REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'EDUCATION NATIONALE

ECOLE NATIONALE POLYTECHNIQUE

DEPARTEMENT ELECTRONIQUE

MEMOIRE DE FIN D'ETUDES

POUR L'OBTENTION DU DIPLOME
D'INGENIEUR D'ETAT
OPTION ELECTRONIQUE

THEME

STRUCTURES SYSTOLIQUES ET SIMULATION A L'AIDE DU " C PARALLELE "

DIRIGE PAR:

Mr. S. BOUSBIA
Mr. A. GOUGAM

REALISE PAR:

Melle. N. KHABER

PROMOTION : JUILLET 1994

E. N. P. 10 AVENUE HACEN BADI EL HARRACH ALGER

DEDICACES

A la mémoire de mon grand-père BELAID

A mes très chèrs parents adorés

A mes très chèrs frères Chafik, Chawki, Fouad et Mahdi sans oublier Safa

A mes deux grand-mères

A tous mes oncles particulièrement Lakhdar et tonton Saad-Eddine

A mes deux tantes Fatiha et Hayet

A tous mes cousins, cousines et amis (es)

A tous ceux qui me sont très chèrs

Votre amour, compréhension et soutien moral m'ont été très précieux.

Pour cela je vous serai éternellement reconnaissante.

REMERCIEMENTS

Je tiens à exprimer mes sincères remerciements à:

* Tous mes enseignants qui ont contribué à ma formation.

* Mes deux promoteurs M^R S. BOUSBIA et M^R H. GOUCAME pour m'avoir proposé le sujet et m'assurer de leur encadrement.

* M^R S.A. CHAREF pour son dévouement et tout ce qu'il a fait pour moi, sans oublier MOUNIA. Qu'ils trouvent dans ce mémoire mon estime et ma profonde reconnaissance.

* Mon père pour son aide et ses conseils précieux qu'il m'a prodigués afin de bien achever ce mémoire, sans oublier M^R M. BOUTOUCHENT pour sa grande collaboration.

* Les responsables de l'INELEC qui m'ont facilité l'accès au laboratoire de recherche de l'institut.

Mes sincères remerciements vont également à tous mes amis (es) qui ont participé de prés ou de loin à la réalisation de ce travail.

Mes respects aux membre de jury qui me fera l'honneur d'apprécier mon modeste travail.

ملخص

تمثل الشبكات انقباضية هدف هذه الدراسة التي عرضت فيها خصائصها و مبادئها. و من أجل اعطاء نموذج لهذه الشبكات تمت دراسة البرمجة 'C' المتوازي.

و أخيرا تم ادخال الترنسبيوتر الذي لا يتجزء من 'C' المتوازي مصحوبا ببعض التطبيقات.

RESUME:

Les réseaux systoliques font l'objet de cette étude. Leurs caractéristiques et leurs principes y sont exposés.

Le langage "C Parallèle" a été étudié dans le but de simuler ces réseaux.

Le transputeur indissociable du "C Parallèle" y a été introduit, accompagné de quelques applications.

ABSTRACT:

The systolic Networks are the subject of this study.

Their caracteristics and principales have been exposed.

The langage "C parallel" has been elaborated in order to simulate these networks.

The transputer, infinitely related to "C parallel" has been introduced in this study, togheter with somme applications.

SOMMAIRE

CHAPITRE VI SIMULTION

VI-1 INTRODUCTION

VI-2 Premier programme : producteur consommateur

VI-3 Deuxièmme programme : produit de deux matrices denses

VI-4 Troisième programme : tri des nombres par ordre croissant

CONCLUSIONS

INTRODUCTION

INTRODUCTION:

Pendant longtemps, l'augmentation de la vitesse des calculateurs était le but de tout constructeur. Au début; elle était basée sur l'amélioration des performances des composants en conservant toujours l'architecture séquentielle. Cependant, cette augmentation des performances s'est trouvée aujourd'hui insuffisante aux besoins des utilisateurs. Pour cela, les constructeurs ont adapté une nouvelle manière de penser qui se resume en un mot: Parallélisme.
L'évolution des circuits VLSI (Very large Scale Integration) pendant cette dernière décénnie a permis la concrétisation de la nouvelle notion: parallélisme et ceci en réalisant divers systèmes aussi intéressants les uns que les autres où l'on distingue les réseaux systoliques qui peuvent traiter un trés grands nombres d'opérations et ceci pèndant un laps de temps trés appréciable, de plus, ces réseaux détiennent un rôle trés important dans le domaine des implémentations des algorithmes numériques dans le domaine de traitement d'image et autre.

Notre projet comporte:
Le chapitre I, qui nous introduit dans le domaine des architectures parallèles, où nous donnes une vision sur ces architectures et leur classifications, le chapitre II où nous parlerons des caractéristiques essentielles des réseaux systoliques, le chapitre III qui décrit la conception de ces réseaux où plusieurs exemples d'algorithmes sont donnés. le chapitre IV qui est consacré au transputeur suivi par le chapitre V qui traite le langage "C parallèle" et finalement le chapitre VI qui traite la simulation de deux applications.

CHAPITRE 1

ARCHITECTURES PARALLELES

I-1 INTRODUCTION:

Le recours au parallélisme n'est guère une idée nouvelle, mais ce n'est qu'au début des années soixantes dix qu'on a commencé à construire de vrais calculateurs parallèles.

Le concept du parallélisme rompt avec l'approche classique qui consiste a gagner de la vitesse en effectuant plus rapidement chaque opération. En calcul parallèle, le gain de vitesse provient de la réalisation simultanée de plusieurs opérations. Ces calculateurs parallèles appelés multiprocesseurs possèdent plusieurs processeurs comme leur nom l'indique (de deux à plusieurs dizaines pour des systèmes généraux ou beaucoups plus pour des machines spécialisées).

Ainsi, autant que l'architecture se complique, autant que les problèmes d'accés mémoire.de communication et de synchronisation sont plus cruciaux, car un grand nombre de processeurs entraine impérativement un fonctionnement totalement indépendant (en particulier asynchrone) en executant des programmes différents, ce qui nous conduit à une décentralisation totale d'où la communication entre processeurs aura un sens insignifiant.

I-2 CLASSIFICATION DES ARCHITECTURES PARALLELES

Les modèles d'architectures parallèles sont nombreux, mais d'après la classification de Flynn en 1960, qui est basée sur la multiplicité des flux d'instructions et de données disponibles matériellement(comme nous le décrit la figure 1), on a 4 modes parallèles (voir figure 2).

- **Le mode SISD (single instruction, single data streams)**

C'est le mode de fonctionnement de la plus part des systèmes conventionnels utilisant des microprocesseurs classiques. Les instructions sont exécutées séquentiellement à un instant donné. Une seule instruction, au plus, peut être traitée et elle n'affécte qu'une seule donnée.

- **Le mode SIMD (single instruction, mutiple data streams)**:

C'est le mode de fonctionnement priviligié des matrices de processeurs. Les calculateurs opèrent sur une donnée vectorielle au lieu d'une donnée scalaire d'où l'appélation "processeurs véctoriels". Le flux multiple de données provenant d'une mémoire est la collection des flux individuels de données contenant chacun un élément du vecteur opérande.

Ces flux individuels de données attaquent plusieurs processeurs élémentaires identiques exécutant la même instruction au même instant, on obtient un

fonctionnement synchrone. La mémoire partagée peut être subdivisée en plusieurs modules qui permettent l'accés aux unités de traitement à travers un réseau d'interconnexion (voir figure 3-a).

- **Le MISD (mutiple instructions, single data stream)**:

Dans ce type de systèmes, chaque donnée est traitée par plusieurs instructions. Un tel mode de fonctionnement trouve généralement peu ou pas d'application, et,est donc non réalisable.

- **Le MIMD (multiple instructions, multiple data streams)**:

C'est le mode réservé aux grands ordinateurs qui combine le parallélisme des données au parallélisme des instructions, où chaque processeur étant un calculateur complet posséde sa propre unité de contrôle et exécute à lui seul son propre programme. Les unités de traitement accèdent à une mémoire partagée par l'intermédiaire d'un réseau d'interconnexion (voir figure 3-b).

