. All.time est prélevé au sein du host en utilisant son horloge interne et représente le temps écoulé depuis le passage de la première donnée (pixel) au réseau (worker) jusqu'au retour de la dernière donnée de l'image(dernier pixel).
La procédure OCCAM exécutée au niveau du host sera donc une variante de :

```
SEQ
  Clock ?Time.on
  Send (all-trame)
  Receive (all-trame)
  Clock ?Time.off
```

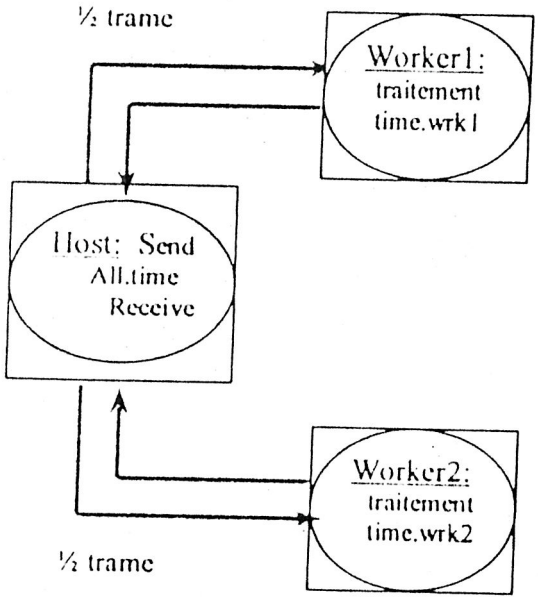
Et All.time sera donnée par time.off Minus time.on : (time.off – time.on).

Time.wrk est le temps ''brut'' de traitement et est prélevé avec l'horloge interne du worker.
la procédure exécutée au niveau du worker peut être décrite comme suite :

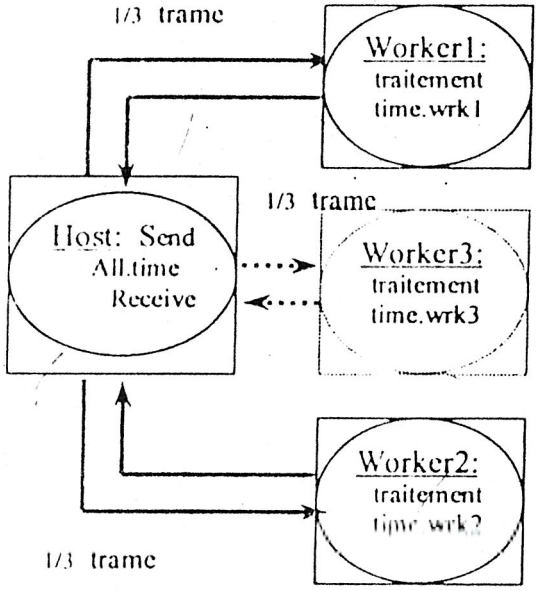
```
SEQ
  Receive(all.trame)
  Clock ?Time.wrk.on
  Traitement(trame)
  Clock ?Time.wrk.off
  Retourne (all-trame)
```

Avec Time.wrk = time.wrk.off Minus time.wrk.on .

Les mesures pour 2 et 3 transputers sont prélevées d'une manière similaire à ce qui était présenté (avec 1 tranputer) à part l'utilisation de la configuration suivante :



- Fig IV.5 configuration physique et modalité de mesures avec deux workers



- Fig IV.6 configuration physique et mesures avec trois workers

Les procédures devants être exécutées au niveau du host et des workers sont respectivement :

- Pour 2 workers :

Au niveau du host :

```
SEQ
    Clock ?Time.on
    Send.to.w1 (1/2 trame)
    Send.to.w2 (1/2 trame)
    Receive.from.w1(1/2 trame)
    Receive.from.w2(1/2 trame)
    Clock ?Time.off
```

All.time : (time.off – time.on).

Au niveau des workers :

```
SEQ
    Receive(1/2 trame)
    Clock ?Time.wrk.on
    Traitement(1/2 trame)
    Clock ?Time.wrk.off
    Retourne (1/2 trame)
```

Avec Time.wrk = time.wrk.off Minus time.wrk.on .

- Pour 3 workers :

Au niveau du host :

```
SEQ
   Clock ?Time.on
   Send.to.w1 (1/3 trame)
   Send.to.w2 (1/3 trame)
   Send.to.w3 (1/3 trame)
   Receive.from.w1(1/3 trame)
   Receive.from.w2(1/3 trame)
   Receive.from.w3(1/3 trame)
   Clock ?Time.off
```

All.time : (time.off – time.on).

Au niveau des workers : SEQ

```
Receive(1/3 trame)
Clock ?Time.wrk.on
Traitement(1/3 trame)
Clock ?Time.wrk.off
Retourne (1/3 trame)
```

Avec Time.wrk = time.wrk.off Minus time.wrk.on .

Il reste à noter que les temps de transferts et de passages de données peuvent être aussi déduits depuis ces deux mesures All.time et Time.wrk.
Les différents algorithmes expérimentés ici ont été groupés en quatre classe en fonction du taux de calcul qu'ils impliquent. par conséquent les rapports d'accélération des traitements appartenant à une même classe sont à peu prés similaires.

IV.2.1 La première classe : regroupent les traitements qui effectuent une transformation linéaire à chaque pixel de la forme :

Pixel.resultat = A* pixel.originale + B . A , B : sont les coefficients d'ajustement.

Nous avons pris comme exemple : l'ajustement de luminosité, du contraste et le seuillage (*thresholding*).

Les coefficients A,B sont définis en fonction du taux d'ajustement désiré (T) selon l'approche introduite dans §II.1 :

1. a. Pour le contraste :

$$A = \frac{255}{255 - 2*T}$$

$$B = \frac{255*T}{255 - 2*T}$$

Et le fragment réalisant l'ajustement ressemble à :

```
SEQ i=0 FOR nbr.line
   SEQ j=0 FOR nbr.colone
      IF
        Pixel.originale[i][j]<T
          Pixels.resultat[i][j] := 0
        Pixel.originale[i][j]>255-T
```

```
                Pixels.resultat[i][j] := 0
            TRUE
              Pixels.resultat[i][j] := A* pixel.originale[i][j]+B
```

Les résultats requis pour 1, 2, 3 transputers sont résumés ci-dessous :

Nombre de workers p	All.time	Time.wrk1	Time.wrk2	Time.wrk3
1 transputer (worker)	*710.91 ms*	*634.3 ms*	xx	xx
2 transputer (worker)	*377.15 ms*	*320.64 ms*	*312.64 ms*	xx
3 transputer (worker)	*253.76 ms*	*215.74 ms*	*210.37 ms*	*215.74 ms*

Et les rapports d'accélération et d'efficacité sont :

Nombre de workers p	Accélération (Speed-up) S	Efficacité E
2 transputer (worker)	*1.87*	*93.5%*
3 transputer (worker)	*2.79*	*93%*

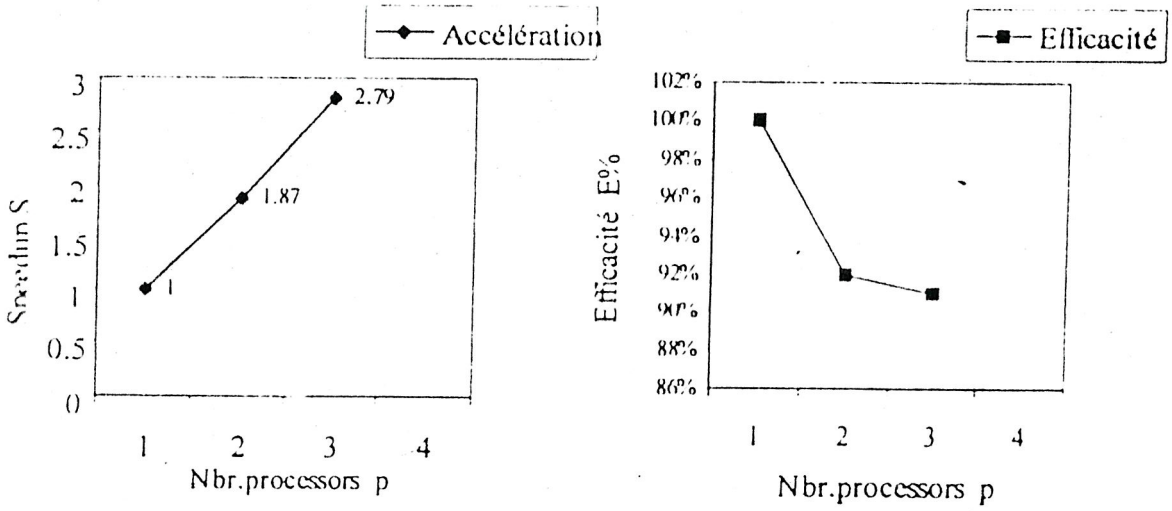
Avec S est évaluée en utilisant les temps totals (All.time) c.a.d que les temps de transfert sont inclus dans le rapport, soit :

$$S_p = \frac{All\ time\ (\ pour\ \ 1\ \ transputer\ \ )}{All\ time\ (\ pour\ \ p\ \ transputer\ \ s)}$$

Et

$$E_p = \frac{S_p}{p}$$

p nombre de transputers.



. Courbes du Speedup et de l'efficacité en fonction du nombre de processeurs .

1. b. Pour l'ajustement de la **luminance** (Brigthness) :

A = 1.
B = T.

Et le fragment réalisant l'ajustement ressemble à :

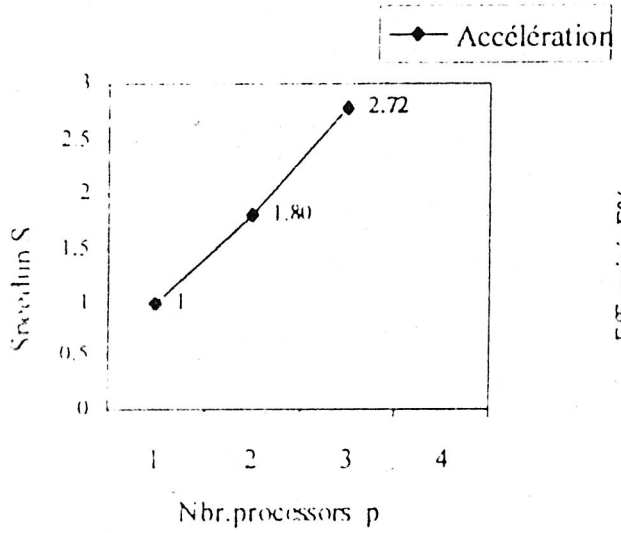
```
SEQ i=0 FOR  nbr.line
  SEQ j=0 FOR  nbr.colone
    Pixels.resultat[i][j] :=  pixel.originale[i][j] + B
```

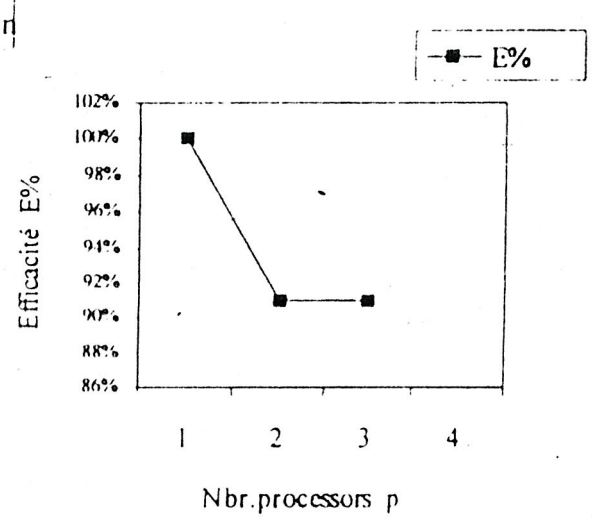
Les mesures prélevées sont :

Nombre de workers p	All.time	Time.wrk1	Time.wrk2	Time.wrk3
1 transputer (worker)	*454.57 ms*	*380.74 ms*	xxx	xxx
2 transputer (worker)	***251.46 ms***	***194.88 ms***	***191.38 ms***	xxx
3 transputer (worker)	*167.1 ms*	***129.15 ms***	***129.15 ms***	***125809 ms***

Et les rapports d'accélération et d'efficacité :

Nombre de workers p	Accélération (Speed-up) S	Efficacité E
2 transputer (worker)	***1.80***	***90%***
3 transputer (worker)	***2.72***	***90%***





1. c. Pour **le seuillage** ou binarisation (thresholding) :

A = 0.
B = Max or Min.

Le fragment réalisant cette opération ressemble à :

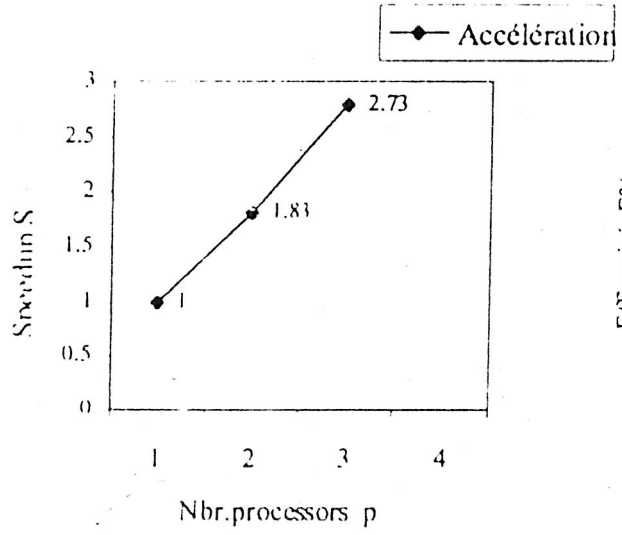
```
SEQ i=0 FOR nbr.line
  SEQ j=0 FOR nbr.colone
    IF
      Pixel.original[i][j] >= T        ------ threshold
        pixel.resultat[i][j] := Max
      TRUE
        pixel.resultat[i][j] := Min
```

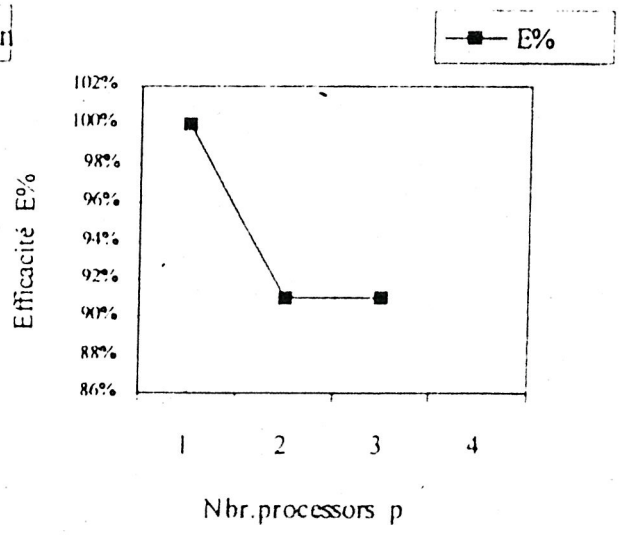
Les mesures prélevées sont :

Nombre de workers p	All.time	Time.wrk1	Time.wrk2	Time.wrk3
1 transputer (worker)	***364.86 ms***	***291.07 ms***	xxx	xxx
2 transputer (worker)	***198.46 ms***	***141.9 ms***	***-141.72 ms***	xxx
3 transputer (worker)	***133.06 ms***	***94.53 ms***	***94.78 ms***	***94.78 ms***

Et les rapports d'accélération et d'efficacité :

Nombre de workers p	Accélération (Speed-up) S	Efficacité E
2 transputer (worker)	***1.83***	***91.5%***
3 transputer (worker)	***2.73***	***91%***





Images résultats :

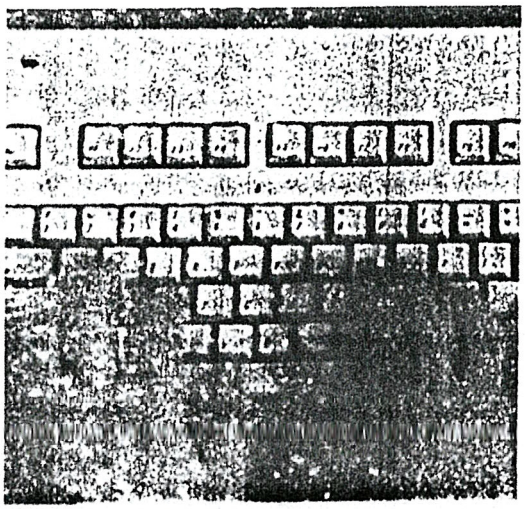


Image origine

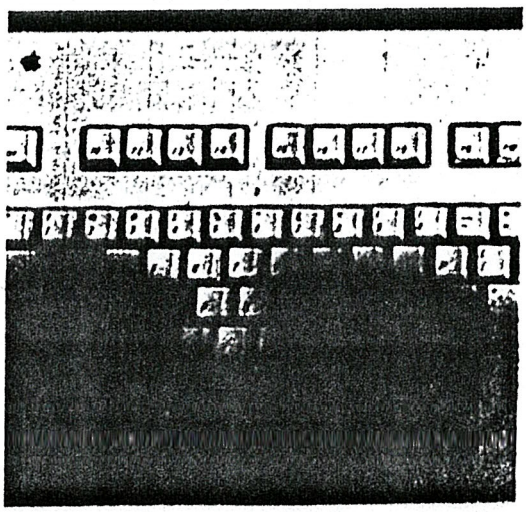


Image à contraste ajusté

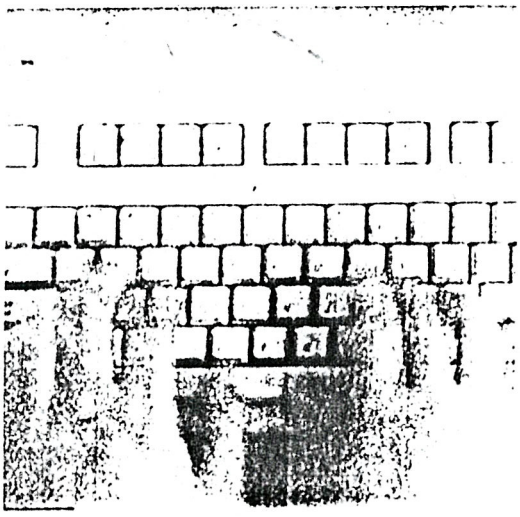


Image à luminance ajustée

IV.2.2 La deuxième classe : renferme les opérations de filtrage appliquées par un masque de convolution. Les mesures effectuées ici sont prises avec des masques (3x3) respectivement pour un passe-bas, un passe-haut et pour la détection de contour (avec les masques de SOBEL).

La procédure de convolution est implantée semblablement à :

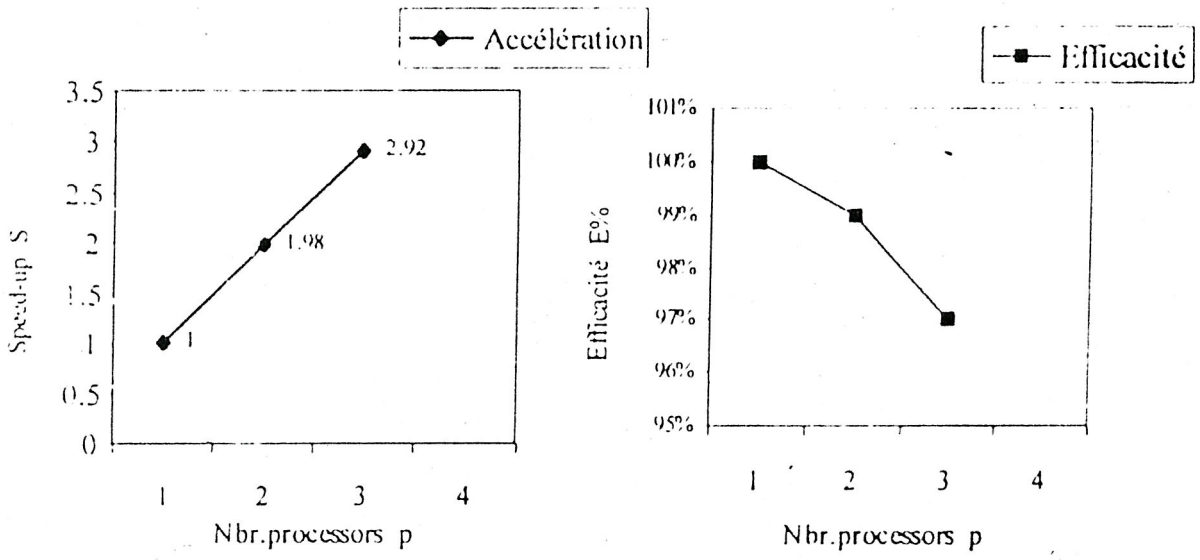
```
SEQ i=0 FOR nbr.line
   SEQ j=0 FOR nbr.colone
      Pixel.resultat[i][j] := pixel.org[i-1][j-1] * mask[1][1] + pixel.org[i-1][j]*mask[1][2]
           +  pixel.org[i-1][j+1]* mask[1][3]  +  pixel.org[i][j-1]* mask[2][1]
           +  pixel.org[i][j]* mask[2][2]  +  pixel.org[i][j+1]* mask[2][3]
           +  pixel.org[i+1][j-1]* mask[3][1]  +  pixel.org[i+1][j]* mask[3][2]
           +  pixel.org[i+1][j+1]* mask[3][3]
```

a- Les mesures prélevées pour le filtre passe-bas et passe-haut confondues :

Nombre de workers p	All.time	Time.wrk1	Time.wrk2	Time.wrk3
1 transputer (worker)	*1697.54 ms*	*1608.96 ms*	xxx	xxx
2 transputers (workers)	*856.96 ms*	*790.02 ms*	*790.02 ms*	xxx
3 transputers (workers)	*582.14 ms*	*539.26 ms*	*539.26 ms*	*539.26 ms*

Les rapports d'accélération et d'efficacité :

Nombre de workers p	Accélération (Speed-up) S	Efficacité E
2 transputers (workers)	*1.98*	*99%*
3 transputers (workers)	*2.92*	*97%*



b- Mesures pour la détection de contours par filtre de Sobel :

Nous avons pris ici deux mesures, une fois pour une détection en appliquant uniquement les masques de Sobel, et une seconde fois pour une détection de contours avec amincissement.

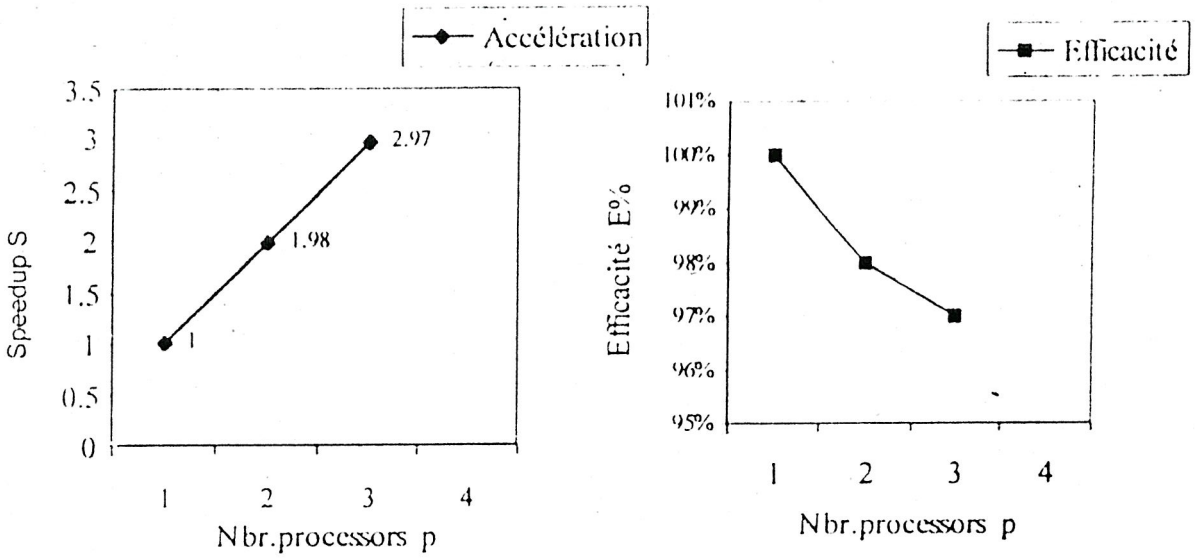
Premièrement application du masque de Sobel autant que filtre :

Les mesures prélevées :

Nombre de workers p	All.time	Time.wrk1	Time.wrk2	Time.wrk3
1 transputer (worker)	*2767.23 ms*	*2689.5 ms*	*xxx*	*xxx*
2 transputers (workers)	*1387.98 ms*	*1326.74 ms*	*1327.74 ms*	*xxx*
3 transputers (workers)	*933.44 ms*	*890.88 ms*	*890.88 ms*	*890.88 ms*

Les rapports d'accélération et d'efficacité :

Nombre de workers p	Accélération (Speed-up) S	Efficacité E
2 transputers (workers)	*1.99*	*99%*
3 transputers (workers)	*2.97*	*98%*



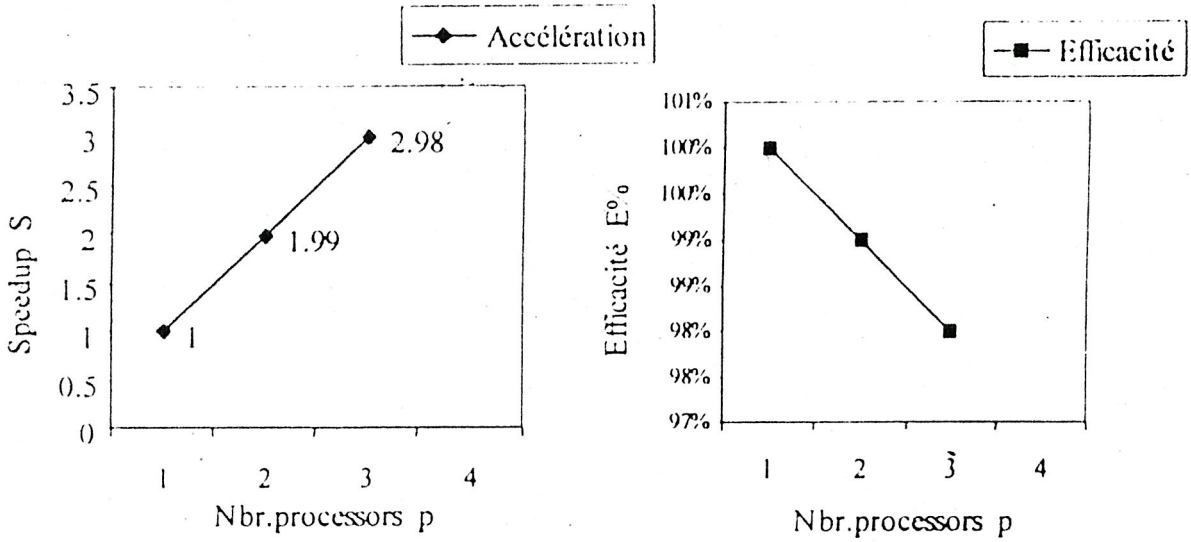
Deuxièmement détection par filtre de Sobel avec amincissement.

Les mesures prélevées :

Nombre de workers p	All.time	Time.wrk1	Time.wrk2	Time.wrk3
1 transputer (worker)	***5464.96 ms***	*5375.54 ms*	*xxx*	*xxx*
2 transputers (workers)	***2710.3 ms***	***2649.92 ms***	*2649.02 ms*	*xxx*
3 transputers (workers)	***1791.04 ms***	***1750.78 ms***	***1750.14 ms***	***1750.78 ms***

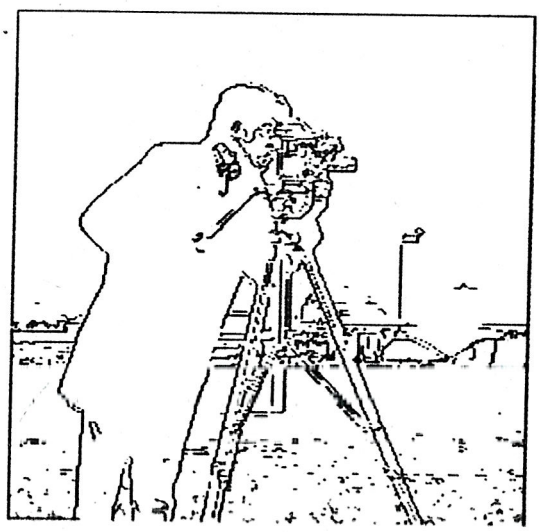
Les rapports d'accélération et d'efficacité :

Nombre de workers p	Accélération (Speed-up) S	Efficacité E
2 transputers (workers)	***1.99***	***99%***
3 transputers (workers)	***2.98***	***98%***



Images résultats :

Image originale



Détection de contours

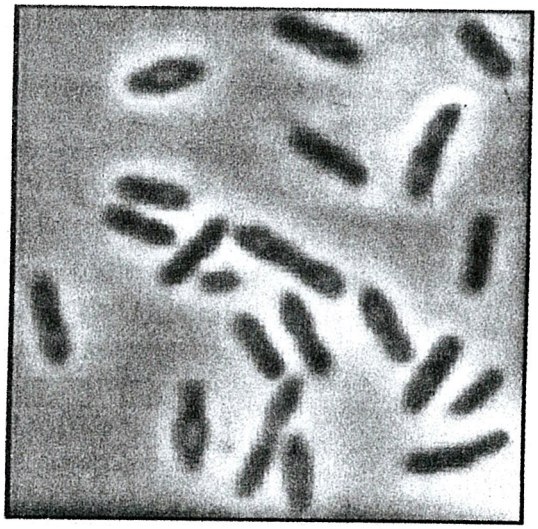


Image originale

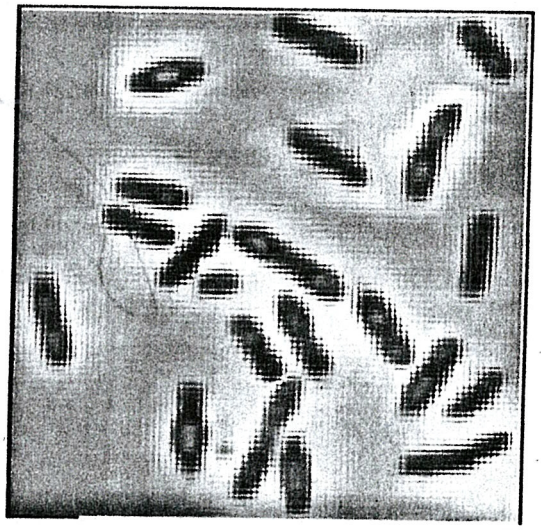


Image filtrée par passe-haut

IV.2.3. La troisième classe : regroupent l'ensemble des filtres morphologiques. Ils sont implantés ici tous par la technique du LUT (look up table).
La routine implémentant ces LUT s'inscrit comme suite :

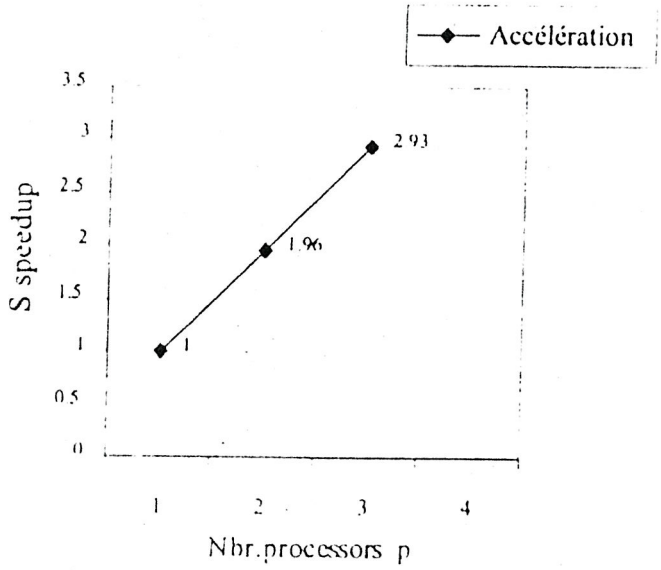
```
SEQ i=0 FOR  nbr.line
   SEQ j= 0 FOR  nbr.colone
      index := pixel.org[i-1][j-1] + pixel.org[i-1][j]* 2 +  pixel.org[i-1][j+1]*4  +
               pixel.org[i][j-1]* 8 +  pixel.org[i][j]* 16 +  pixel.org[i][j+1]* 32 +
               pixel.org[i+1][j-1]* 64 + pixel.org[i+1][j]* 12 + pixel.org[i+1][j+1]* 256
      pixel.resultat[i][j] := lut[index]
```

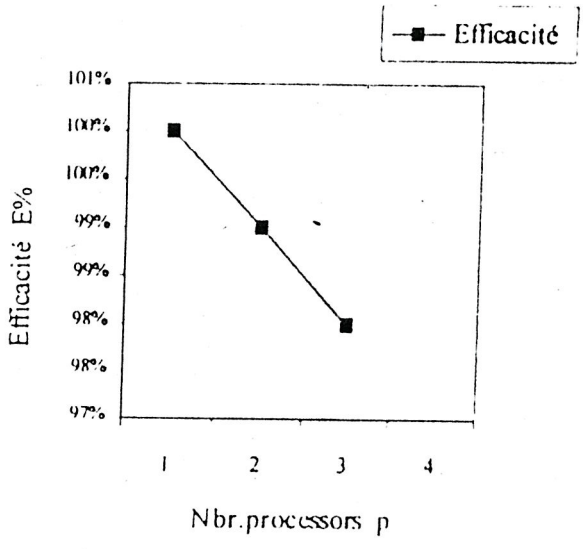
les mesures prélevées ici se rapportent aussi valablement à tout l'ensemble des traitements relevant à cette classe(à savoir l'érosion, la dilatation, le *shringing*, l'ouverture, la fermeture ...).

Nombre de workers p	All.time	Time.wrk1	Time.wrk2	Time.wrk3
1 transputer (worker)	***2560.38 ms***	***2484.22 ms***	xxx	xxx
2 transputers (workers)	***1292.16 ms***	***1232.13 ms***	***1232.13 ms***	xxx
3 transputers (workers)	***878.34 ms***	***835.65 ms***	***835.58 ms***	***835.65 ms***

Les rapports d'accélération et d'efficacité :

Nombre de workers p	Accélération (Speed-up) S	Efficacité E
2 transputers (workers)	***1.96***	***98%***
3 transputers (workers)	***2.93***	***96%***



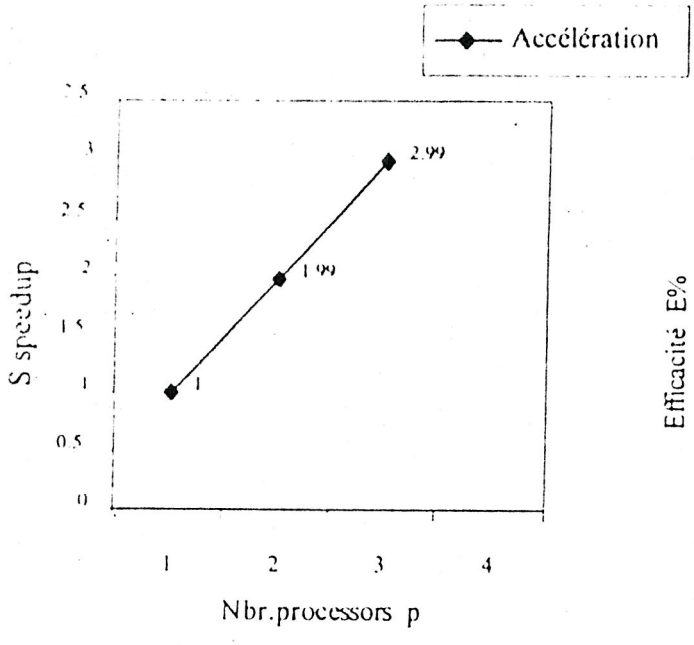


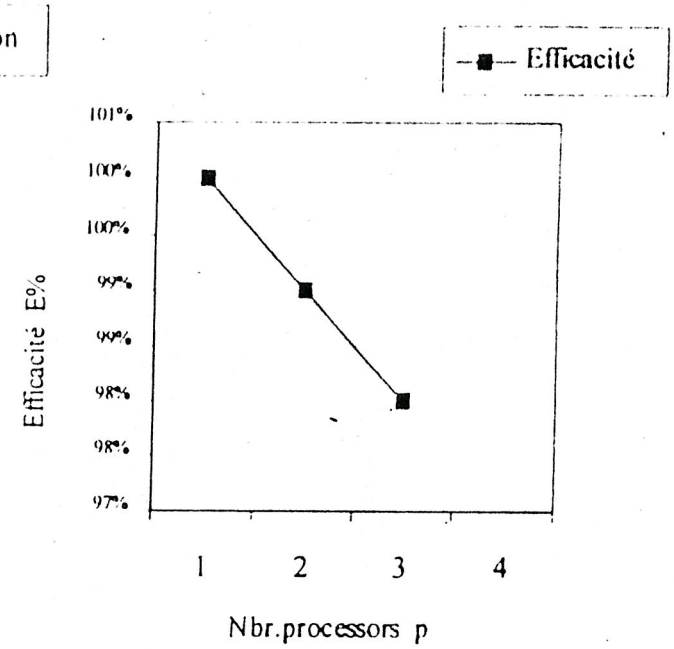
IV.2.4 La quatrième classe : rassemble certains traitements bas niveau n'ayant pas de similarité en terme de traitement avec les classes précédentes. Nous avons considéré ici le filtrage médian et l'égalisation d'histogramme. le premier se distingue par le temps un peu long par rapport aux traitements précédents alors que le deuxième s'implémente avec une configuration particulière(voir §II.1.5).
Les mesures prises pour le filtre médian sont :