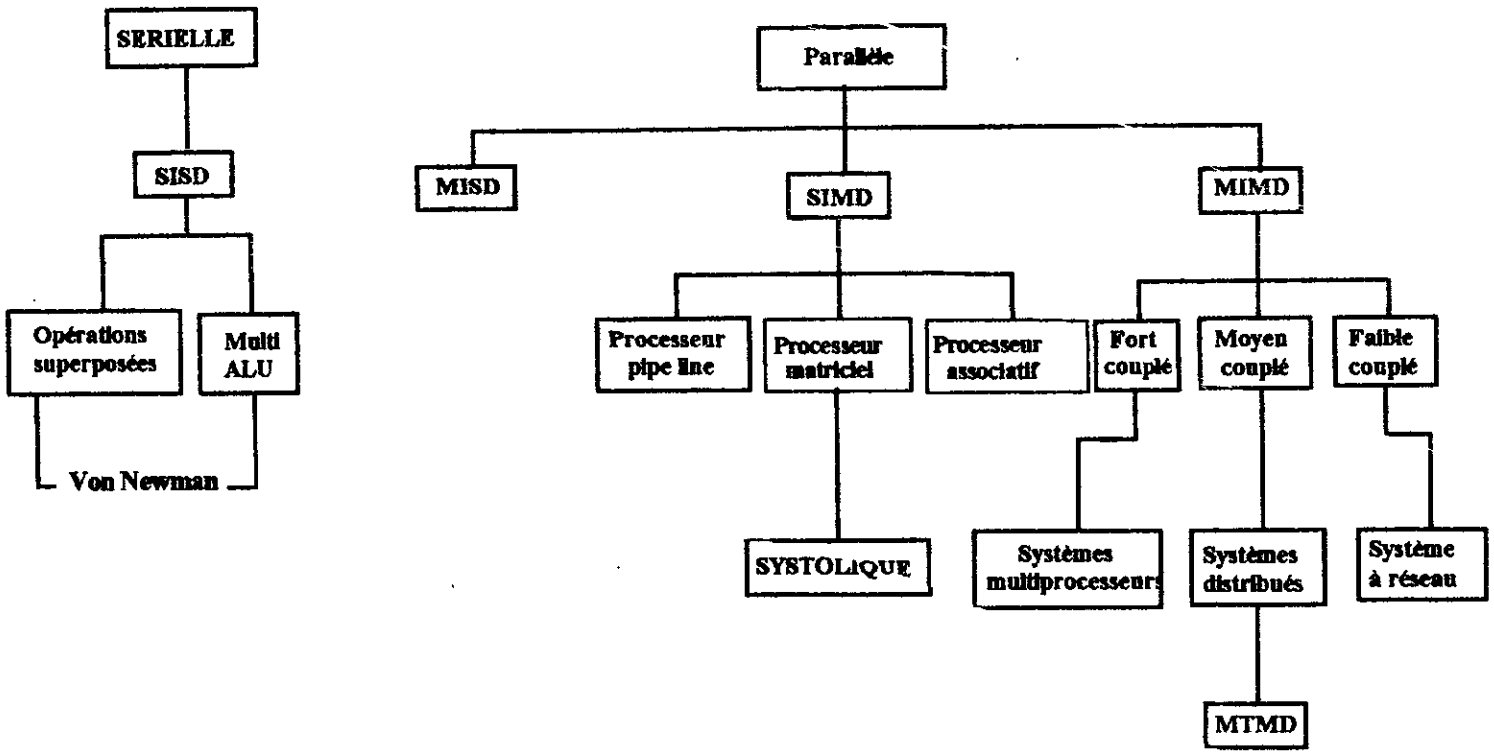


Figure 1: CLASSIFICATION DE FLYNN DES SYSTEMES

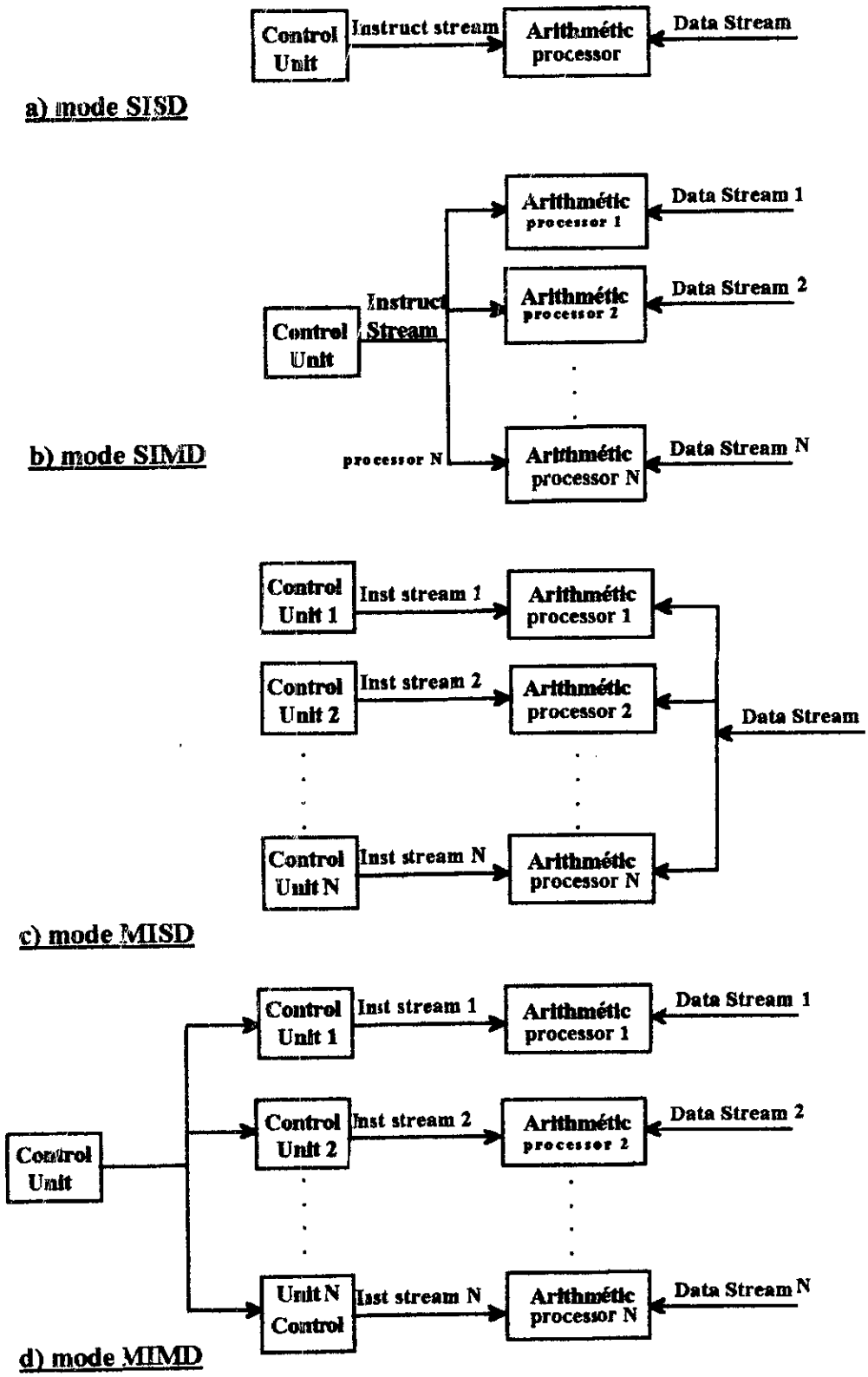

Control Unit
Instruct stream
Arithmétic processor
Data Stream
a) mode SISD
Control Unit
Instruct Stream
Arithmétic processor 1
Arithmétic processor 2
Arithmétic processor N
Data Stream 1
Data Stream 2
Data Stream N
b) mode SIMD
Control Unit 1
Control Unit 2
Control Unit N
Inst stream 1
Inst stream 2
Inst stream N
Arithmétic processor 1
Arithmétic processor 2
Arithmétic processor N
Data Stream
c) mode MISD
Control Unit
Control Unit 1
Control Unit 2
Unit N Control
Inst stream 1
Inst stream 2
Inst stream N
Arithmétic processor 1
Arithmétic processor 2
Arithmétic processor N
Data Stream 1
Data Stream 2
Data Stream N
d) mode MIMD

Figure 2

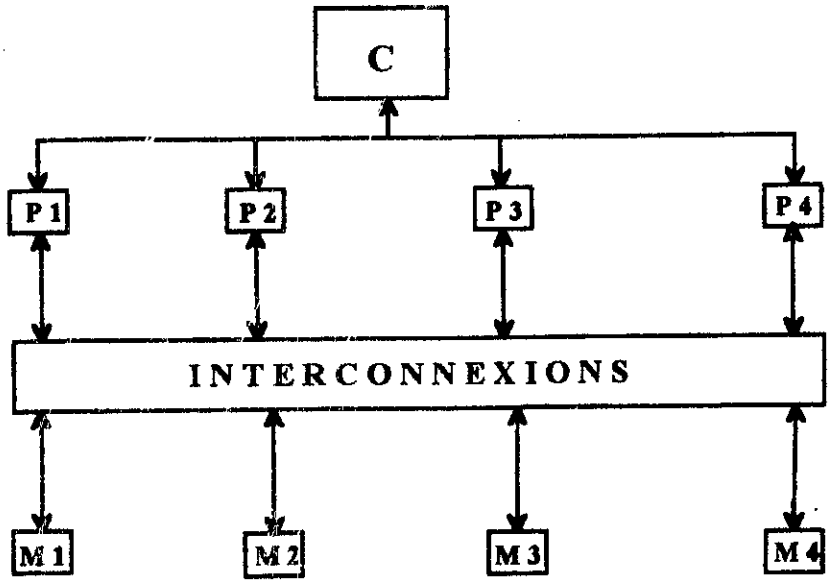


Figure 3 - a : ARCHITECTURE SIMD

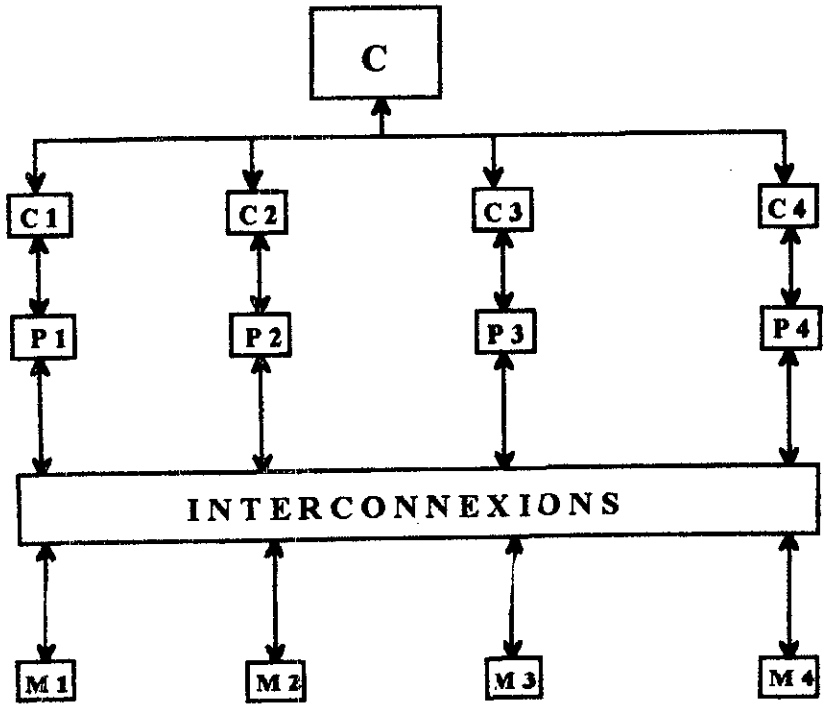


Figure 3 - b : ARCHITECTURE MIMD

Ci : Unité de contrôle
Pi : Processeur
Mi : Mémoire

FIGURE 3

I-3 PROGRAMMATION DES SYSTEMES A MULTIPROCESSEURS:

Théoriquement, la programmation des systèmes à multiprocesseurs est une programmation en parallèle vraie, définie comme une programmation parallèle qui est étroitement liée à la programmation temps réel permettant **l'exécution de plusieurs applications, en même temps,** sur un système sous le contrôle d'un logiciel de gestion.

Mais souvent, cette programmation s'avère impossible à réaliser, ce qui nous a poussé vers la programmation concurente.

Par définition, un programme concurrent est un programme non sequentiel, qui a la potentialité d'être exécuté en parallèle.

L'execution de ce programme, est réalisée par des opérations éffectuées en parallèle, simultanément.

Deux problèmes sont communs à toutes les applications concurentes:

* Le problème de l'exclusion mutuelle (accés à des ressources partagées).
* Le problème de la synchronisation entre les processus.

I-4 LA NOTION DE PROCESSUS:

Un processus est l'outil logiciel priviligié pour gérer le temps processeur. Un processus comme un sous-programme classique, exécute un certain nombre

d'opérations. Mais au lieu de les exécuter séquentiellement, les programmes concurents juxtaposent les processus d'une façon trés simple, en déffinissant les liens entre eux:

* ordre d'exécution des processus.

* allouer le temps processeur nécéssaire pour l'exécution du processus (une bonne allocation conduit à une bonne synchronisation).

* déterminer les communications entre processus pour l'échange de données. [1]

CHAPITRE 2

RESEAUX SYSTOLIQUES:

CARACTERISTIQUES

II-1 INTRODUCTION:

Les architectures systoliques sont le résultat du développement de la technologie des semi-conducteurs et des applications qui demandent une capacité de traitement extensive. La première réaction de toute personne se trouvant face à un concept d'architecture systolique, tombe sous l'admiration de l'élégance et la potentialité des grandes performances du système.

En effet, le principe est tiré, du mouvement de contraction du coeur appelé "Systole". Ce terme évoque l'idée d'une pulsation rythmique, qui peut être qualifiée de "battement" dans une machine systolique, où l'information traitée progresse à travers le réseau (machine systolique), subissant à chaque battement un calcul qui la combine avec les résultats déjà produits. D'où la nomination systolique, qui provient de l'analogie entre la circulation des flux de données dans le réseau et celle du sang dans le corps humain. L'horloge qui assure la synchronisation globale constitue le coeur du système. [2,5]

II-2 DEFINITION:

En 1978, HT Kung et CE Leiserson ont introduit le concept de structure systolique, qui est une simple machine parallèle spécialisée, agencée en forme de réseau. Ces réseaux se composent d'un grand nombre de cellules

élémentaires identiques et localement interconnectées (figure 4). Chaque cellule reçoit des données en provenance des cellules voisines, effectue un calcul simple, puis transmet les résultats aux cellules voisines, un temps de cycle plus tard. Seules, les cellules situées aux frontières du réseau communiquent avec le monde extérieur. Les cellules évoluent en parallèle, sous le contrôle d'une horloge globale (synchronisation totale). [2,3]

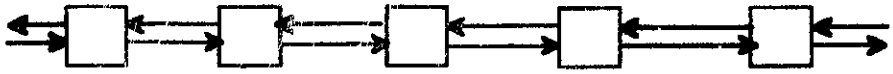
On distingue deux catégories de structures systoliques:

Première catégorie - **structures systoliques pures:** ce sont les structures systoliques dont on a déjà parlé et qui sont caractérisées par

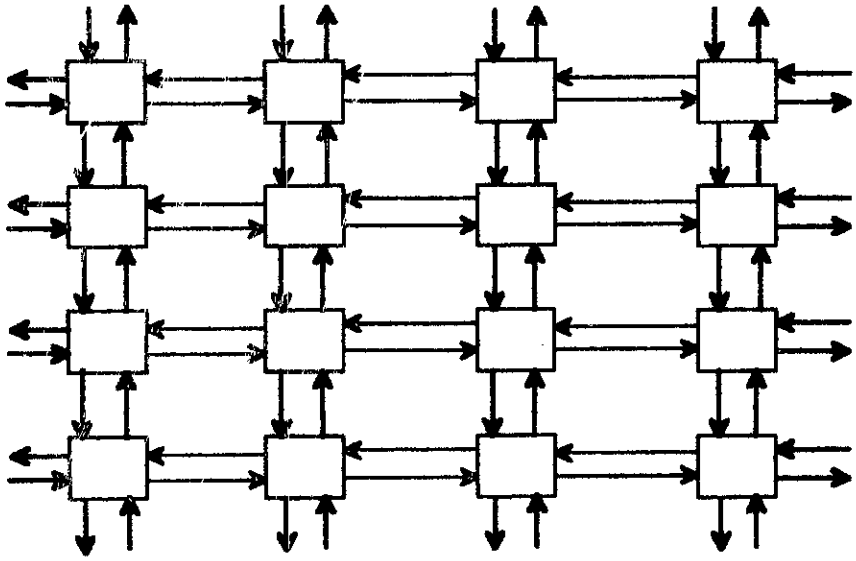
* des cellules identiques.
* des cellules sans mémoire locale.
* chaque cellule communique avec ses voisins seulement.
* le réseau fonctionne avec une seule horloge de synchronisation.