Nombre de workers p	All.time	Time.wrk1	Time.wrk2	Time.wrk3
1 transputer (worker)	*9722.63 ms*	*9645.7 ms*	xxx	xxx
2 transputer (worker)	*4887.23 ms*	*4806.9 ms*	*4827.14 ms*	xxx
3 transputer (worker)	*3227.11 ms*	*3172.16 ms*	*3186.94 ms*	*3186.94 ms*

Les rapports d'accélération et d'efficacité :

Nombre de workers p	Accélération (Speed-up) S	Efficacité E
2 transputers (workers)	*1.99*	*99%*
3 transputers (workers)	*2.98*	*99%*





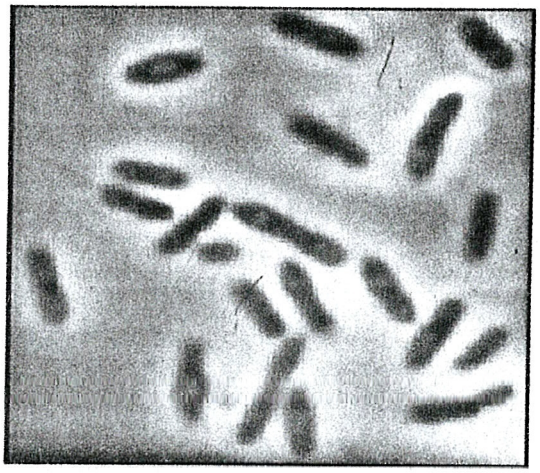


Image en niveaux de gris

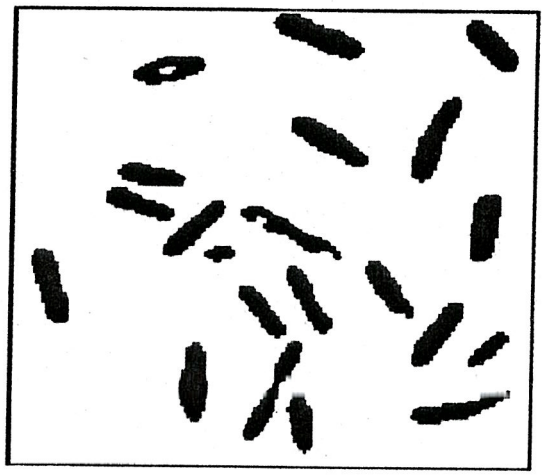


Image binarisée

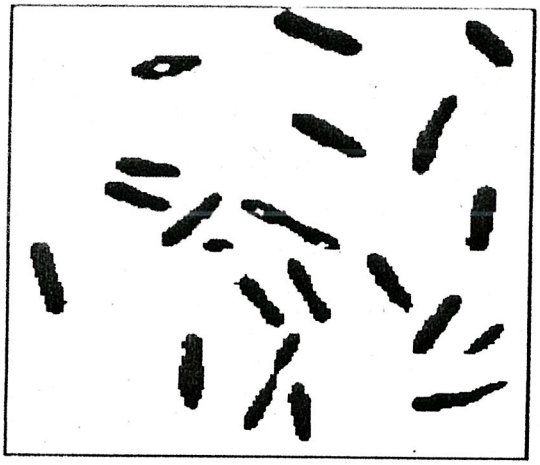
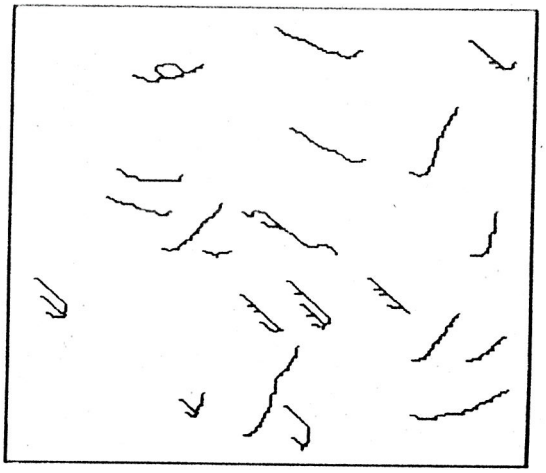


Image par érosion



skelétisation

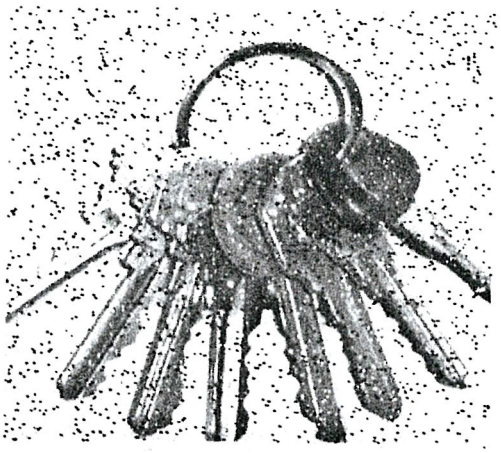


Image avec bruit ' Salt & Papper

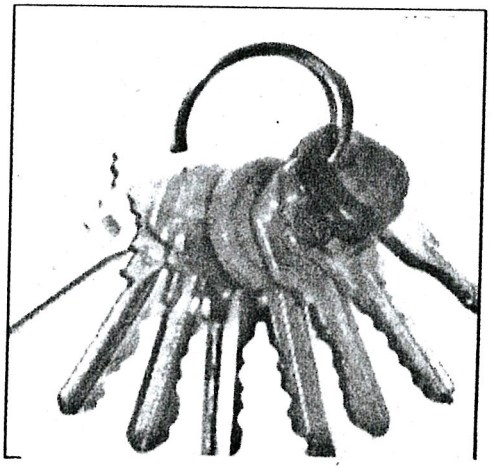
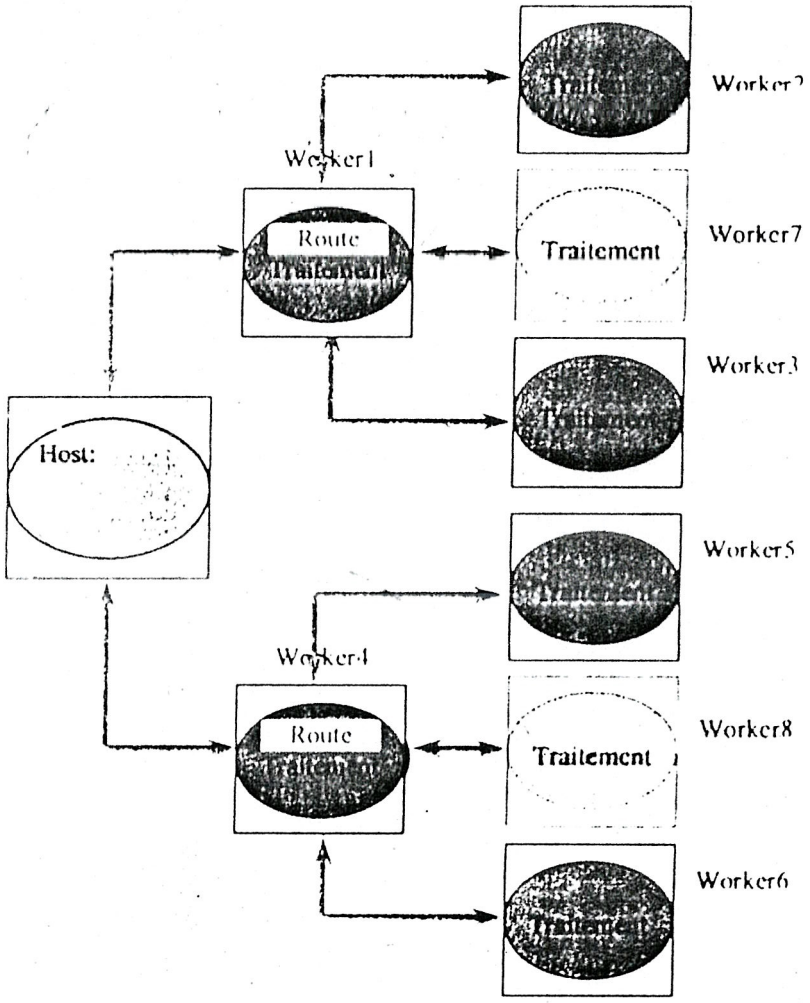


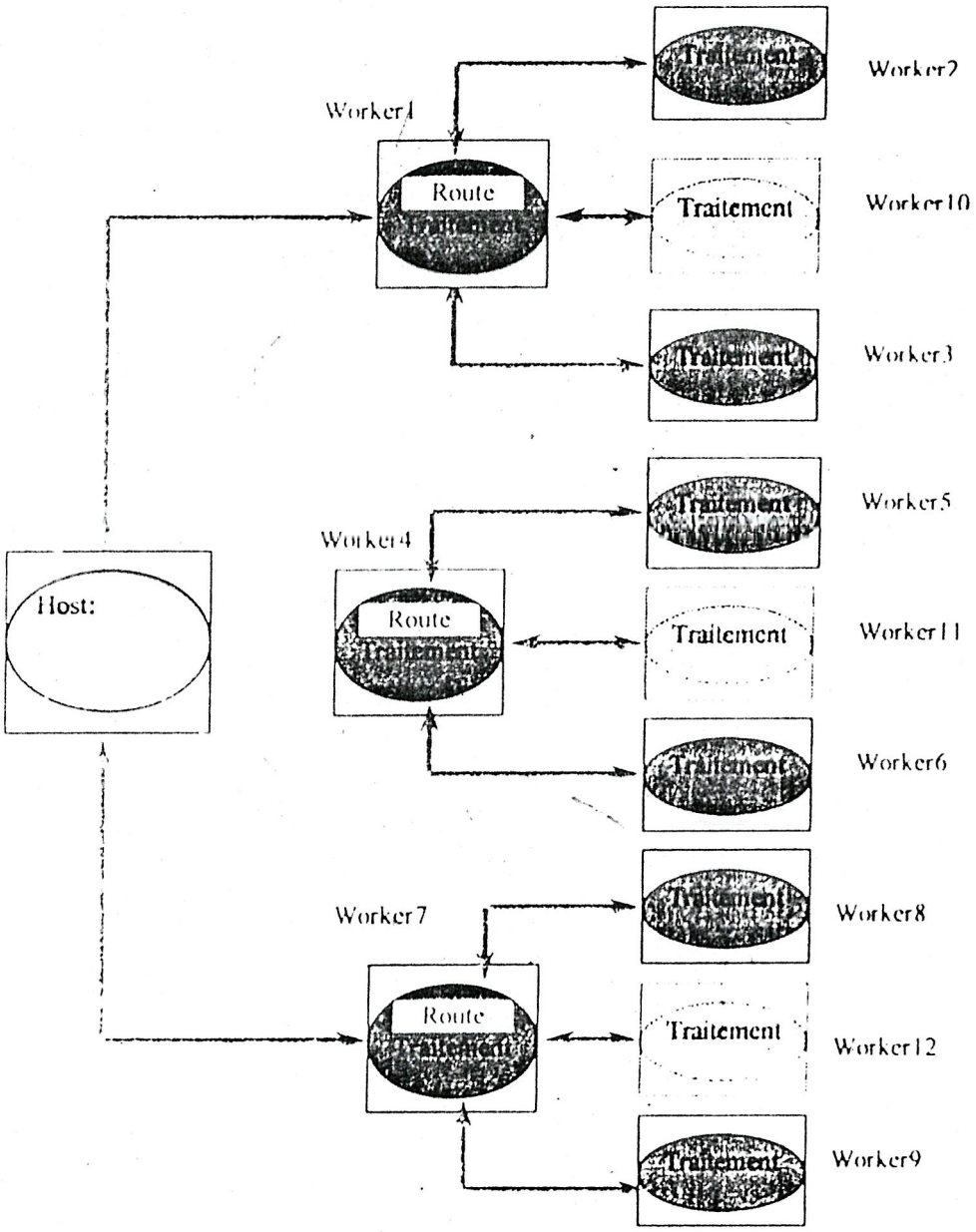
Image après filtrage médian

IV.3 Simulation pour un grand nombre de workers (6,8,9,12) :

Dans ce qui suit on présente une méthode pour une simulation de 6, 8 ,9 et 12 transputers en utilisant uniquement 3 transputers. Les configurations avec 6, 8 transputers à simuler sont illustrées sur la figure IV.6. (8 transputers en pointillé).
Le réseau de 9, 12 transputers (workers) à simuler est aussi en structure arbre et est donnée sur la figure IV.7.(12 transputers en pointillé).



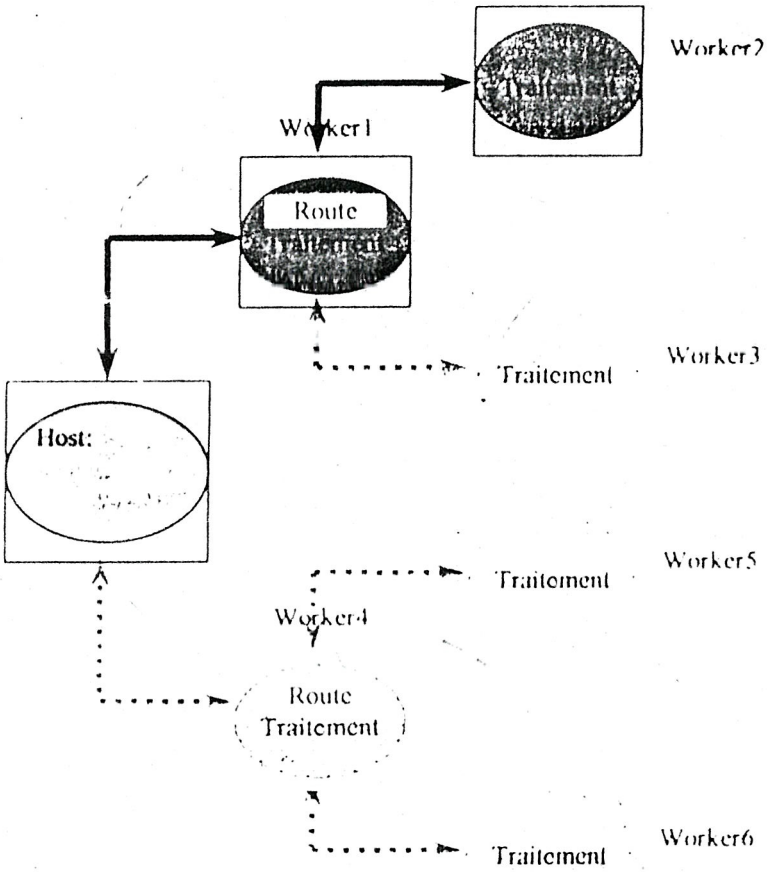
-Fig IV.6 : configuration en arbre avec 6 / 8 transputers.
(Les workers 7, 8 en pointillés concernent la simulation de 8 transputers)



-Fig IV.7 : configuration en arbre avec 9 /12 transputers. -
(Les workers 10, 11 et 12 en pointillés concernent la simulation de 12 transputers)

Le principe de la simulation est expliqué sur la figure IV.8. Le réseau à 6 workers dans cette figure est la configuration à simuler. Les transputers physiquement utilisés sont marqués en traits continus (host, worker1, worker2).
Du fait que les communications sont découplées, c'est à dire que les transferts sur la branche supérieure et la branche inférieure du Host s'opèrent en même temps et de la même manière pour les deux branches des "routeurs "(worker 1 , worker 4), le temps d'exécution total sera uniquement le temps de traitement sur une seule branche (celle marquée en trait continu gras).

Donc il suffit de suivre le parcours d'une partition d'image ($\frac{1}{6}$ *trame*) marquée en trait continu sur la figure IV.8 et de mesurer le temps de traitement correspondant.
La simulation de 8, 9, et 12 transputers est opérée avec le même principe.



-Fig IV.8 : Principe de la simulation :

i- configuration à simuler. (en pointillé)
ii- éléments physiques en trait continu.

Les essais ont donné les résultats suivants :

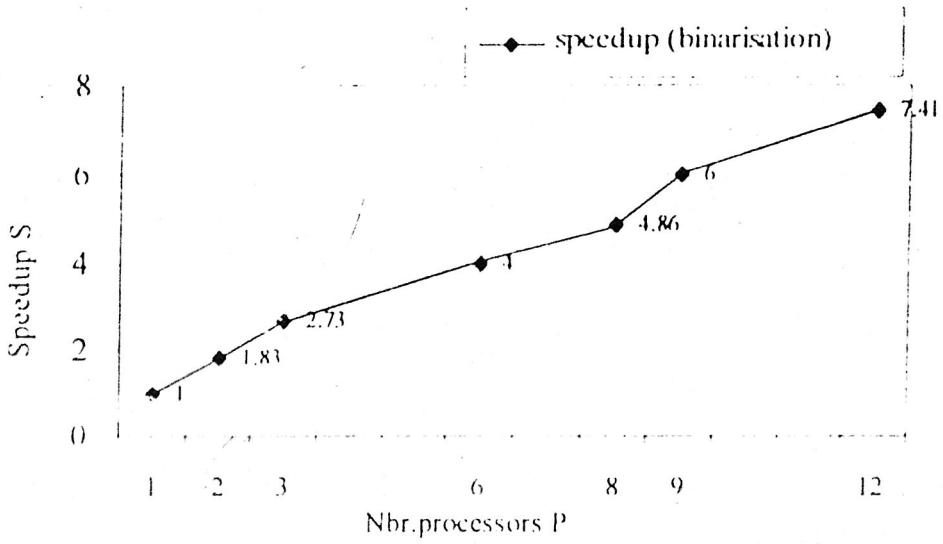
IV.3.1 Pour les traitements de la première classe :

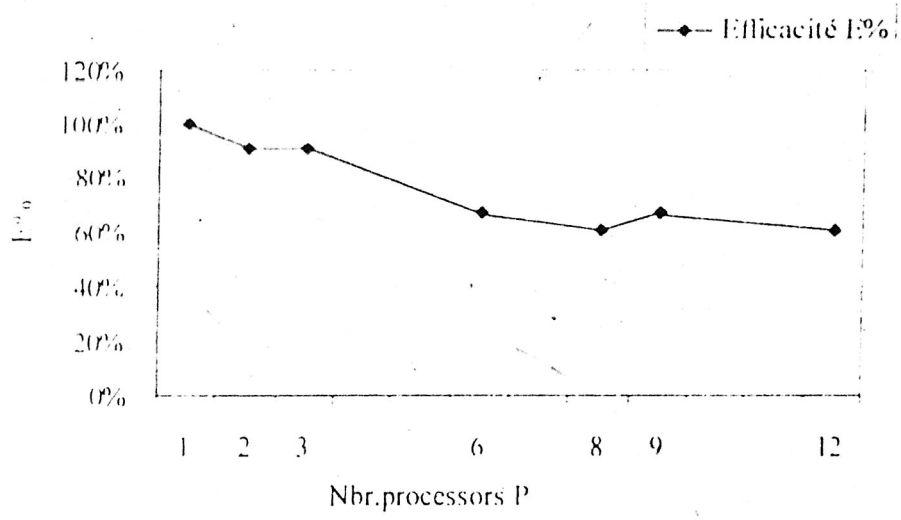
Nombre de workers p	contraste		Seuillage	
	All.time	Time.wrk	All.time	Time.wrk
6 transputers (workers)	*152.38 ms*	*108.8 ms*	*92.35 ms*	*46.08 ms*
8 transputers (workers)	*118.85 ms*	*82.24 ms*	*75.58 ms*	*34.37 ms*
9 transputers (workers)	*100.16 ms*	*72.38 ms*	*61.25 ms*	*29.95 ms*
12 transputers (workers)	*79.68 ms*	*53.95 ms*	*49.6 ms*	*22.46 ms*

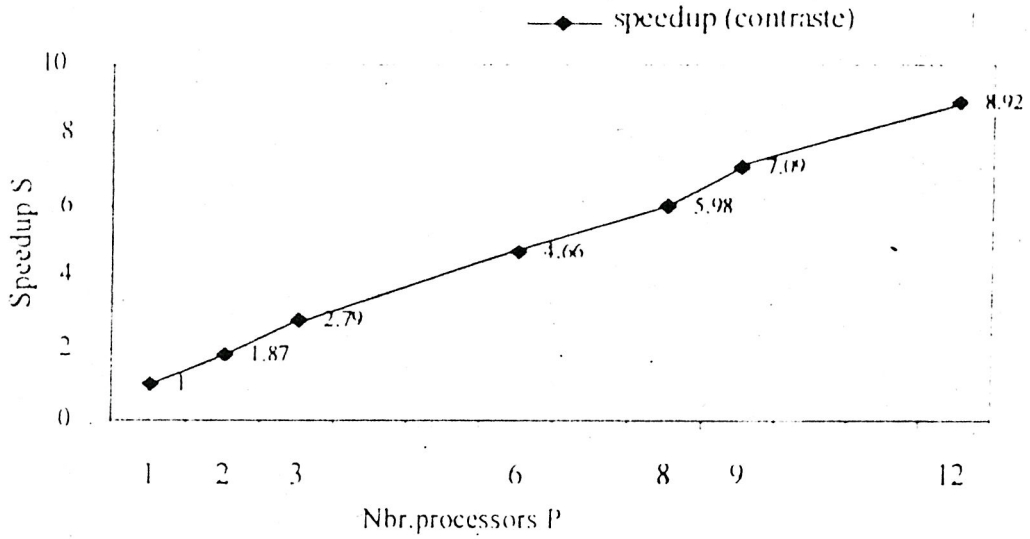
Nombre de workers p	Ajustement de luminance	
	All.time	Time.wrk
6 transputers (workers)	*111.42 ms*	*66.11 ms*
8 transputers (workers)	*90.05 ms*	*49.34 ms*
9 transputers (workers)	*73.54 ms*	*43.14 ms*
12 transputers (workers)	*59.52 ms*	*32.38 ms*

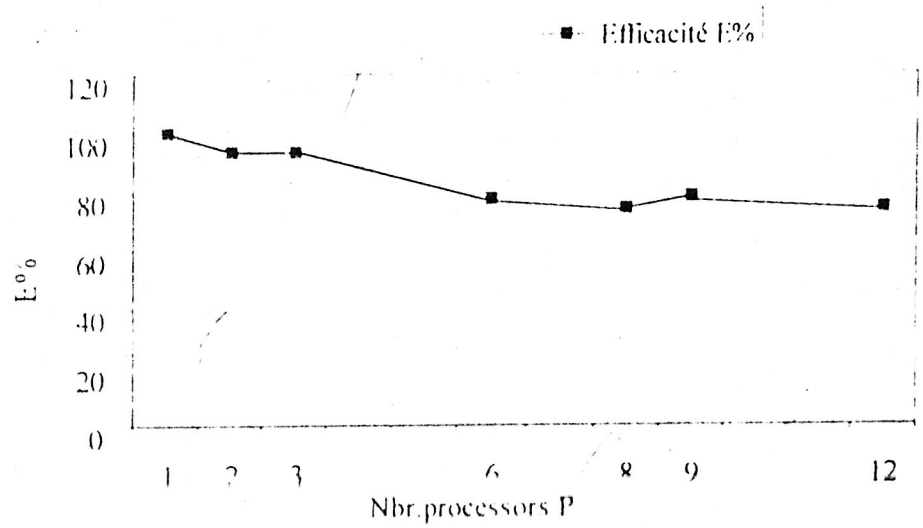
Les rapports d'accélération et d'efficacité :

Nombre de workers p	contraste		Binarisation		luminance	
	(Speed-up) S	Efficacité E	(Speed-up) S	Efficacité E	(Speed-up) S	Efficacité E
6 transputers (workers)	*4.66*	*77%*	*4*	*67%*	*4.01*	*67%*
8 transputers (workers)	*5.98*	*74%*	*4.86*	*60%*	*4.96*	*62%*
9 transputers (workers)	*7.09*	*78%*	*6*	*67%*	*6.08*	*67%*
12 transputers (workers)	*8.92*	*74%*	*7.41*	*61%*	*7.51*	*61%*


speedup (binarisation)
Speedup S
8
6
4
2
0
1
1.83
2.73
4
4.86
6
7.41
1 2 3 6 8 9 12
Nbr.processors P


Efficacité E%
120%
100%
80%
60%
40%
20%
0%
1 2 3 6 8 9 12
Nbr.processors P


speedup (contraste)
Speedup S
10
8
6
4
2
0
1
1.87
2.79
4.66
5.98
7.09
8.92
1 2 3 6 8 9 12
Nbr.processors P



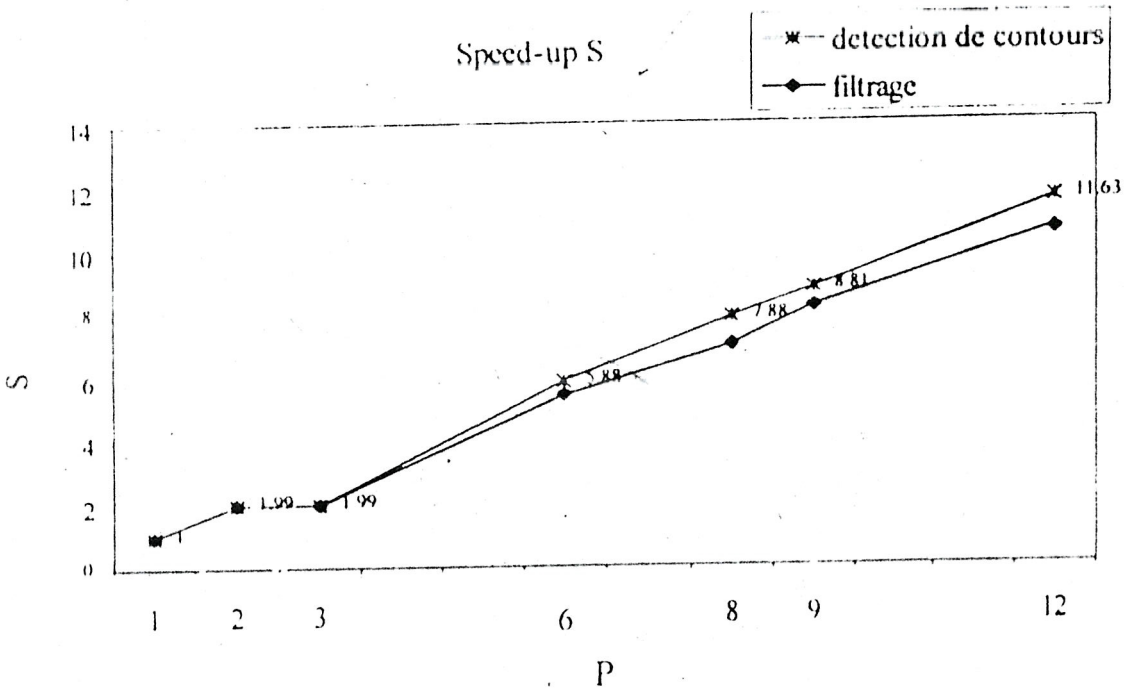
IV.3.2. Pour les traitements relevants à la deuxième classe :

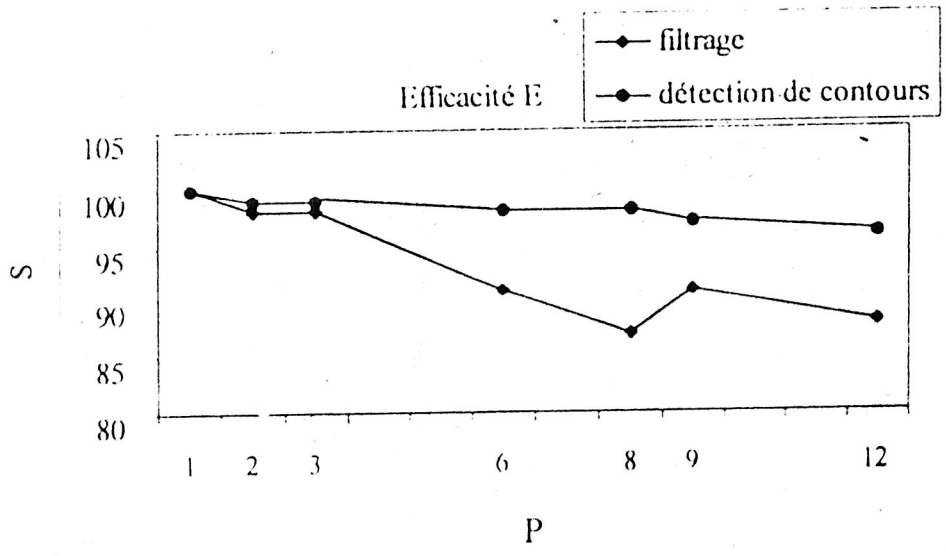
Nombre de workers p	Filtrage passe-haut		Filtrage passe-bas	
	All.time	Time.wrk1	All.time	Time.wrk1
6 transputers (worker)	***309.44 ms***	***264.77 ms***	*307.01 ms*	*263.49 ms*
8 transputers (worker)	***242.52 ms***	***201.66 ms***	*242.11 ms*	*200.77 ms*
9 transputers (worker)	***207.36 ms***	***176.45 ms***	*205.63 ms*	*175.62 ms*
12 transputers (worker)	***159.68 ms***	***132.29 ms***	*160.45 ms*	*131.71 ms*

Nombre de workers p	Détection de contour avec L'Opérateur de Sobel		Détection de contour avec amincissement	
	All.time	Time.wrk1	All.time	Time.wrk1
6 transputers (workers)	***484.35 ms***	***436.63 ms***	***929.02 ms***	***872.45 ms***
8 transputers (workers)	***363.78 ms***	***320.83 ms***	***621.46 ms***	***642.62 ms***
9 transputers (workers)	***323.46 ms***	***289.34 ms***	***619.71 ms***	***579.9 ms***
12 transputers (workers)	***247.04 ms***	***215.94 ms***	***4700.2 ms***	***433.6 ms***

Les rapports d'accélération et d'efficacité :

Nombre de workers p	filtrage		Détection de contours	
	(Speed-up) S	Efficacité E	(Speed-up) S	Efficacité E
6 transputers (workers)	*5.48*	*91%*	*5.89*	*98%*
8 transputers (workers)	*7*	*87%*	*7.88*	*98%*
9 transputers (workers)	*8.18*	*91%*	*8.81*	*97%*
12 transputers (workers)	*10.63*	*88%*	*11.62*	*96%*



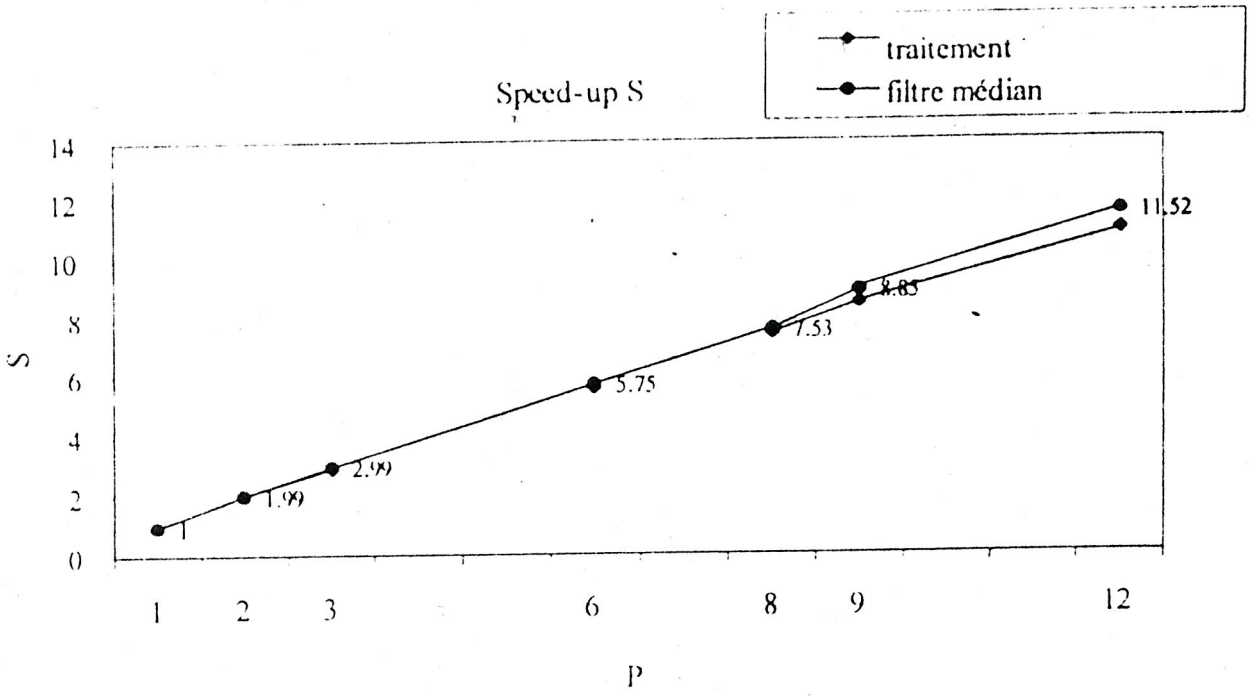


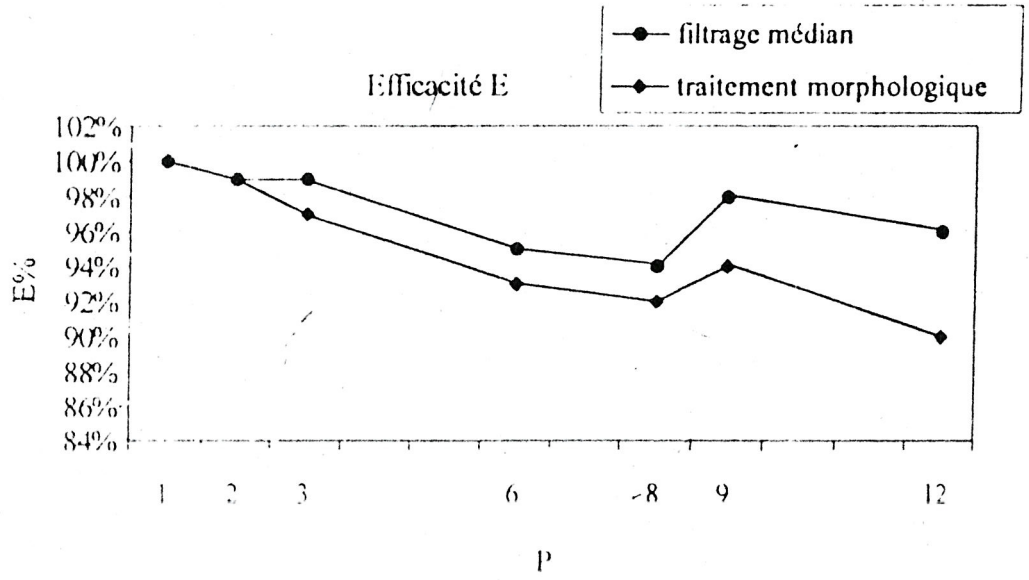
IV.3.3 Pour les traitements morphologiques et le filtrage médian :

Nombre de workers P	Traitement morphologique		Filtrage médian	
	All.time	Time.wrk1	All.time	Time.wrk1
6 transputers (workers)	*450.75 ms*	*409.98 ms*	*1688.32 ms*	*1652.03 ms*
8 transputers (workers)	*345.22 ms*	*302.67 ms*	*1290.18 ms*	*1256.38 ms*
9 transputers (workers)	*302.02 ms*	*273.34 ms*	*1122.5 ms*	*1097.79 ms*
12 transputers (workers)	*235.07 ms*	*204.93 ms*	*843.97 ms*	*825.25 ms*

Les rapports d'accélération et d'efficacité :

Nombre de workers P	Filtrage morphologique		Filtrage médian	
	(Speed-up) S	Efficacité E	(Speed-up) S	Efficacité E
6 transputers (workers)	*5.6*	*93%*	*5.75*	*95%*
8 transputers (workers)	*7.41*	*92%*	*7.53*	*94%*
9 transputers (workers)	*8.47*	*94%*	*8.85*	*98%*
12 transputers (workers)	*10.89*	*90%*	*11.52*	*96%*





IV.4 Interprétations :

Les premières conclusions qu'on peut tirer à partir de ces mesures est que le parallélisme s'avère plus efficace surtout avec les traitements à taux de calcul important. Autrement dit le speedup est aussi important autant que le temps de calcul est plus élevé par rapport au temps des transferts. Ainsi les valeurs de speedup les plus importantes sont dues aux traitements de la deuxième classe (détection de contours) et au filtrage médian. Les accélérations ont été calculées par rapport à all.time sur un (1) transputer qui au faite renferme aussi comme temps de transfert le temps de passage de la trame dans l'étage. les accélérations auront du donner respectivement 2, 3 avec 2, 3 transputers (workers) si le transfert et le traitement ont été totalement découplés, et les transferts s'opéraient en concurrence. On s'aperçoit que les mesures requises ne s'écartent plus de cet ordre idéal (Efficacité de 98% pour la 2ème classe). Ce qui signifie que le découplage des transferts n'est pas recommandé jusqu'ici(utilisation de 2, 3 transputers). Les accélérations ont été calculées selon l'approche présentée au début du chapitre.