Deuxième catégorie - **structures semi-systoliques:** Ce sont des structures avec le même principe de fonctionnement, mais caractérisées par:

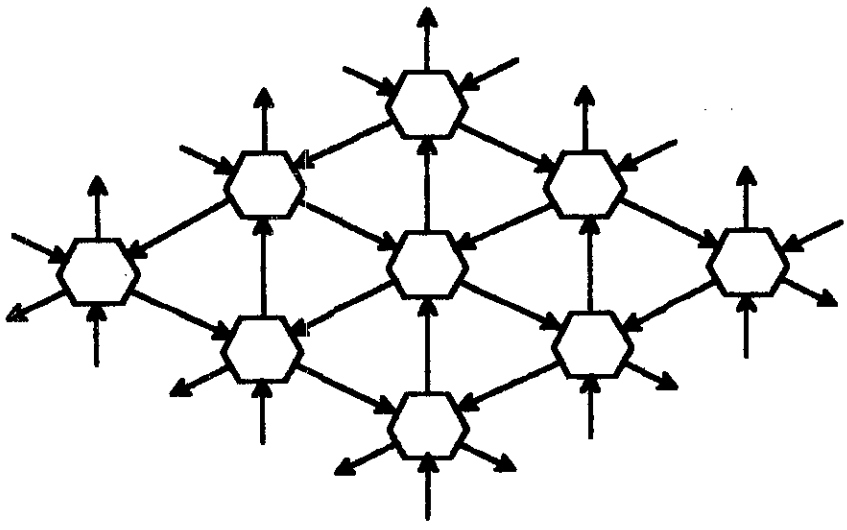
* des cellules avec mémoire locale, qui dans certain cas possèdent des ALU.
* possibilité de communication entre deux cellules non voisines.
* possibilité d'utiliser plus d'une horloge car la synchronisation globale n'est pas indispensable.
* possibilité d'avoir une asynchronisation totale.



a) Réseau linéaire



b) Réseau orthogonal



c) Réseau héxagonal

Figure 4: EXEMPLES DE TOPOLOGIE

II-3 PRINCIPE DES RESEAUX SYSTOLIQUES:

Une architecture systolique est un réseau régulier de processeurs relativement simples, où les données circulent à travers la structure de façon synchrone et selon un cheminement fixe.

Dans l'architecture systolique, plusieurs calculs sont effectués sur une même donnée à l'intérieur du réseau. Une fois qu'une donnée en provenance de la mémoire externe est lue par le réseau, elle passe de cellule en cellule et peut donc être utilisée plusieurs fois (voir figure 5-b).

Ce mode de calcul est à l'opposé du fonctionnement basé sur un accès mémoire à chaque utilisation d'une donnée, comme nous le décrit la figure (5-a).

Par conséquent, un réseau systolique peut être étendu pour traiter un problème avec un nombre très important d'opérations sans qu'il faille imposer pour autant une augmentation correspondante de la bande passante de la mémoire externe. Cette propriété confère aux réseaux systoliques un avantage majeur sur les architectures traditionnelles, qui sont limitées par "le goulot d'étranglement de Von Newmann".

Néanmoins, bien que seules les cellules frontières d'un réseau systolique communiquent avec le monde extérieur, il reste une limite incontournable en matière d'entrées/sorties, à savoir le nombre de plots disponibles sur un même boîtier VLSI.

Nous concluons, que les structures systoliques sont adaptées à l'implémentation d'algorithmes bornés par les calculs, où le fonctionnement en temps réel délivre les sorties au même rythme des entrées, plutôt qu'à la résolution de problèmes bornés par les entrées/sorties.

Dans un algorithme borné par les calculs, le nombre de calculs élémentaires est plus grand que le nombre de données en entrées/sorties comme nous le décrit l'exemple de multiplication de deux matrices de taille n, qui nécessite $O(n^3)$ multiplications et $O(n^2)$ additions pour $O(n^2)$ données.

Par ailleurs, un algorithme borné par les entrées/sorties, comme le cas d'une addition de deux matrices de taille n, qui exige n^2 additions pour $3n^2$ opérations d'entrées/sorties, est un algorithme qui ne se prête pas à un traitement VLSI.[4,5]

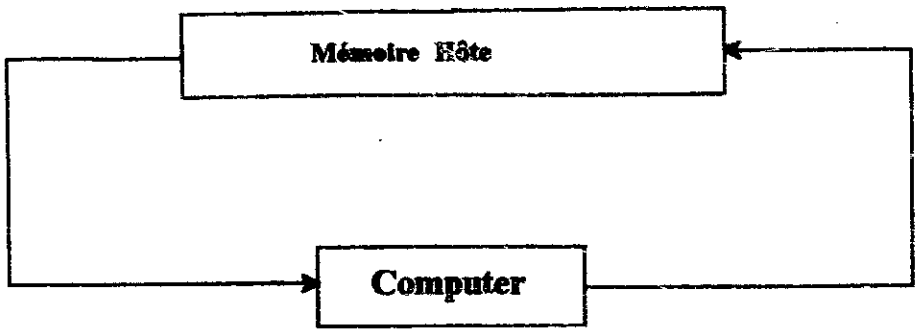


Figure 5 - a: PRINCIPE DE VAN NEW MAN

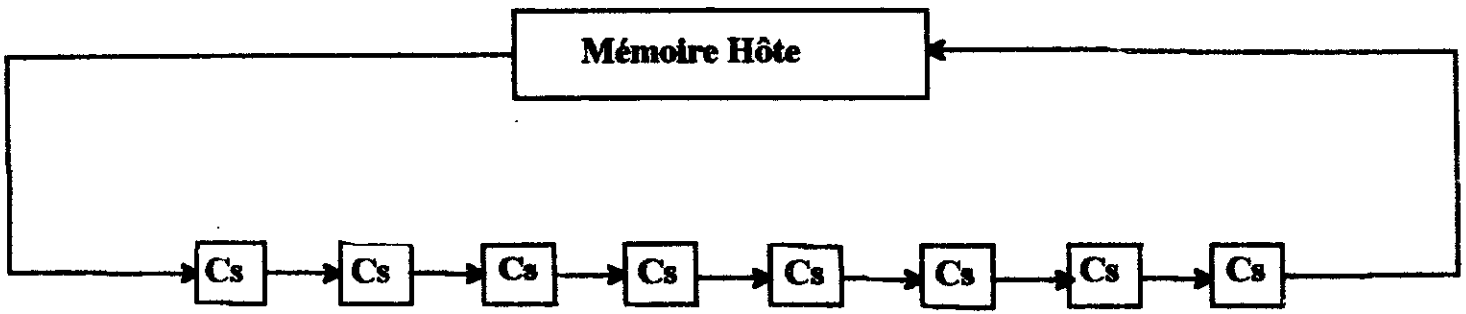


Figure 5 - b: PRINCIPE DES RESEAUX SYSTOLIQUES

FIGURE 5

Cs : Cellule systolique

II-4 SIMPLICITE ET REGULARITE:

- Les architectures systoliques sont composées à partir d'un petit nombre de cellules de base (avantage par rapport à une architecture composée d'une grande variété de cellules complexes).

- L'interconnexion locale et régulière des cellules, facilite grandement l'implantation topologique.

- D'une manière générale, la modularité et la faible sortance (nombre maximal de copies d'une même variable crées à l'intérieur du réseau) des réseaux systoliques leur donnent une grande reconfigurabilité: ils sont adaptables à la taille et à la nature du problème traité.

De même, dans le domaine de la testabilité, on peut profiter de la régularité de l'architecture pour élaborer une séquence de test pour valider la fabrication. [4]

II-5 OBJECTIF:

L'objectif qui est en général fixé à une architecture systolique, est d'être taillée sur mesure, ASIC (application spécific intergrated circuits) pour une application donnée, afin d'accélérer certains types de traitements.

Le problème rencontré, est de minimiser le coût de conception et de réalisation de cette architecture, tout en lui donnant une puissance de calcul suffisante pour l'application à laquelle elle est destinée.

Leur intérêt est en grande partie lié aux possibilités offertes par les techniques d'intégration.

En d'autre termes, il s'agit de processeurs spécialisés que l'on adjoint à un processeur appelé "Hôte" de type conventionnel plutôt que de calculateurs généraux . voir figure 8. [4]

II-6 PRINCIPE DE LOCALITE:

Lorsqu'on cherche à tirer parti d'une architecture parallèle pour exécuter un algorithme donné, le problème principal peut se résumer de la façon suivante: Comment répartir l'algorithme sur les processeurs, de telle sorte, qu'ils soient tous actifs, en permanence, pendant la durée de l'exécution de l'algorithme, et ce, en effectuant un travail utile ?