D'une autre point de vue, on peut calculer les speed-ups comme étant non pas le rapport :

$$\frac{all.time \quad (1 \quad processor)}{all.time \quad (P \quad processors)}$$

mais à partir du rapport : $\frac{Time.wrk \quad (1 \quad processor)}{All.time \quad (P \quad processors)}$.

Autrement dit, on ne prend compte que du temps de traitement sur 1 processeur (dénominateur), le temps de transfert est omis. Ainsi, les speed-ups seront dans les ordres de valeurs suivantes :

Pour 1ère classe : sur 2 (*workers*) S = 1.68 . sur 3 (*workers*) S = 2.5.
Pour 2ème classe : sur 2 (*workers*) S = 1.98 . sur 3 (*workers*) S = 2.98.
Pour 3ème classe : sur 2 (*workers*) S = 1.92 . sur 3 (*workers*) S = 2.82.
Pour 4ème classe : sur 2 (*workers*) S = 1.98 . sur 3 (*workers*) S = 2.98.

Ces rapports seront utiles pour la prédiction de performance au cas où il s'agit d'un multiplexage temporel.

On remarque aussi que le speed-up diminue avec l'utilisation de 6,8,9,12 workers (2 étages) par rapport à l'utilisation d'un seul étage(utilisation de 2, 3 workers). Ce ci est tout à fait logique car les transferts augmentent en accroissant le nombre d'étages. Bien que pour les traitement à taux de calcul élevé cette diminution n'est que très légère.
Aussi la pente au niveau du segment reliant les speedups correspondant à P=8 et P=9 est importante par rapport à celle du segment reliant les speedups correspondant à P=6 et P=8 de la courbe.
D'ou l'efficacité obtenue en utilisant 9 workers est supérieure à l'efficacité obtenue avec 8 (E_9 = 98% alors que E_8 =94%). Ce qui conduit à dire que jusqu'à une certaine limite l'accélération est meilleure tant qu'il y a plus de liens de communication.
Grosso modo le parallélisme est efficace jusqu'ici (jusqu'à une utilisation de 12 workers) pour les traitements avec un taux de calcul pareil ou excédant celui des traitements de la deuxième classe.
Le temps de transfert est donné théoriquement par : $t_{com} = \alpha + \beta.L$ [41][42] où α est le temps de start-up ou de mise en place des transferts , L la longueur du message et β est un paramètre lié à la vitesse de transfert sur les liens donné en secondes/octet.
Dans ce qui suit seront développés les calculs nécessaires à la validité de cette formule pour les différentes classes.
La cadence de transfert sur les liens de communication des transputers tel que signalé dans les caractéristiques du transputer est de 20 Mbit/s. donc on admet que le β est de 0.4 μ s /octet.

Le temps de communication est évalué comme suit : $2t_{com}$ = All.time – time.wrk sur 1 tranputer pour tous les traitements. Et est pour la majorité des traitements dans l'intervalle : [73.8 - 78.8 ms] de largeur de 4ms correspondant bien au taux d'instabilité de l'horloge remarquable aussi au niveau des time.wrk1 et time.wrk2 qui tentant sont égaux tentant sont en décalage d'envers 4 ms.
l'horloge en est la seul cause à cette différence car d'une part en reprenant une exécution donnée plusieurs fois on repère pour certaine reprises des mesures identiques et pour d'autres des mesures légèrement différées.

Le temps de passage d'une trame est donc de $\frac{76}{2} = 38\ ms$ si on prend comme temps $2t_{com}$ =76 ms : la moyenne dans l'intervalle précédent. Et comme la résolution est de 256x256 :

$$\alpha = t_{com} - \beta.L = 38 - 0.4*10^{-3}*256^2\ ms = 11.8 ms$$

Pour voir dans une mesure donnée la validité de loi d'Amdhal. On considère les transferts comme étants une tache séquentielle non parallélisable, et le traitement comme une tache parallèlisable à 50% pour une exécution sur deux workers.
Ainsi le processus P est composé de P_{com} *et* P_{trait} *en séquence* avec :
$f_{com}=1$ *et* $f_{trait}=0.5$ les fraction séquentielles associées respectivement à P_{com} (processus de transfert) et P_{trait} (processus de traitement).
Donc on devra avoir :

$$S = \frac{1}{f} = \frac{1}{\frac{t_{com} * f_{com} + t_{trait} * f_{trait}}{t_{com} + t_{trait}}} = \frac{t_{all}}{t_{com} + t_{trait} * 0.5}$$

En utilisant les mesures du filtrage et en les rapportant à cette formule on a :
S = 1.94, au moment où le S mesuré est de 1.99 . Cela est du au fait que le transfert n'est pas tout à fait un processus séquentiel mais plutôt il y a lieu à un multiplexage temporel avec le temps de traitement.
La même remarque peut se faire pour 6, 8, 9, 12 workers. Où l'on a toujours les processus P_{com} et P_{trait} . P_{com} processus de communication comme étant une tache non parallélisable. Et P_{trait} processus de traitement avec les fractions séquentielles :
$\frac{1}{6}, \frac{1}{8}, \frac{1}{9}, \frac{1}{12}$ respectivement pour les exécutions avec 6, 8, 9, 12 workers.
On arrive ainsi, à définir la marge inférieure du speedup qu'on devra au minimum atteindre. Si on a des speedup en dessous de cette marge ça signifiera bien qu'il y a lieu à un mal arrangement du réseau. Cette marge est donnée sur la courbe ci-dessous (figureIV.4) :
En utilisant les mesures du filtrage morphologique :

P_{com} = 76.16 ms P_{trait} = 2560.38 ms

et :

$$S = \frac{1}{f} = \frac{1}{\frac{t_{com} * f_{com} + t_{trait} * f_{trait}}{t_{com} + t_{trait}}}$$

On a pour 6 workers :

$$S_6 = \frac{1}{f} = \frac{t_{all}}{t_{com} + t_{trait} * \frac{1}{6}} = 5.22$$

Pour 8 workers :

$$S_8 = \frac{1}{f} = \frac{t_{all}}{t_{com} + t_{trait} * \frac{1}{8}} = 6.62$$

Pour 9 workers :

$$S_9 = \frac{1}{f} = \frac{t_{all}}{t_{com} + t_{trait} * \frac{1}{9}} = 7.27$$

Et pour 12 workers :

$$S_{12} = \frac{1}{f} = \frac{t_{all}}{t_{com} + t_{trait} * \frac{1}{12}} = 9.04$$

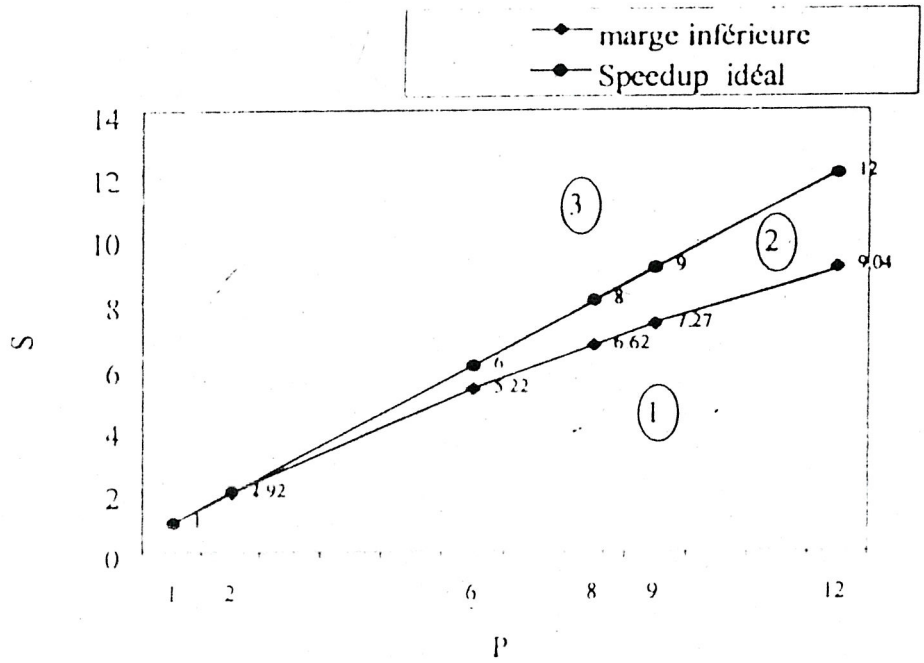


Fig IV.4 : tracé de la marge inférieure de S par rapport à S idéal

Les mesures ci-dessus permettent de distinguer trois zones sur le graphe :

- La zone 3 est inabordable (zone impossible).
- La zone 2 correspond à des valeurs de speed-up considérées comme admissibles (normales).
- Des valeurs de speed-up appartenant à la zone 1 signifient une inefficacité.

On peut bien vérifier que tous les speed-ups obtenus précédemment (sections IV.2 et IV.3) appartiennent à la zone 2 qu'on a considérée comme zone des accélérations efficaces.

Chapitre V

Application : Tri de pièces de monnaie

L'application présentée ici est un exemple simple de procédé de vision industrielle. Nous voulons mettre en évidence la notion de décomposition d'un procédé industriel donné en une séquences de taches, d'en faire les adaptations et les allocations convenables selon une structure PPF sur un système à transputers et d'analyser les rapports de performance requis en fonction des organisations adoptées dans chaque étage PPF. En fait la réalisation d'un tri parfait de pièces de monnaie s'inscrit dans l'axe de reconnaissance de forme et de reconnaissance de caractères.

V.1 Présentation du problème :

La classe expérimentée ici est l'ensemble de pièces de monnaie algériennes (voir figure V.2) et le tri ne portera que sur le critère de forme extérieure (polygonale ou circulaire), sur le volume, l'homogénéité de régions et éventuellement sur le motifs de formes et de caractères. Le problème de tri est décrit comme suit :

On envisage de faire un tri de pièces de monnaie qui défilent sur un tapis roulant en utilisant un système d'acquisition relié à notre système à base de transputers gouvernant un bras manipulateur qu'on appel ''trieur''. Pour un traitement en temps réel on doit pouvoir ajuster les paramètres suivant :

- la vitesse d'entraînement du tapis roulant.
- Le temps de réponse du système.
- Le temps de réponse du trieur.

En supposant que la période d'échantillonnage (temps séparant 2 images successives) soit égale à T , le temps de traitement est égal à T_r et le temps de réponse du trieur est égal à T_{tr} , la condition suivante doit être vérifier :

$$T_r + T_{tr} < T$$

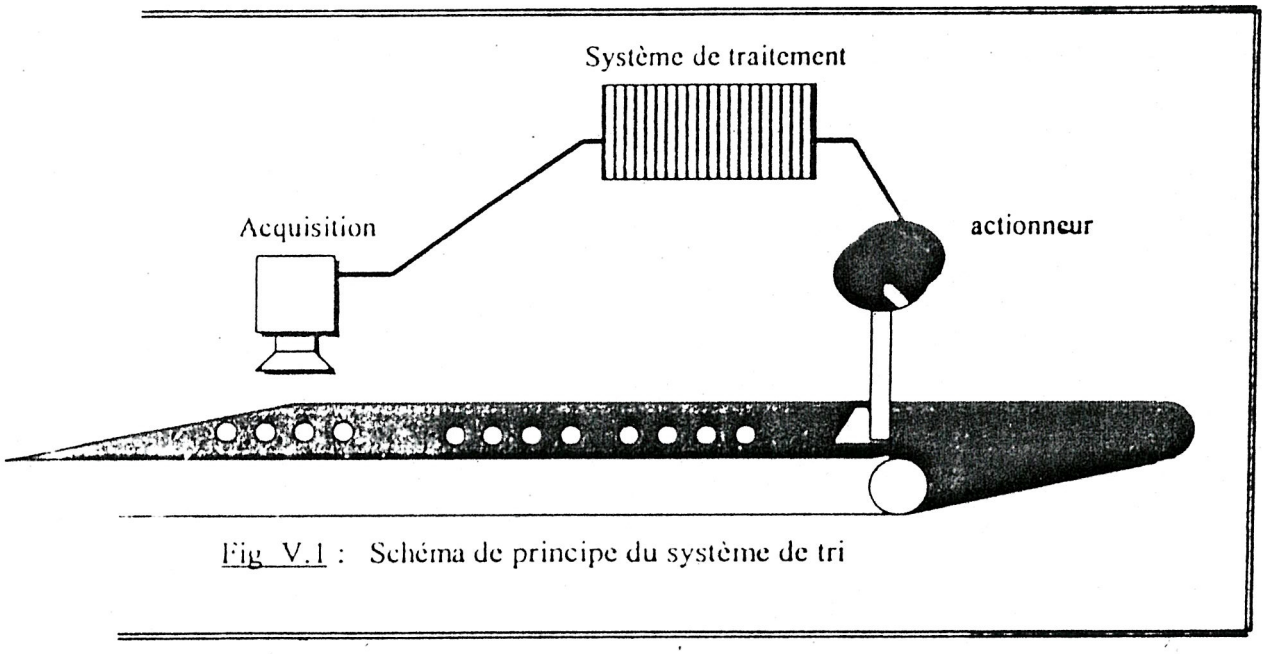


Fig V.1 : Schéma de principe du système de tri

V.2 La procédure de tri et son *Mapping*:

La procédure de tri établie est un ensemble de traitements séquentiels ' bas niveau'. Et comporte les étapes suivantes :

- L'étape de seuillage (binarisation) : Elle doit être appliquer de manière à bien repérer le corps de la pièce, et pour pouvoir procéder aux traitements ultérieurs.
- L'étape de calcul de l'aire de la pièce : le calcul de l'aire permet de définir 3 classes :
  - ° la classe A de petites pièces. (1 et 2 dinars).
  - ° la classe B de pièces moyennes.(5 dinars dernière version).
  - ° la classe C de grandes pièces.(5, 10, 20 et 50 dinars).

La classe B comporte un seul type de pièce et donc ne nécessite pas une autre étape de discrimination.

La classe A comporte 2 pièçes et la discrimination se fera à base des motifs.

La classe C comporte 4 pièces et la discrimination se fera à base du critère de la forme du corps (polygonale ou circulaire) et le critère de couleur.

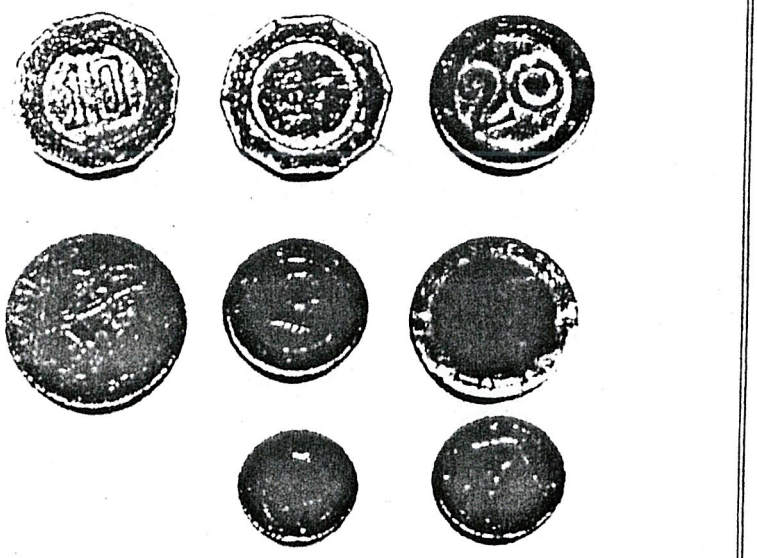


Fig V.2 : Classe d'objets à trier

Conformément à ce qui était présenté au chapitre I, l'allocation (*Mapping*) suit une structure PPF. L'organigramme de la version séquentielle de l'algorithme est donnée sur la Figure V.3. Sa modélisation par un graphe de dépendance(GD) est montrée sur la Figure V.4.

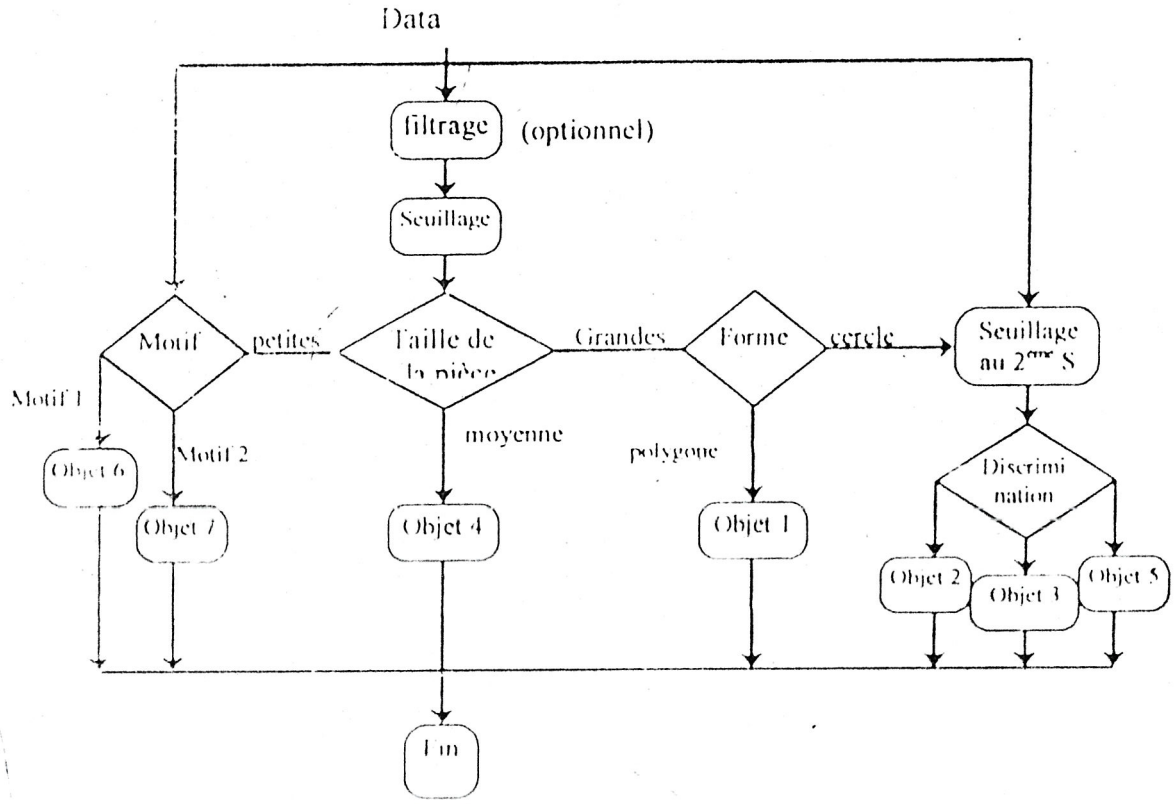


Fig V.3 : Organigramme de la procédure de tri .

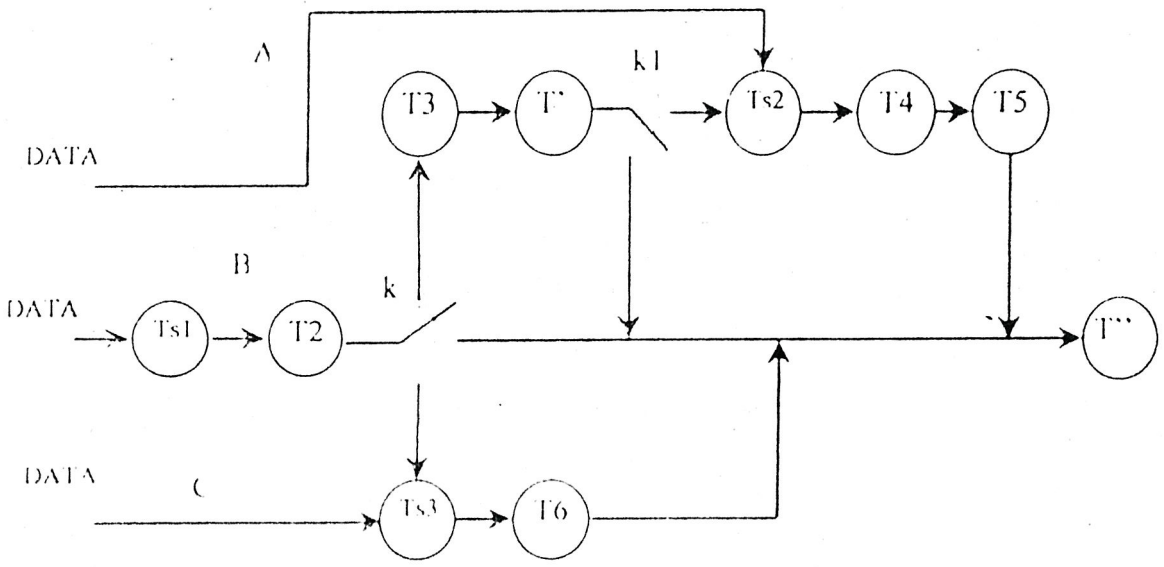


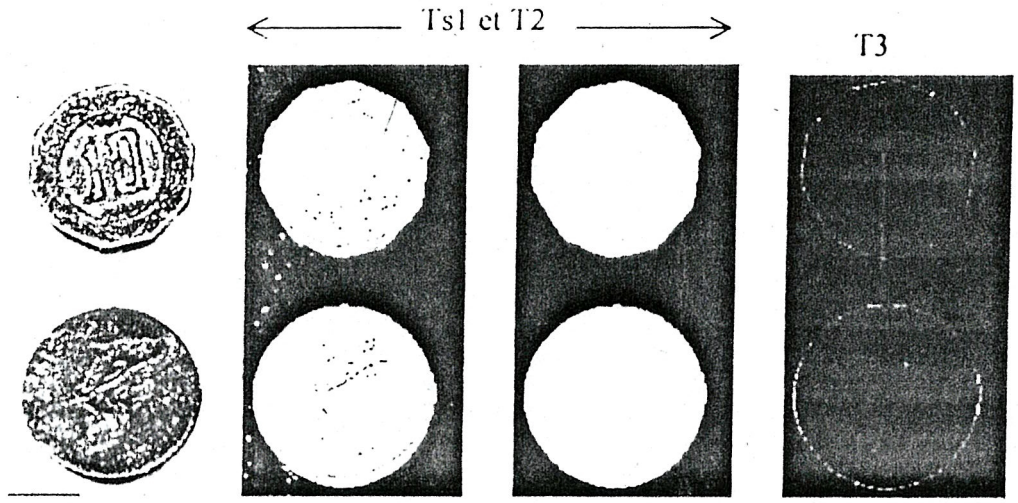
Fig V.4 : Modélisation de l'algorithme par un graphe.

Dans le graphe les cercles représentent des nœuds où seront exécutées les taches. les flèches quant à elles montrent l'acheminement des données.
'k' est un commutateur qui modélise l'aspect conditionnel ' Si '. Selon le résultat requis au niveau du nœud T , une seule branche est active : la branche médiane ou la branche supérieure ou la branche inférieure. Le même principe s'applique pour le commutateurs k1. Ce mode de représentation est choisi de manière à se rapprocher des graphes adoptés dans les outils d'aide au développement d'applications parallèles [32].
Les taches mentionnées au niveau des nœuds sont désignées comme suit :
La tache Ts1 : Premier seuillage. (discrimination de la pièce par rapport à l'arrière plan. Figures V.5.b et V.5.c).
La tache T2 : tache de calcul de l'aire de la pièce. La valeur de cette aire permet de choisir les taches qui suivent :

La tache T3 : est une détection de contour par morphologie (soustraction de l érodé de l'image binaire de l'image binaire elle même FigureV.5.d).
La tache T' : est le processus de calcul du centre de gravité et de la fonction de distribution Fd des points du contour par rapport au centre de gravité. Selon cette distribution, le processus se termine par la tache (T'') (décision finale), ou continue avec une autre suite de processus (Ts2, T4, T5):
La tache Ts2 : est un seuillage permettant la discrimination de la couleur orée. Le chemin A signifie que le processus utilise les données de l'image initiale(l'image en niveau de gris).
La tache T4 : est un ensemble de traitements morphologiques en vue d'une bonne mise en forme des zones segmentées(voir Figures V.7. b et c).
La tache T5 : est le processus de calcul de la fonction de distribution Fd2 des points des zones segmentées par rapport au centre de gravité.
La tache Ts3 : est un seuillage permettant la préparation à l'étape de discrimination de motifs.
La tache T6 est le processus d'extraction de motif.

La tache T'' : est la tache final dans le tri. Selon les résultats précédents, on établi les décision ; et les actions nécessaires.

Les resultats des différents traitements décrits dans l'algorithme sont représentés sur les figures V.5, V.6, V.7 et V.8 :



- a - - b - - c - - d -

Fig V.5 : progression du traitement
- a image d'entrée. c fermeture.
- b binarisation. d – détection de contour par morphologie.

La fonction Fd (la distribution des points de la circonférence par rapport au centre de gravité) pour chaque image de Fig V.5.d est donnée ci-dessous :

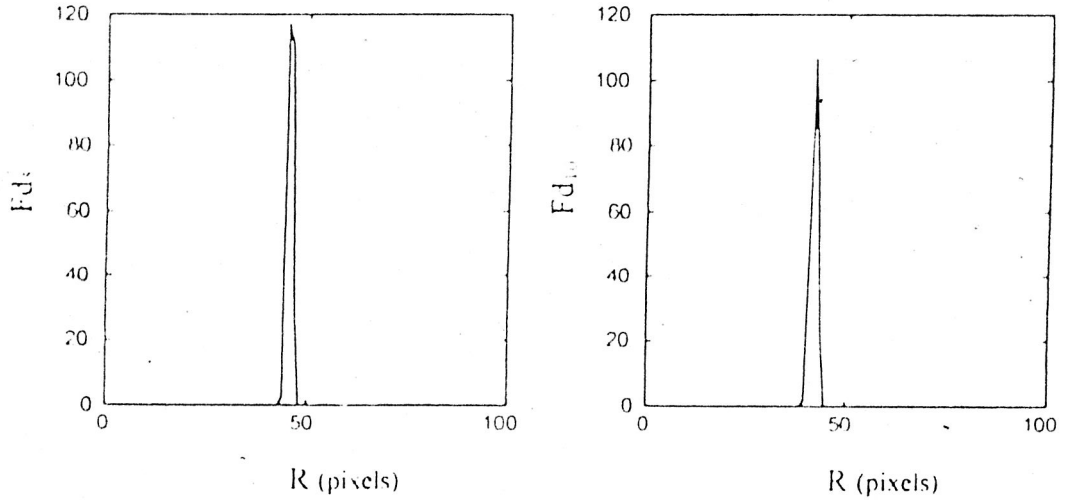


Fig V.6 : fonctions de distribution des points des circonférences par rapport au centre de gravité : Fd_5 pour la forme circulaire, Fd_{10} pour le polygone.

La discrimination des pièces de 5, 50, 20 dinars est établie à la base du critère de la couleur. Si le corps est à une seule couleur (argentée), on est face à une pièce de 5 dinars ; Sinon, si celui ci est à deux couleurs (argentée et orée) alors il s'agit de :
- Pièce de 20 dinars si le résultat de la binarisation est en forme de bague.
- Pièce de 50 dinars si le résultat de la binarisation est en forme de disque.

Voir figure V.7.

La tache (Ts2) est un seuillage avec un niveau de seuil correspondants à la couleur orée suivi d'un calcul de la fonction de distribution (T5) cette fois ci d'une zone homogène par rapport au centre de gravité.
Le centre de gravité est déterminé selon les expressions suivantes :

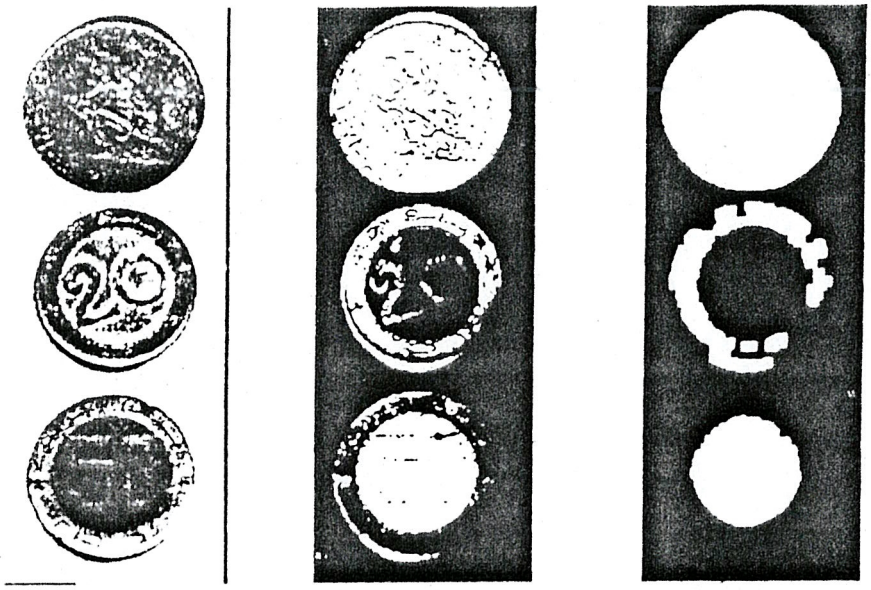
$$x_c = \frac{M_x^1}{M^0} \qquad y_c = \frac{M_y^1}{M^0}$$

ou x_c, y_c sont les coordonnées du centre de gravité, M^0 le moment d'ordre zéro et M_x^1, M_y^1 les moments d'ordre un par rapport à x et y respectivement.

La fonction de distribution (de la zone segmentée) par rapport au centre de gravité est une fonction de (R), où R est l'écart d'un point de la zone par rapport au centre de gravité et est donné par :

$$R = \sqrt{(x_g - i)^2 + (y_g - j)^2}$$

avec x_g, y_g : les coordonnées du centre de gravité.
i , j : les coordonnées d'un point appartenant à la zone considérée.



-a- -b- -c-

Fig V.7 : localisation de la couleur argentée.
a – image d'entrée.
b – seuillage.
c – résultat obtenu après une séquence de traitement morphologiques (fermeture, érosion, ouverture, dilatation.....)

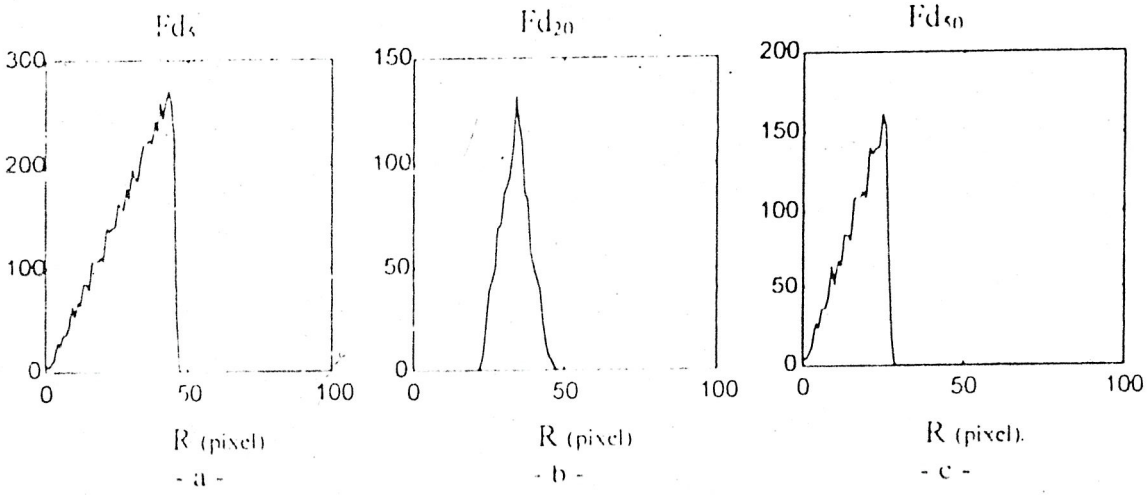


Fig V.8 a, b, c : distributions de l'écart des points de la zone argentée par rapport au centre de gravité Respectivement pour les pièce de 5, 20 et50 dinars.

On remarque bien que la fonction de distribution pour un disque (5 ou 50 dinars) est en forme de dent de scie(figures V.8.a et V.8.c). Le disque le plus grand (5 dinars) a la distribution la plus large et la plus haute(figureV.8.a). Alors que pour la forme en bague (20 dinars) la fonction ne prend de valeur que pour un abscisse R supérieur à 25 (R>25). (figureV.8.b).

La transformation du graphe précédent (Figure V.4) en une structure PPF peut être représentée comme suit :

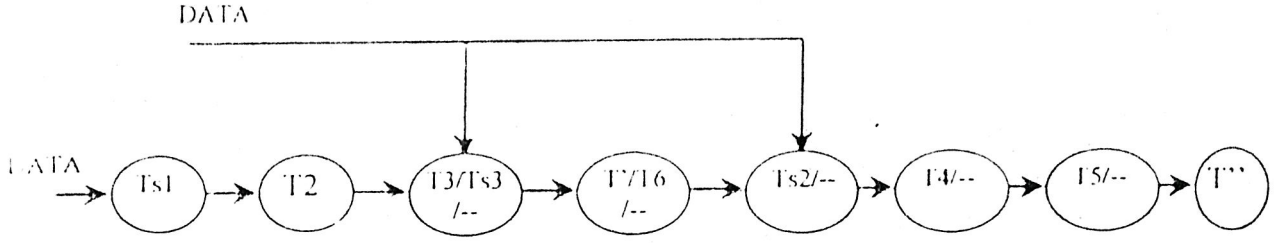


Fig V.9 : Transformation du graphe V.4 en une structure PPF.
(Fusion de la structure conditionnelle).

On constate qu'en allant du troisième nœud (cercle) et jusqu'au septième les taches à exécuter dépendent des résultats du deuxième nœud. La spécification T3/Ts3/-- par exemple dans le nœud 3 signifie que c'est T3 ou Ts3 qui est active, ou il s'agit d'un simple passage de données(--).
Cette structure peut être aisément reportée sur un pipeline de processeurs. On peut aussi accroître la granularité du pipeline. La tache T4 par exemple peut être décomposée en 2 ou 3 traitements morphologiques (érosion, ouverture,...).

On constate donc, dans ce cas qu'on a pas beaucoup de parallélisme algorithmique, toutes les taches s'exécutent en séquence. Le parallélisme qui peut avoir lieu est un parallélisme de données au sein de chaque nœud.
Deux approches peuvent être donc envisagées pour la réalisation du parallélisme de données (Figure V.10 et Figure V.11). Dans la première configuration chaque nœud est restructuré en une forme d'arbre. Dans la deuxième, on établis des liaisons directes entre les *workers* des deux nœuds adjacents en vue de minimiser les transferts de données.