En examinant de prés ce problème dans la réalité, on s'aperçoit rapidement que la condition principale de succès n'est pas de disposer de processeurs qui calculent vite, encore que cela ne puisse nuire, mais plutôt que ces processeurs puissent communiquer entre eux de façon efficace, c'est à dire qu'ils puissent

échanger de l'information aussi rapidement qu'ils accèdent à (et opèrent sur) leurs propres données.

Ici intervient une donnée essentielle de la technologie actuelle. Il est impossible de réaliser une machine ayant un nombre de processeurs élevés, qui peuvent communiquer deux à deux de façon directe. (voir figure 6).
Tout réseau de communication fait intervenir d'une façon ou d'une autre le concept de localité: pour une répartition spatiale donnée des processeurs, il n'est possible de relier efficacement un processeur qu'avec un voisin de celui-ci. (voir figure 7).

En conséquence, **un algorithme ne pourra être programmé de façon éfficace sur la machine, que si son execution peut être répartie de telle sorte, que chaque processeur n'ait à communiquer qu'avec ses voisins au sens du réseau de communication de l'architecture.** Il s'agit l'à d'une propriété très forte.

Une seconde conséquence du mode de connexion locale, concerne le rapport entre entrées/sorties et calcul. Puisqu'il n'est pas possible de relier chaque processeur à tous les autres, il est de la même façon difficile d'imaginer que chaque processeur soit relié directement à l'environnement extérieur de la machine, qu'on appelle l'Hôte du système, censé fournir les données de l'algorithme et recueillir ses résultats. D'une façon ou d'une autre, cela signifie qu'il faut que la complexité de l'algorithme repose sur les calculs et non sur la

quantité d'entrées/sorties qu'il doit effectuer. C'est ce qu'on appelle un **algorithme borné par les calculs**. Ceci est très important et doit être pris pour règle: **Le volume de calculs à effectuer doit primer largement sur les transferts de données à réaliser**. [1,4]

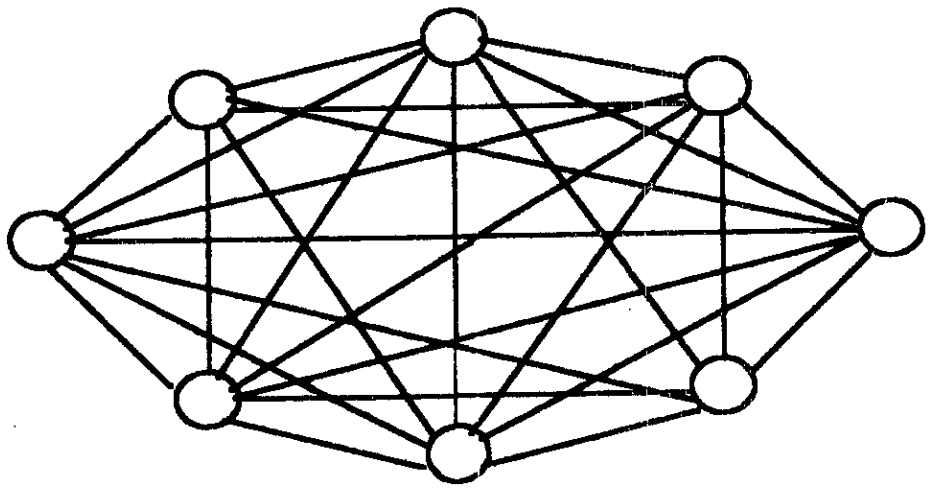


Figure 6 : Réseau à maillage complet

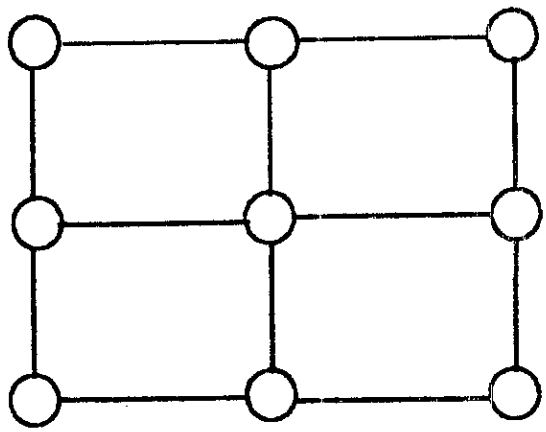


Figure 7 : Présentation planaire

Figure 8 : Structure du système systolique

II-7 CARACTERISTIQUES DES RESEAUX SYSTOLIQUES:

A partir de leur structures, on peut en déduire que les caractéristiques essentielles et principales des réseaux systoliques sont:

* un parallélisme massif et décentralisé.

* des communications locales.

* un mode opératoire synchrone.

II-8 APPLICATIONS DES RESEAUX SYSTOLIQUES:

On distingue deux domaines d'applications des réseaux systoliques et qui sont:

* ***Applications numériques et traitement du signal:***

- Traitement du signal:

. convolution unidirectionnelle.

. convolution bidirectionnelle et corrélation.

. lissage par médiane.

. transformée de Fourrier discrète.

. projections géométriques.

. systèmes adaptatifs.

. filtres de Kalman.

- Arithmétiques matricielles:

. multiplication matrice - vecteur.

. multiplication matrice - matrice.

. triangulation de matrice.

. résolution des systèmes triangulaires.

décomposition QR.

. problème aux valeurs propres.

* ***Applications non numériques:***

- Structures de données:

. pile, file d'attente.

. tri.

. base de données relationnelles.

- Graphes et algorithmes géométriques:

. fermeture transitive, problème de chemin dans un graphe.

. arbre de degré minimum, composantes connexes.

. enveloppes convexes.

- Manipulation de chaînes de caractères:

. occurrence d'un mot dans une machine.

. plus longue sous suite.

. reconnaissance de langages.

. programmation dynamique.

- Algèbre des polynômes et arithmétique:

. multiplication, division euclidienne, PGCD de polynômes.

. arithmétique entière et dans les corps finis multiplication, division, PGCD[4]

CHAPITRE 3

CONCEPTION DES RESEAUX SYSTOLIQUES

III-1 INTRODUCTION:

Les applications des systèmes systoliques sont assez larges, comme on l'avait déjà cité, le but de notre projet est d'étudier les architectures systoliques puis simuler quelques applications, tels que la **multiplication matricielle et le TRI**. Pour cela nous avons jugé indispensable d'illustrer la l'implémentation de quelques algorithmes, objet de ce chapitre.

Pour une réalisation optimale d'une application sur un réseau systolique, on doit, à partir d'un problème posé:

1- *Régulariser l'algorithme: où on distingue deux méthodes:*

- Méthodes algébriques.
- Méthodes graphiques.

2- *Le Mapping sur architecture:*

Après régularisation de l'algorithme, on doit l'adapter à une architecture donnée. Pour cela, on n'a pas de méthodes précises à suivre, le chercheur doit s'inspirer de son intuition personnelle.

III-2 CONVOLUTION NON-RECURSIVE:

La convolution est une opération très utilisée dans différents domaines tel que le traitement de signal, définie comme suit: $Yi = \sum_{k=1}^{K} a_k \, x_{i-k+1}$

soit la suite de données x_1, x_2,.... qui donnera pour tout entier $i \geq K$ un Y_i tel que:

$$Y_i = a_1.x_i + a_2.x_{i-1} + a_3.x_{i-2} + ...+a_k \, x_{i-k+1}$$

où les a_1, a_2, ...a_k sont des coefficients fixés.

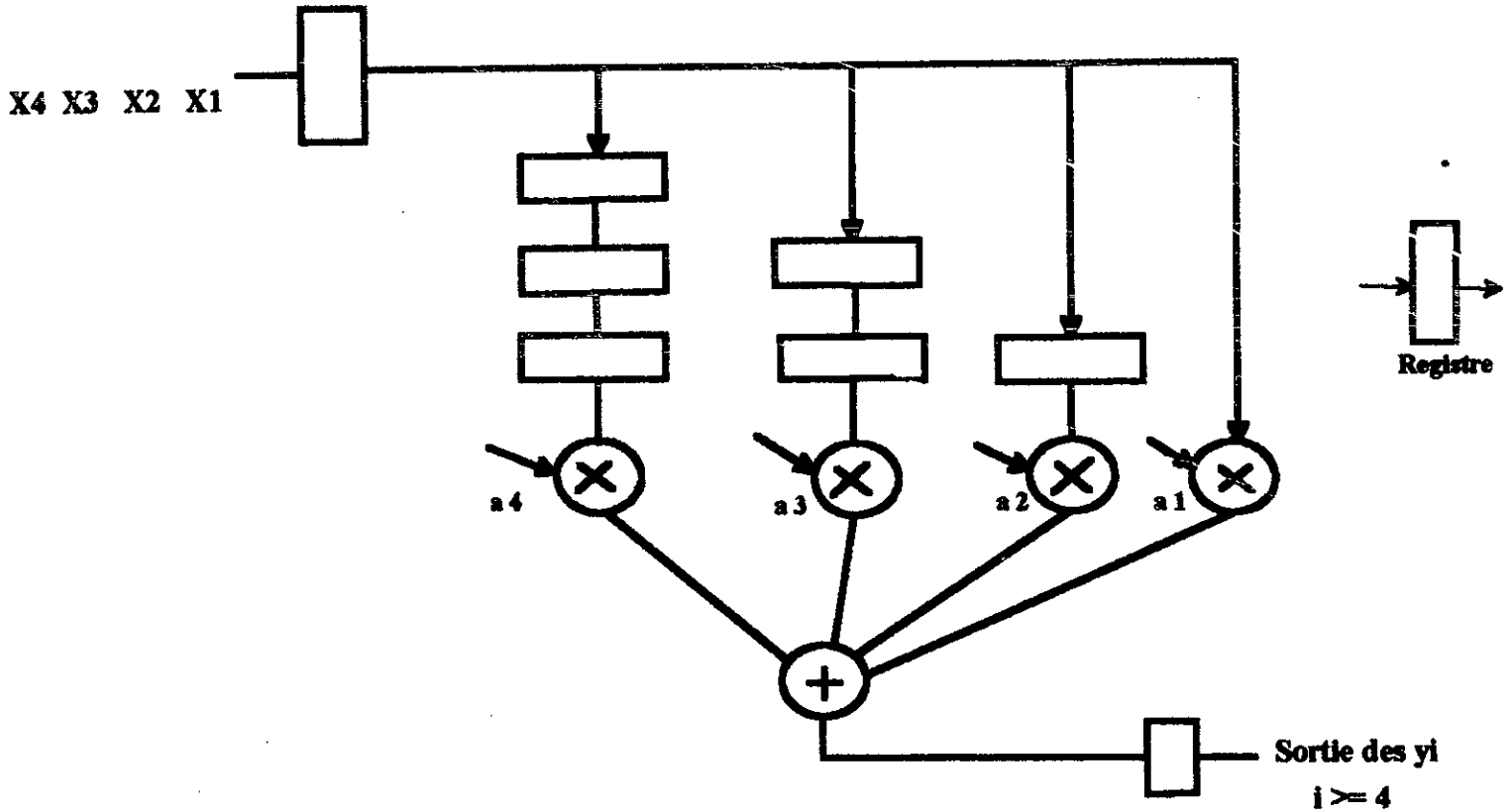
On va résoudre le problème de deux manières différentes dans le but de mettre en valeur la meilleure solution.

La première, bien connue en théorie du filtrage numérique, est décrite par la figure 9 (pour K = 4).

III-2-1 PRINCIPE DE FONCTIONNEMENT (solution classique):

Le principe de fonctionnement est simple: à chaque top d'horloge, un nouvel x_i entre dans le circuit, les K multiplications auront lieu en parallèle, puis on additionne les K résultats soit, avec un additionneur à K entrées, ou encore en utilisant (K - 1) additionneurs à deux entrées disposées en série. Notons qu'un nouvel Yi est calculé à chaque cycle T_k , défini comme la plus petite période d'horloge permettant la réalisation d'une multiplication et de (K-1) additions.

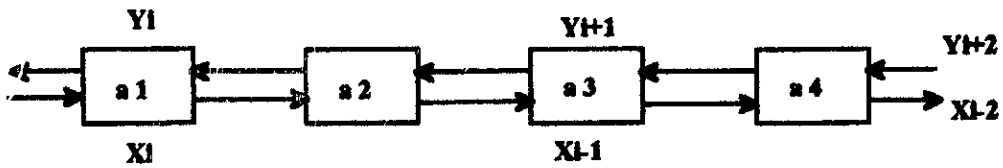
La deuxième solution proposée par un réseau systolique est illustrée par la figure-10- (pour K= 4).



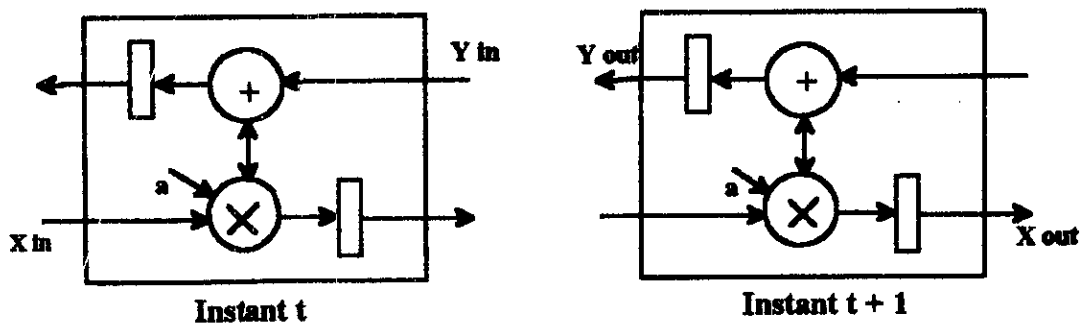
CIRCUIT CLASSIQUE DE CONVOLUTION

L'élément symbolisé par un réctangle appelé "registre", représente un point de mémorisation, commandé par l'horloge du circuit.

Figure 9 : Première solution



RESEAU SYSTOLIQUE DE CONVOLUTION NON RECURSIVE



X out = X in

Y out = Y in + a . Xin

UNE CELLULE DU RESEAU

Figure 10

III-2-2 PRINCIPE DE FONCTIONNEMENT (Solution Systolique)

Le réseau systolique est composé de K cellules identiques . Il fonctionne de manière totalement synchrone au rythme d'une horloge globale qui délivre des "tops" à toutes les cellules. Chaque cellule est tenue de réaliser une multiplication suivie d'une addition d'où le nom de cellule MAC (Multiplication - Accumulation).

Il y a deux flots de données dans le réseau :

- Les x_i circulent de gauche à droite , gagnant une nouvelle cellule à chaque top et sans être modifiés durant leur parcours. L'ordinateur Hôte délivre un nouvel x_i tous les deux tops , c'est à dire que x_i entre dans le réseau à un temps $t = 2i-1$ en supposant que x_1 rentre à $t = 1$.

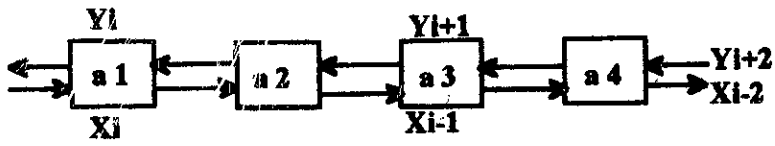
- Les Y_i circulent de droite à gauche (sens inverse des x_i) à la même vitesse , et se calculent par accumulations successives. Initialement leur valeur est zéro , et quand ils sont délivrés en sortie du réseau , ils ont accumulé leur valeur définitive .