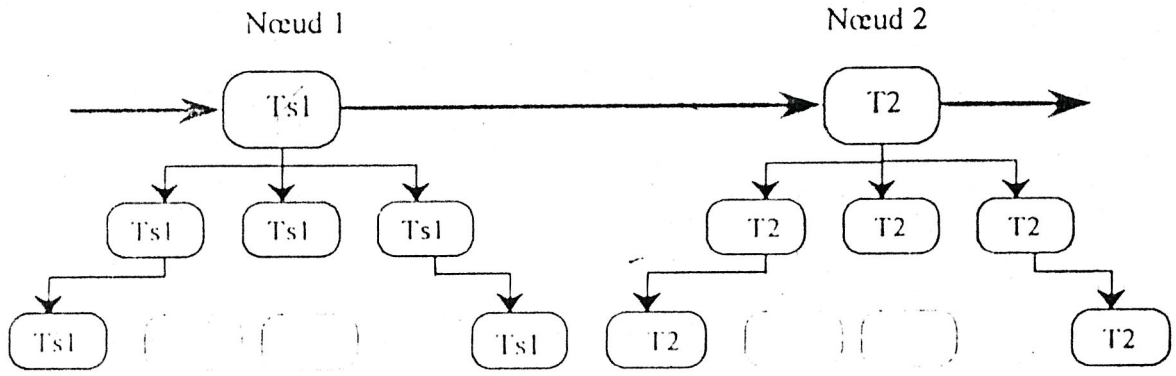


Fig V.10 : Nœud 1 et 2 ont une structure d'arbre .

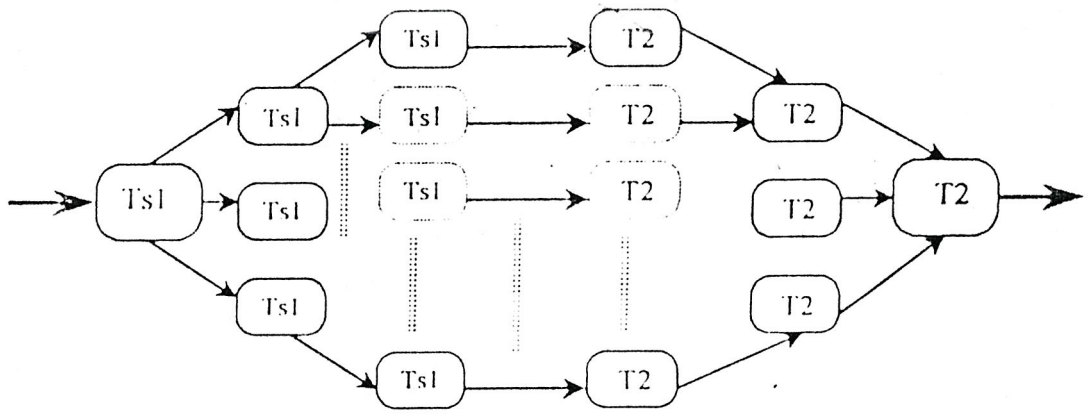


Fig V.11 : Etablissement des liaisons entre le nœud 1 et le nœud 2

En vue de mettre en évidence les temps de calculs effectués sur les deux configurations, les deux premières taches ont été implantées sur deux nœuds à 3 transputers *workers* chacun.(voir Figure V.12 et Fig V.13).

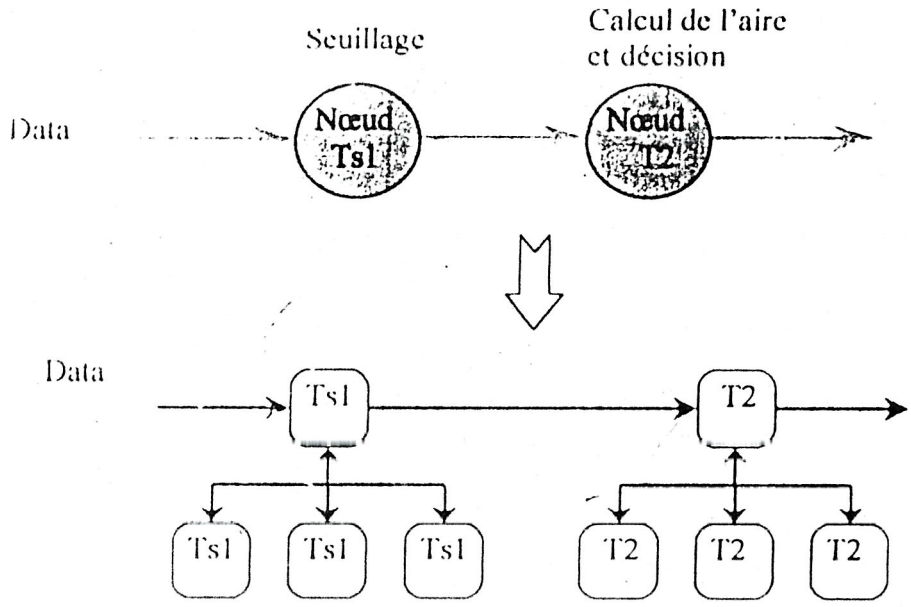


Fig V.12 : Nœud 1 et nœud 2 ont une structure d'arbre. (configuration 1).

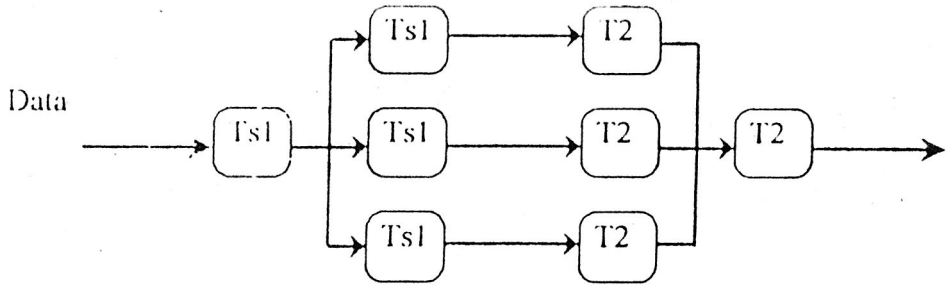


Fig V.13 : Etablissement des liaisons entre le nœud 1 et le nœud 2. (configuration 2).

Les résultats obtenus sont donnés ci dessous :

Configuration	Timing du 1er traitement Ts1		Timing du 2ème traitement T2		Timing totale
Configuration 1	Time.all	Time.wrk	Time.all	Time.wrk	
	133.82 ms	95.49 ms	84.35 ms	59.65 ms	***256.17 ms***
Configuration 2	Time.wrk		Time.wrk		***177.47 ms***
	93.7 ms		58 ms		

On voit que la deuxième configuration est plus rapide que la première. Ceci s'explique par le fait que dans la deuxième organisation le retour des données au transputer (serveur) est évité et les données transitent directement via les *workers* des deux noeuds.

V.3 Implantation de la procédure de tri sur un pipeline de trois transputers :

C'est donc la deuxième configuration qui est adoptée dans le *mapping* général de l'application utilisant trois transputers qui se présente comme suit :
On fragmente le pipeline en trois partitions. Chaque partition est attribuée à un transputer de manière à avoir l'allocation des taches suivant la figure V.14.

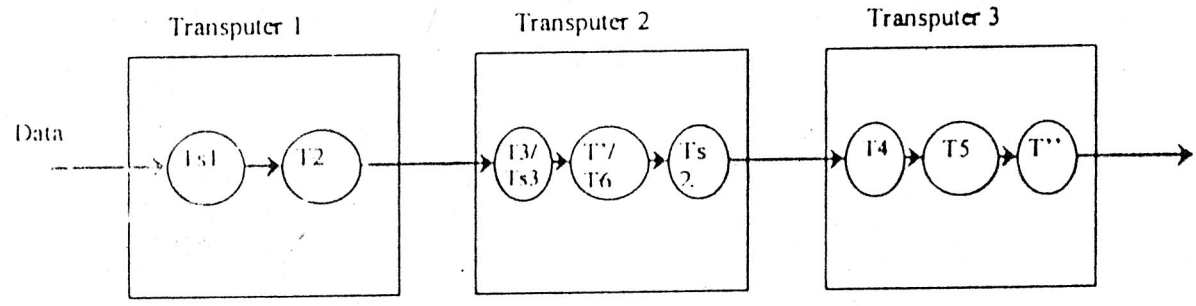


Fig V.14 : Mapping de l'application sur trois transputers.

On a adopté ici (pour l'écriture du programme) le plus long chemin qui est la branche supérieure. Les taches considérées sont donc : Ts1, T2, T3, T', Ts2, T4, T5, T''.
En outre, on établie un parallélisme de données selon la configuration étudiée précédemment (Fig V.13) de manière logicielle (car un seul transputer permet de simuler l'exécution de plusieurs processus parallèles).
Le parallélisme implanté se présente comme suit :

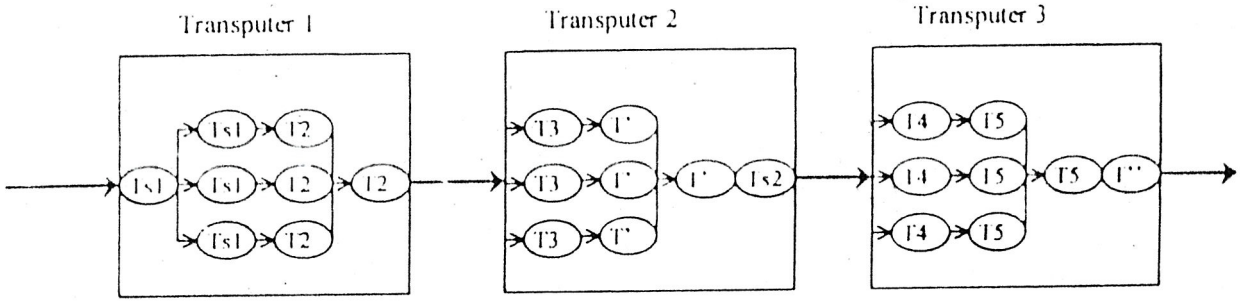


Fig V.15 : Mapping de l'application sur trois transputers avec parallélisme simulé au sein de chaque transputer.

Les routines écrites en OCCAM pour les trois transputers sont données ci-dessous :

1. Sur le premier transputer :

```
---- Déclarations ----
VAL N IS 512:
VAL nbr.line IS 256:
VAL nbr.col IS 256:
VAL nbr.p IS 3:
VAL nbr.ln.b IS ROUND((nbr.line)/nbr.p):
PROTOCOL flot IS INT;INT;[nbr.ln.b][nbr.col]BYTE:
---- Liens physiques ----
CHAN OF flot hst.out1,hst.in1,hst.out2,hst.in2:
PLACE hst.out1 AT 1:      ----link1.out  is 1:
PLACE hst.in1  AT 5:      ----link1.in   is 5:
PLACE hst.out2 AT 3:      ----link2.out  is 2:
PLACE hst.in2  AT 7:      ----link2.in   is 6:
---- Procédures (Ts1,T2,T) ----
... Procédure threshold.
... Procédure dilate
... Procédure erode
... Procédure superficie évaluation.
---- Liens logiciels (locales) ----
[3]CHAN OF flot loc.lk.in,loc.lk.out:
[3]CHAN OF flot loc.out.sec:
[3]CHAN OF flot loc.out.ther:
[3]CHAN OF flot loc.out.forth:
---- routine parallèle ----
[3]INT S,nm.b:
INT delais,time.on,time.off,bloc.received,A:
SEQ
  bloc.received:=00
  delais:=00(INT)
  clock ? time.on
  PAR
    PAR i=0 FOR 3
      loc.lk.in[i] ! i;delais;[image FROM (nbr.ln.b*i)+1 FOR
                                nbr.ln.b]
    PAR i=0 FOR 3
      threshold(loc.lk.in[i],loc.lk.out[i])
    PAR i=0 FOR 3
      dilate(loc.lk.out[i],loc.out.sec[i])
    PAR i=0 FOR 3
      erode(loc.out.sec[i],loc.out.ther[i])
    PAR i=0 FOR 3
      superf.eval(loc.out.ther[i],loc.out.forth[i])
    WHILE bloc.received<3
      PRI ALT i=0 FOR 3
```

```
        loc.out.forth[i] ? nm.b[i];S[i];[image.result FROM
                                    (nbr.ln.b*i)+1 FOR nbr.ln.b]
          SEQ
            bloc.received :=bloc.received + 1
            A := A + S[i]
  SEQ i=0 FOR 3
    Hst.out1 ! nm.b[i];A;[image.result FROM
                                    (nbr.ln.b*i)+1 FOR nbr.ln.b]
  clock ? time.off
  timing()
```

---

2. la routine de la configuration se présente comme suit :

```
---- configuration ----
CHAN OF flot from.hst.to.wrk1, from.wrk1.to.wrk2:
CHAN OF flot to.hst.from.wrk1, to.wrk1.from.wrk2:
PLACED PAR
  PROCESSOR 0 T8
    PLACE from.hst.to.wrk1 AT link3.in : --7
    PLACE from.wrk1.to.wrk2 AT link2.out : --2
    PLACE to.wrk1.from.wrk2 AT link2.in : --6
    PLACE to.hst.from.wrk1 AT link3.out : --3
    Node1(from.hst.to.wrk1, from.wrk1,to.wrk2)
  PROCESSOR 1 T8
    PLACE from.wrk1.to.wrk2 AT link1.in : --5
    PLACE to.wrk1.from.wrk2 AT link1.out : --1
    PLACE to.hst.from.wrk2 AT link0.out : --0
    PLACE from.hst.to.wrk2 AT link0.in : --4
    Node2(from.wrk1,to.wrk2, to.hst.from.wrk2)
```

Les procédures Node1 et Node2 sont décrites comme suit :

i. Nœud 1 :

```
VAL nbr.line IS 256:
VAL nbr.col IS 256:
VAL nbr.p IS 3:
VAL nbr.ln.b IS ROUND((nbr.line)/nbr.p):
PROTOCOL flot IS INT;INT;[nbr.ln.b][nbr.col]BYTE:
```

```
Node1(CHAN OF flot lk.in, lk.out)
---- Déclarations ----
INT time.wrk.on, time.wrk.off, delais:
[3]INT Fd,nm.b:
TIMER clock.wrk:
[nbr.line][nbr.col]BYTE data,data.result:
INT bloc.received:

---- Liens logiciels (locales) ----
[3]CHAN OF flot loc.lk.in,loc.lk.out:
[3]CHAN OF flot loc.out.sec:
[3]CHAN OF flot loc.out.ther:

... Procédure edge.détection
... Procédure centre.gravité.evaluation
... Procédure Fd.distribution

SEQ
  WHILE bloc.received<3
    ALT
      lk.in ? nmr.b;delais;[ data FROM (nbr.ln.b*i)+1
                             FOR nbr.ln.b]
        SEQ
          bloc.received :=bloc.received + 1
  bloc.received00-: (INT)
  clock.wrk ? time.wrk.on
  PAR
    PAR i=0 FOR 3
      loc.lk.in[i] ! i;delais;[data FROM (nbr.ln.b*i)+1 FOR
                               nbr.ln.b]
    PAR i=0 FOR 3
      edge.detection(loc.lk.in[i],loc.lk.out[i])
    PAR i=0 FOR 3
      Centre.gravité.éval(loc.lk.out[i],loc.out.sec[i])
    PAR i=0 FOR 3
      Fd.distribution(loc.out.sec[i],loc.out.ther[i])
    WHILE bloc.received<3
      PRi ALT i=0 FOR 3
        loc.out.ther[i] ? nm.b[i];Fd[i];[data.result FROM
                                 (nbr.ln.b*i)+1 FOR nbr.ln.b]
          SEQ
            bloc.received :=bloc.received + 1

  clock ? time.wrk.off
  SEQ i=0 FOR 3
```

```
lk.out.1 ! nm.b[i];Fd[i];[data.result FROM
                                (nbr.ln.b*i)+1 FOR nbr.ln.b]
```

ii. Nœud 2 :

```
VAL nbr.line IS 256:
VAL nbr.col IS 256:
VAL nbr.p IS 3:
VAL nbr.ln.b IS ROUND((nbr.line)/nbr.p):
PROTOCOL flot IS INT;INT;[nbr.ln.b][nbr.col]BYTE:

Node2(CHAN OF flot lk.in, lk.out)
---- Déclarations ----
INT time.wrk.on, time.wrk.off, delais:
[3]INT Fd2,nm.b:
TIMER clock.wrk:
[nbr.line][nbr.col]BYTE data,data.result:
INT bloc.received:
---- Liens logiciels (locales) ----
[3]CHAN OF flot loc.lk.in,loc.lk.out:
[3]CHAN OF flot loc.out.sec:
[3]CHAN OF flot loc.out.ther:
[3]CHAN OF flot loc.out.forth:


... Procédure threshold
... Procédure close
... Procédure centre.gravité.évaluationure
... Procédure Fd.distribution
SEQ
  WHILE bloc.received<3
    ALT
      lk.in ? nmr.b;delais;[ data FROM (nbr.ln.b*i)+1
                                FOR nbr.ln.b]
        SEQ
          bloc.received :=bloc.received + 1
 bloc.received00-: (INT)
 clock.wrk ? time.wrk.on
 PAR
   PAR i=0 FOR 3
```

```
      loc.lk.in[i] ! i;delais;[data FROM (nbr.ln.b*i)+1 FOR
                                  nbr.ln.b]
    PAR i=0 FOR 3
      threshold(loc.lk.in[i],loc.lk.out[i])
    PAR i=0 FOR 3
      close(loc.lk.out[i],loc.out.sec[i])
    PAR i=0 FOR 3
      Centre.gravité.éval(loc.out.sec[i],loc.out.ther[i])
    PAR i=0 FOR 3
      Fd.distribution(loc.out.ther[i],loc.out.forth[i])
    WHILE bloc.received<3
      PRI ALT i=0 FOR 3
        loc.out.forth[i] ? nm.b[i];Fd[i];[data.result FROM
                                  (nbr.ln.b*i)+1 FOR nbr.ln.b]
          SEQ
            bloc.received :=bloc.received + 1
  clock ? time.wrk.off
  SEQ i=0 FOR 3
    lk.out.l ! nm.b[i];Fd[i];[data.result FROM
                                  (nbr.ln.b*i)+1 FOR nbr.ln.b]
```

Exemple : programme de filtrage morphologique sur un réseau de 6 workers :

dans ce qui suit est donné un programme exemple montrant l'aspect de la configuration, l'allocation, la distribution des taches et la gestion des liens de communication avec le langage OCCAM. Il s'agit d'un programme de filtrage morphologique d'une image en niveau de gris de 256x256 de résolution, exécuté sur un réseau de 7 transputers dont un est le transputer maître arrangés en une structure d'arbre (voir . §IV.3).
Un programme ''réseau'' comme déjà signalé est composé d'un module EXE et un module PROGRAM les deux en plies (Folds) tel que :

. . . EXE Procédures d'interfaçage et de distribution de données.
. . . PROGRAM procédures de traitement et de routage sur réseau.