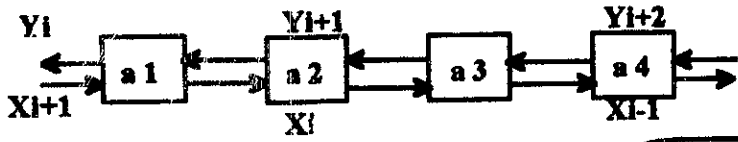
$$Y_i = a_1 x_i + a_2 x_{i-1} + a_3 x_{i-2} + a_4 x_{i-3}$$

Pour nous convaincre, plaçons nous à l'instant t où la variable Y_{i+2} se trouve en entrée du réseau, initialisée avec la valeur $Y_{i+2} = 0$

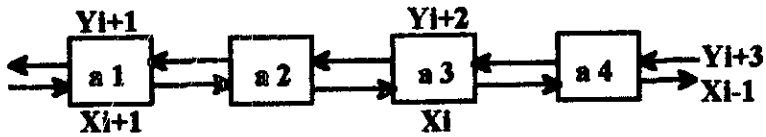
L'exemple est illustré par la figure 11, où à (t + 1) la première accumulation aura lieu dans la quatrième cellule, où on aura:

$$Y_{i+2} = Y_{i+2} + a_1 x_{i+2}$$

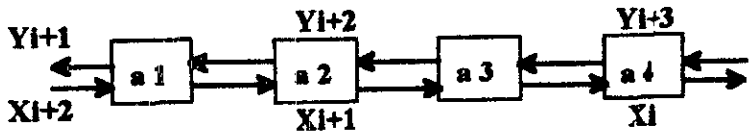




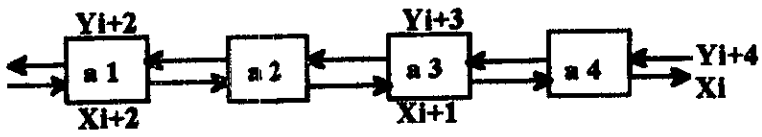
$Yi+2 = Yi+2 + a4\ Xi-1$



$Yi+2 = Yi+2 + a3\ Xi$



$Yi+2 = Yi+2 + a2\ Xi+1$



$Yi+2 = Yi+2 + a1\ Xi+2$

Figure 11

Trois (03) tops après, la dernière accumulation se fera dans la première cellule et la valeur définitive obtenue sera:

$$Y_{i+2} = a_1 x_{i+2} + a_2 x_{i+1} + a_3 x_i + a_4 x_{i-1}$$

Il faut noter que seule la cellule de gauche communique avec l'ordinateur Hôte. Le port d'entrée de la cellule droite est fictif, puisque les Yi sont initialisés à zéro (voir figure 12). Chaque variable xi en provenance de l'Hôte n'est lue qu'une fois et elle est réutilisée plusieurs fois dans le réseau, ce qui permet d'augmenter les performances sans pour autant accroître le débit nécessaire en matière d'entrées/sorties.

On remarque que plusieurs variables Y_i sont calculées en parallèle. Le réseau fonctionne en pipeline. Ainsi pendant le calcul de Y_{i+2} se termine le calcul de Y_{i+1} et débute celui de Y_{i+3} .Un nouveau résultat est délivré par le réseau tous les deux temps de cycle T_{syst} .

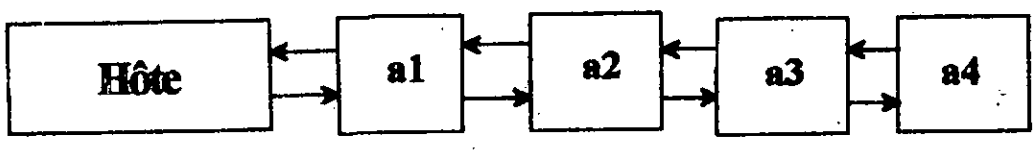


Figure 12: COMMUNICATION AVEC L'EXTERIEUR

III-2-3CONCLUSIONS:

Après l'étude des deux méthodes, nous concluons que le réseau systolique est beaucoup plus performant que la solution classique et cela pour plusieurs raisons.

* Pour K assez grand, TK > 2Tsyst (Tsyst est le temps de cycle d'une cellule systolique, et correspond au temps nécessaire pour une multiplication suivie d'une addition).

* Le second avantage est que Tsyst ne dépend pas de K, la taille du filtre.

. *Notons, que la principale faiblesse du réseau systolique et que les cellules ne sont actives que tous les deux tops, par conséquent, chaque cellule ne travaille qu'un cycle sur deux.* [4]

III-3 LA MULTIPLICATION MATRICIELLE:

Comme pour la convolution, nous allons parler de notre deuxième application qui est la multiplication matricielle, et pour cela nous commençons par la multiplication matrice vecteur et nous terminons par la multiplication des matrices denses.

III-3-1 MULTIPLICATION MATRICE - VECTEUR:

On a le produit matrice vecteur Y= A. X, tel que A= (a_{ij}), matrice carrée d'ordre n et X, Y des vecteurs à n composantes. Notre inconnue est le vecteur Y. Pour résoudre ce système, nous avons les formules récurrentes suivantes, où les calculs éffectués sont des multiplications suivies d'additions

$$Y_i^{(0)} = 0$$

$$Y_i^{(k)} = Y_i^{(k-1)} + a_{ik}\ x_k$$

$$Y_i = Y_i^{(n)}$$

Nous concluons qu'un réseau formé de cellules MAC, dont le fonctionnement est décrit par la figure 13, sera une bonne solution au problème. La seule différence avec la convolution non récursive est que le registre interne de chaque cellule MAC doit être changé dynamiquement au cours du temps, d'où la nécessité d'adjoindre un port d'entrée/sortie vertical pour recevoir les coefficients de la matrice A. Nous décrirons un exemple avec n=3 (figure 14) correspondant à la multiplication matrice-vecteur.

$$\begin{pmatrix} Y_1 \\ Y_2 \\ Y_3 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

L'algorithme est illustré par la figure (15) où :

* les coefficients de la matrice A entrent dans le réseau par diagonales, un nouvel élément tous les deux tops.

* les composants du vecteur X circulent inchangés de droite à gauche.

* les composants du vecteur Y circulent en sens inverse des Xi ,à valeur initiale zéro. En passant à travers le réseau, chaque Yi accumule ses produits partiels.

Le réseau de Kung et Leiserson comprend (2n-1) cellules. Après un délai de (2n-1) tops, les Yi commencent à émerger à la droite du réseau. 2 (n-1) tops plus tard, la dernière composante Yn quitte le réseau. Donc il faut (4n-3) cycles pour réaliser un produit matrice-vecteur.

On note que, en régime permanent, chaque cellule du réseau ne travaille qu'une pulsation sur deux, mais ceci n'implique pas que l'efficacité est e = 1/2. Le nombre de pas en séquentiel est n^2 et nous obtenons:

$$e = \frac{T_{seq}}{P.T_p} = \frac{n^2}{(2n-1)(4n-3)}$$ (pour n grand : e = 1/8)

Tseq: temps nécessaire pour effectuer l'opération en séquentiel.

Tp : Temps nécessaire pour effectuer l'opération en parallèle.

P : Nombre de cellules nécessaire pour effectuer l'opération.