Les titres venant après EXE et PROGRAM sont des commentaires arbitraires expliquant le contenu de ces modules.
Dans cet exemple le module EXE comprend une partie déclaration de variables, des procédures d'acquisition et des procédures d'E/S de données et ainsi qu'une procédure de distribution de données au réseau désignée ici par Dispatch.
Le contenu du module EXE est le suivant :

```
((( EXE procédures :interfaçage et distribution de données

#USE userhdr
#USE derivio
#USE userio
#USE msdos
#USE krnlhdr
#USE afiler

VAL nbr.p IS 6:
VAL nbr.line IS 256:
VAL nbr.col IS 256:
VAL nbr.ln.b IS ROUND(nbr.line/nbr.p):
[nbr.line][nbr.col]BYTE image,image.result:
[nbr.ln.b][nbr.col]BYTE buffer:
TIMER clock:
INT time.on, time.off, time :
[6]INT time.wrk:
INT char, delais,nbr:
PROTOCOL flot IS INT;INT;[nbr.ln.b][nbr.col]BYTE :
CHAN OF flot hst.out1,hst.in1,hst.out2,hst.in2:
PLACE hst.out1 AT link1.out:
PLACE hst.in1  AT link1.in:
PLACE hst.out2 AT link3.out:
PLACE hst.in2  AT link3.in:
```

... PROCEDURE1 Acquisition
... PROCEDURE2 E/S
... PROCEDURE3 Timing

... PROCEDURE1 Dispatch

}}}

le fold Dispatch comprend ce qui suit :

```
{{{ dispatch
INT nbr.block.received:
INT nmr.b,tm.wrk:
SEQ
  delais:=00(INT)
  nbr.block.received:=00(INT)
  SEQ i=0 FOR nbr.line
    SEQ j=0 FOR nbr.col
      IF
        image[i][j] > threshold(BYTE)
          image[i][j]:= 01(BYTE)
        TRUE
          image[i][j]:= 00(BYTE)

  clock ? time.on
  SEQ i=0 FOR 3
    nmr.b:=i
    PAR
      hst.out1 ! nmr.b;delais;[image FROM (nbr.ln.b*i) FOR
                                nbr.ln.b]
      hst.out2 ! nmr.b;delais;[image FROM (nbr.ln.b*(i+3))
                                FOR nbr.ln.b]

  WHILE nbr.block.received < nbr.p
    ALT
      hst.in1 ? nmr.b;tm.wrk;[buffer FROM 0 FOR nbr.ln.b]
        SEQ
          nbr.block.received:= nbr.block.received + 1
          time.wrk[nmr.b] :=tm.wrk
          [image.result FROM nmr.b*nbr.ln.b FOR nbr.ln.b] :=
                    [buffer FROM 0 FOR nbr.ln.b]
      hst.in2 ? nmr.b;tm.wrk;[buffer FROM 0 FOR nbr.ln.b]
        SEQ
          nbr.block.received:= nbr.block.received + 1
          time.wrk[nmr.b] :=tm.wrk
          [image.result FROM nmr.b*nbr.ln.b FOR nbr.ln.b] :=
                    [buffer FROM 0 FOR nbr.ln.b]
```

```
  clock ? time.off
  SEQ i=1 FOR (6*nbr.ln.b)-2
    SEQ j=1 FOR nbr.col-2
      IF
        (INT image.result[i][j]) < 2
          image.result[i][j]:=BYTE(
                              (INTimage.result[i][j])*255)

}}}
```

Le module PROGRAM comprend deux modules SC : SC root.skel procédure et SC skel procédure et le *fold* configuration où est spécifié la configuration. Le premier module SC est à charger au niveau des deux ''routeurs'' (worker1,worker4), alors que le deuxième est à charger au niveau des workers restants : worker2,worker3,worker5,worker6 . (voir structure arbre §IV.3).

```
{{{ PROGRAM traitement et routage
VAL nbr.p IS 6:
VAL nbr.line IS 256:
VAL nbr.col IS 256:
VAL nbr.ln.b IS ROUND(nbr.line/nbr.p):
PROTOCOL flot IS INT;INT;[nbr.ln.b][nbr.col]BYTE :
... SC root.skel procedure
... SC skel procedure

... configuration
}}}
```

{{{ root.skel procédure

```
PROC root.skel(CHAN OF flot lk.wrk.in, lk.to.w2,lk.to.w3,
                            lk.from.w2,lk.from.w3,lk.wrk.out)
  CHAN OF flot lk.loc.in,lk.loc.out:
  INT time.wrk.on, time.wrk.off, tm.w,
      delais,delais2,nmr.b1,nmr.b2 :
  TIMER clock.wrk:
  [nbr.ln.b][nbr.col]BYTE data,data.result:
  INT b.received,index:
  [512]INT lut:

  SEQ
    b.received:=00(INT)
    PAR
      SEQ
        lk.loc.out ? nmr.b1;delais;[data.result FROM 0 FOR
                                    nbr.ln.b]
        clock.wrk ? time.wrk.on
        SEQ i=1 FOR nbr.ln.b-2
          SEQ j=1 FOR nbr.col-2
```

```
        SEQ
          index := (((INT data.result[i-1][j-1]) +
                    (2 * (INT data.result[i-1][j]))) +
                    (4*(INT data.result[i-1][j+1]))) +
                    (8 * (INT data.result[i][j-1]))
          index := ((index + (16*(INT
                     data.result[i][j]))) +
                    (32 * (INT data.result[i][j+1]))) +
                   (64*(INT data.result[i+1][j-1]))
          index := (index + (128 * (INT
                    data.result[i+1][j]))) +
                   (256*(INT data.result[i+1][j+1]))

        data.result[i][j]:=BYTE (lut[index])
    clock.wrk ? time.wrk.off
    tm.w := time.wrk.off MINUS time.wrk.on
    lk.loc.in ! nmr.b1;tm.w;[data.result FROM 0 FOR
                                nbr.ln.b]
SEQ
  WHILE (b.received<6)    ---- AND(b.returned<1)
    ALT
      lk.wrk.in ? nmr.b2;delais2;[data FROM 0 FOR
                                    nbr.ln.b]
        SEQ
          IF
            nmr.b2=2 OR nmr.b2=5
              SEQ
                lk.loc.out ! nmr.b2;delais2;[data FROM
                                        0 FOR nbr.ln.b]
                b.received:=b.received+1
            nmr.b2=2 OR nmr.b2=4
              SEQ
                lk.to.w2 ! nmr.b2;delais2;[data FROM 0
                                        FOR nbr.ln.b]
                b.received:=b.received+1
            nmr.b2=0 OR nmr.b2=3
              SEQ
                lk.to.w3 ! nmr.b2;delais2;[data FROM
                                        0 FOR nbr.ln.b]
                b.received:=b.received+1
            TRUE
              SKIP
      lk.from.w2 ? nmr.b2;delais2;[data FROM 0 FOR
                                    nbr.ln.b]
        SEQ
          lk.wrk.out ! nmr.b2;delais2;[data FROM 0 FOR
                                        nbr.ln.b]
          b.received:=b.received+1
      lk.from.w3 ? nmr.b2;delais2;[data FROM 0 FOR
                                    nbr.ln.b]
        SEQ
```

```
                lk.wrk.out ! nmr.b2;delais2;[data FROM 0 FOR
                                                nbr.ln.b]
              b.received:=b.received+1
            lk.loc.in ? nmr.b2;delais2;[data FROM 0 FOR
                                            nbr.ln.b]
            SEQ
              b.received:=b.received+1
              lk.wrk.out ! nmr.b2;delais2;[data FROM 0 FOR
                                                nbr.ln.b]
:
}}}
```

{{{ skel procédure

```
VAL nbr.p IS 6:
VAL nbr.line IS 256:
VAL nbr.col IS 256:
VAL nbr.ln.b IS ROUND(nbr.line/nbr.p):

PROTOCOL flot IS INT;INT;[nbr.ln.b][nbr.col]BYTE:

PROC skel(CHAN OF flot lk.wrk.in,lk.wrk.out)
  INT time.wrk.on, time.wrk.off, tm.w, delais,nmr.b :
  TIMER clock.wrk:
  [nbr.ln.b][nbr.col]BYTE data,data.result:
  INT index:
  [512]INT lut:
  SEQ
    lk.wrk.in ? nmr.b;delais;[data FROM 0 FOR nbr.ln.b]
    SEQ
      clock.wrk ? time.wrk.on
      SEQ i=1 FOR nbr.ln.b-2
        SEQ j=1 FOR nbr.col-2
          SEQ
            index := (((INT data[i-1][j-1]) +
                      (2 * (INT data[i-1][j]))) +
                      (4*(INT data[i-1][j+1]))) +
                      (8 * (INT data[i][j-1]))
            index := ((index + (16*(INT data[i][j]))) +
                      (32 * (INT data[i][j+1]))) +
                      (64*(INT data[i+1][j-1]))
            index := (index + (128 * (INT data[i+1][j]))) +
                       (256*(INT data[i+1][j+1]))
            data.result[i][j]:=BYTE (lut[index])
      clock.wrk ? time.wrk.off
      tm.w := time.wrk.off MINUS time.wrk.on
      lk.wrk.out ! nmr.b;tm.w;[data.result FROM 0 FOR
                                  nbr.ln.b]
:
}}}
```

{{{configuration

```
PROTOCOL flot IS INT;INT;[nbr.ln.b][nbr.col]BYTE:
[2]CHAN OF flot from.hst, to.hst :
[2]CHAN OF flot to.up, from.up, to.down, from.down:

PLACED PAR i=0 FOR 2
  PROCESSOR 3*i T8
    PLACE from.hst[i] AT 7:        ----link3.in is 7:
    PLACE to.up[i] AT 2:           ----link2.out is 2:
    PLACE from.up[i] AT 6:         ----link2.in is 6:
    PLACE to.down[i] AT 1:         ----link1.out is 1:
    PLACE from.down[i] AT 5:       ----link1.in is 5:
    PLACE to.hst[i] AT 3:          ----link3.out is 3:
    root.skel(from.hst[i], to.up[i], to.down[i], from.up[i],
              from.down[i], to.hst[i])
    PLACED PAR
      PROCESSOR i+1 T8
        PLACE to.up[i] AT 6:          ----link2.out is 2:
        PLACE from.up[i] AT 2:        ----link2.in is 6:
        skel(to.up[i],from.up[i])
      PROCESSOR i+2 T8
        PLACE to.down[i] AT 5:          ----link2.out is 2:
        PLACE from.down[i] AT 1:        ----link2.in is 6:
        skel(to.down[i],from.down[i])

}}}
```

• Conclusion

Dans ce travail nous avons analysé les performances d'un réseau de transputers. Nous avons mesuré des temps d'exécution d'un ensemble de traitements d'images bas niveau sur une configuration en arbre. Ceci a été réalisé en deux étapes : Dans la première étape on a prélevé des mesures pour des exécutions sur 2, 3 transputers sans faire de découplage entre les transferts et les traitements. Les résultats obtenus s'approchaient des calculs théoriques pour la majorité des traitements. (On a obtenu des efficacités de E% = 98 et E% = 99 pour le filtrage et la détection de contours). Donc le parallélisme s'est avéré efficace en utilisant un nombre réduit de transputers et la réalisation de découplage entre transferts et traitements ne s'est avéré pas nécessaire.
Dans la seconde étape on a simulé la parallélisation des traitements sur 6, 8, 9, et 12 transputers en utilisant 3 transputers physiques. Donc les transferts de données et les traitements ont été considérés totalement découplés. Les rapports d'efficacité obtenus étaient légèrement inférieurs à ceux obtenus dans la première étape car les *overheads* ont augmenté en augmentant le nombre de processeurs. Toutefois les rapports d'accélération sont restés relativement bons pour les traitements à taux de calculs élevés. (E% = 98, E% = 95 pour la détection de contours et le filtrage médian).
Comme exemple d'application de vision industrielle, on a traité le problème de tri de pièces de monnaie algériennes. On a constaté que la procédure renfermait plus de potentiel intrinsèque séquentiel. Donc, son *Mapping* était réalisé selon une structure PPF. On a envisagé deux schémas pour établir le parallélisme de données. Le premier consistait à la restructuration de chaque nœud du PPF en arbre. Dans le deuxième, les structures en arbre ont été inclinées de manière à ce que le passage des données s'effectue directement à travers les workers de deux nœuds adjacents. Il s'est avéré que le deuxième schéma était plus performant, car il engendrait moins d'*overheads*.
Nous avons conclu donc que les mesures relevés renseignaient amplement sur les performances des systèmes à transputers et des systèmes similaires (Réseaux de DSP). Les points que nous suggérons pour une étude ultérieure peuvent être extraits de l'exemple d'application de tri et concernent l'étude de la communication interprocesseurs dans le contexte de l'Adéquation Algorithme Architecture AAA.

BIBLIOGRAPHIE:

1. Traitement de l'image sur micro-ordinateur.
   Jean Jacques TOUMAZET . SYBEX.
2. Vision par ordinateur ' outils fondamentaux ' .
   Radu Horaud _ Olivier Monga . HERMES.
3. traitement numérique du signal .
   M. Bellanger . Masson .
4. Digital image processing.
   Gregory A. Baxes Wiley.
5. Digital image processing.
   R. Castleman
6. Architecture des ordinateurs.
   John L. Henessy - David A. Patterson. Ediscience.
7. Les multiprocesseurs.
   Daniel Tabak.
8. La programmation parallèle.
   J.P. Banatre
9. Principes des systèmes d'exploitation.
   A. Silbershatz
10. Real time systems.
    B. Baley
11. The transputer applications notebook. Systems and performance.
    INMOS.
12. The transputer applications notebook. Architecture and software.
    INMOS.
13. Transputer development system. Delivery manual.
    Prentice hall.
14. A tutorial introduction to occam programming.
    Dick pountain. Billings & Sons.
15. Parallélisation d'algorithmes de traitements d'images sur un réseau de Transputers. Thèse de magister. Université Ferhat abbess de Setif.
16. A.C. downton, R.W.S Tregidgo : " Top-down structured parallelisation of embedded image processing " , IEE Proc –vis. Image signal process.,1994.
17. Ficret Ercan M ; Fung Y F ; " Communication in a large DSP array for image processing ".SPIE proceedings series , 1997.
18. B.Schmidt, M.Schimmler " Morphological Hough transform on the instruction Systolic array ". Lecture notes in computer science. 1997.

19. H.Tomiyasu, S.Kusakabe. " Co-processor system design for Fine-Grain message handling in KUMP/D".Lecture notes in computer science.1997.
20. M.Fleury, C.Downton. " Karhunen-loève transform : An exercice in simple image-processing parallel pipelines ". Lecture notes in computer science. 1997.
21. W.C.lau, Y.F.fung :' Communication mechanisms for a heterogeneous image processing' , 1997 IEEE International conferance on intelligent processing systems.
22. V.Getov, E.Hernandez " Message passing performance of parallel computers ". Lecture notes in computer science. 1997.
23. L.Bhuyan, Q.yang. "Performance of multiprocessor interconnection networks". IEEE. 1989.
24. G.Haase " New matrix-by-vector multiplication based on a nonoverlapping domain decomposition data distribution". Lecture notes in computer science. 1997.
25. A.Kiper. " A fully parallel symmetric matrix transformation". Lecture notes in computer science. 1997.
26. M.Hadad. " Parallel Sparse LU decomposition with application to semiconductor device simulation ". Lecture notes in computer science. 1997.
27. T.Rauber, G.Ruger. " Scalability of parallel sparse cholesky factorization". Lecture notes in computer science. 1997.
28. E.Santos. "Optimal parallel algorithms for solving tridiagonal linear systems". Lecture notes in computer science. 1997.
29. L.Haas, F.Yang. " Adéquation de l'algorithme de canny-Deriche généralisé sur architecture DSP avec l'environnement Syndex". Traitement du signal. V 14. 1997.
30. D.Deigny, F.Garcia. " De l'architecture à l'algorithme un exemple : le détecteur de contours de Deriche". Traitement du signal. V 14. 1997.
31. D.Ginhac, J.Serot. " Evaluation de l'outil Syndex en vue de prototypage rapide d'application de traitement d'images sur machine MIMD-DM". Traitement du signal. V 14. 1997.
32. M.Israel, D.Dupont. "De la spécification formelle au partitionnement matériel logiciel". Traitement du signal. V 14. 1997.
33. M.Renaudin, F.Robin. "AAAA : asynchronisme et adéquation algorithme architecture ". Traitement du signal. V 14. 1997.
34. M.Renaudin, F.Robin. "Un réseau cellulaire VLSI fonctionnellement asynchrone pour le filtrage morphologique d'images". Traitement du signal. V 14. 1997.
35. Orlando S ; " A comparison of implementation strategies for nonuniforme data parallel computation " ; Journal of parallel and distributed computing ;1998.

36. Ojeda Guerra N ; " Hardware mapping of a parallel algorithme for matrix-vector multiplication overlaping communication and computation" ; Lecture notes in computer science ;1998.
37. ITO M R ;Duong S ;Mcfee ; " Pipe-lined algorithm and parallel architecture for real-time detection of sparse small objects in images " ; SPIE proceedings series, 1997.
38. Anconaf ; Rowetta S ; " An image recognition system implemented on hierarchical parallel architectures" ; International journal of pattern recognition and artificial intelligence ; 1998.
39. Platzner M ; Peters L ; "Fast signature segmentation on a multi-DSP architecture " ; SPIE proceedings series ;1997.
40. Panos M.pardalos, Mauricio G.C. Resend ' DIMACS : Series In Discrete Mathematics and Theoritical Computer Science', volume 22. 1994.
41. Apostolos Gerasoulis , jia Jiao , ' A multistage approche for scheduling task graphs on parallel machines ' , DIMACS .1994.
42. Per S. Laursen , ' Experience with a synchronous parallel branch and bound algorithm' , DIMACS . 1994.
43. J.M Corridoni , A. Del bimbo ; : 'Automatic video segmentation through editing analysis'. IEEE transactions on pattern analysis and machine intelligence,.
44. Luc Vincent, Pierre Soile ; :' Watersheds in digital spaces : an efficient algorithme based on immersion simulation ', IEEE transactions on pattern analysis and machine intelligence,1991.
45. H. Digabel, C. Lantuéjoul ; :" Iterative algorithme " ; 2nd European symp Quantitative analysis of microstructures in mateial science,Riederer Verlag 1978.
46. S Beucher , C Lantuéjoul ; " Use of watersheds in contour detection " ; Workshop Image Processing ; Rennes, France 1979.
47. J.P Braclaire ; : 'Image segmentation With topological Maps and inter-pixel representation' , journal of visual communication and image representation,1998.