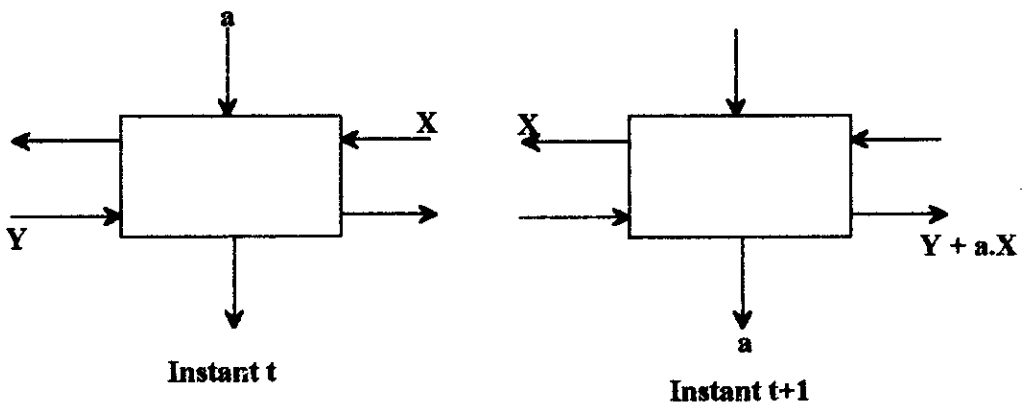


Figure 13

-	-	a33	-	-
-	a32	-	a23	-
a31	-	a22	-	a13
-	a21	-	a12	-
-	-	a11	-	-

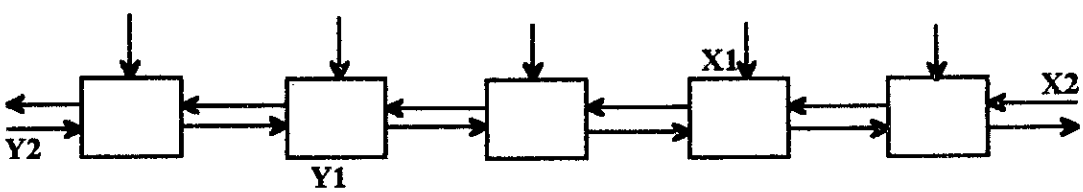


Figure 14

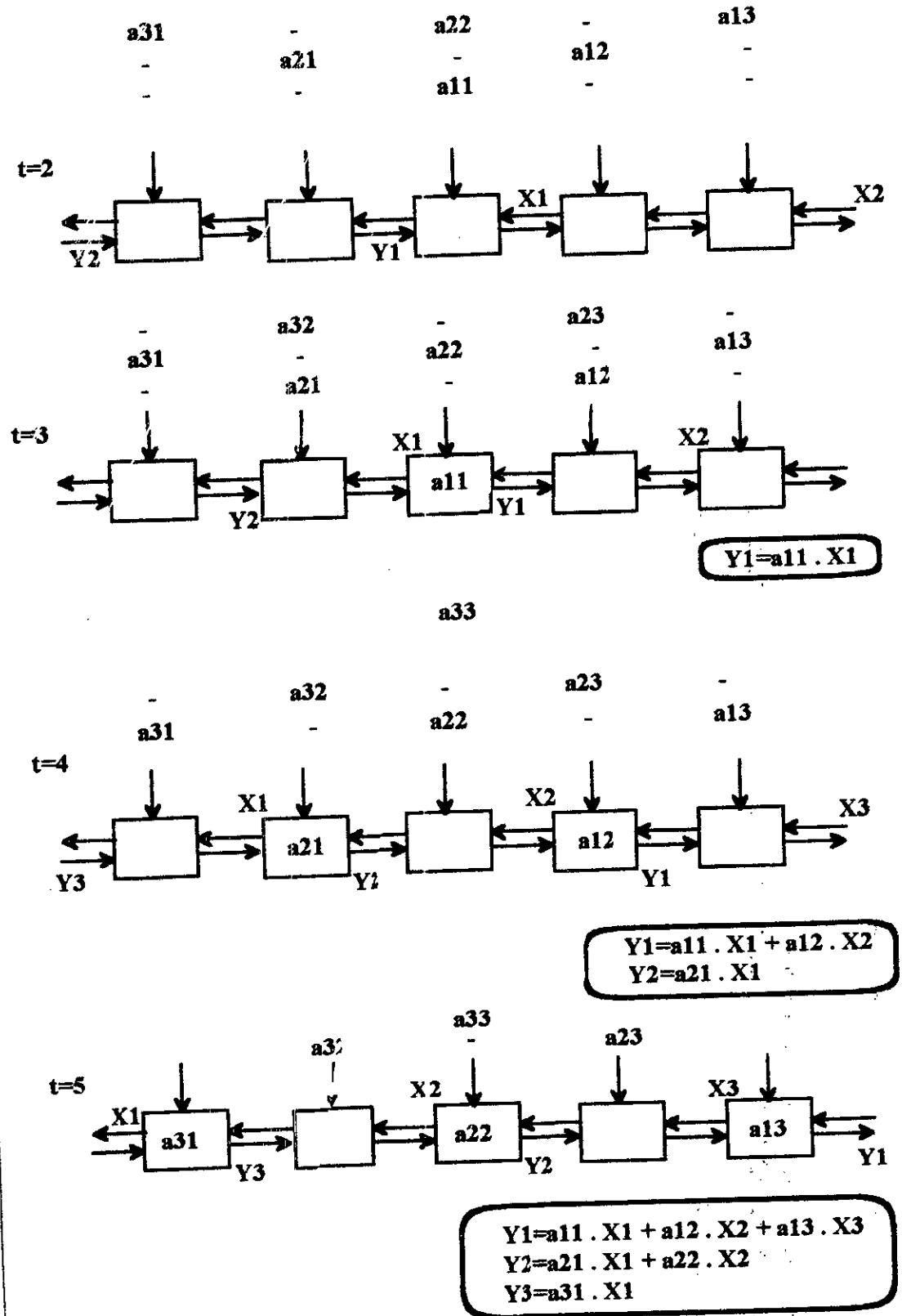
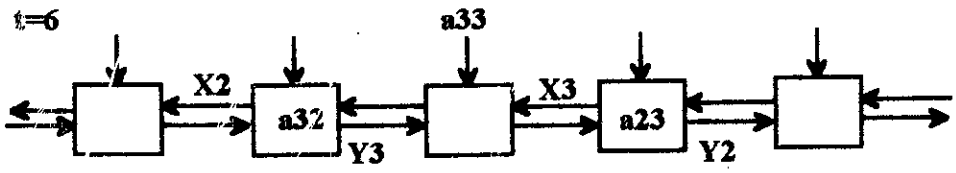
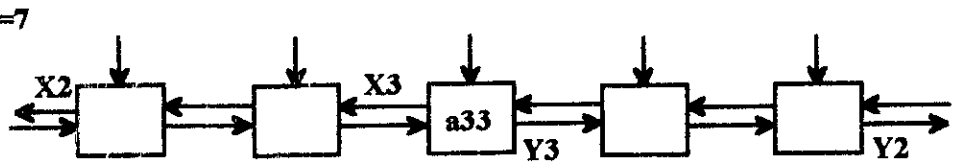

a31
a21
a22
a12
a13
a11
t=2
X1
X2
Y2
Y1
a32
a23
t=3
Y1=a11 . X1
a33
t=4
X3
Y3
Y2=a21 . X1
Y1=a11 . X1 + a12 . X2
t=5
Y1=a11 . X1 + a12 . X2 + a13 . X3
Y2=a21 . X1 + a22 . X2
Y3=a31 . X1

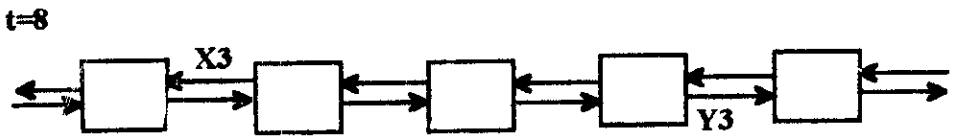
Figure 15

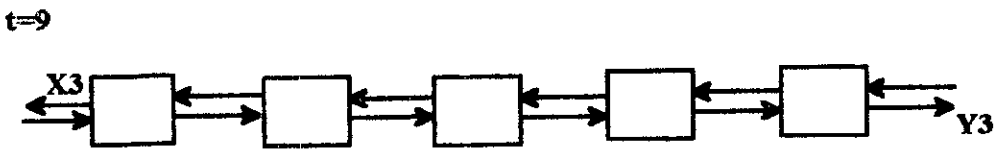


$Y2 = a21 . X1 + a22 . X2 + a23 . X3$
$Y3 = a31 . X1 + a32 . X2$



$Y3 = a31 . X1 + a32 . X2 + a33 . X3$





MULTIPLICATION MATRICE - VECTEUR (n=3)

Figure 15

Ceci est dû à l'initialisation du réseau: aucun calcul n'est effectué pendant n-1 premiers pas (n=3 : les 2 premiers tops), pendant que x1 et y1 circulent en sens inverse pour rejoindre la cellule centrale.

Il est important de souligner que seules les cellules frontières communiquent avec l'Hôte, grâce à leurs ports d'entrée et sortie horizontaux, tandis que les cellules internes ne communiquent avec l'Hôte que par port vertical, elles reçoivent les X_i et les Y_i de leurs voisins.

III-3-2 SYSTEME LINEAIRE TRIANGULAIRE:

C'est le problème inverse de la multiplication matrice-vecteur . Soit une matrice triangulaire inférieur **A** d'ordre **n** et un vecteur **b** à **n** composantes, calculons la solution **x** du système linéaire **Ax = b**. Le vecteur solution

$\mathbf{x} = (x_1, x_2, x_3, \ldots\ldots x_n)$ se calcule à l'aide des formules récurrentes:

$$x_i^{(0)} = 0$$

$$x_i^{(k)} = x_i^{(k-1)} + a_{ik}\, x_k \qquad 1 <= k <= i-1$$

$$x_i = (b_i - x_i^{(i-1)}) / a_{ii}$$

La similarité de cette récurrence avec le produit matrice - vecteur nous conduit à envisager un réseau linéaire semblable, en supprimant les cellules relatives à la partie triangulaire supérieure de la matrice et en remplaçant la cellule carrée qui réalise une multiplication suivie d'une addition en recevant la diagonale par une

cellule ronde qui réalisera une soustraction suivie d'une division, dont le fonctionnement est décrit par la figure 16. Les valeurs finales des composantes du vecteur solution x sont données par cette cellule.

La figure 17 est la solution qui résoud le système pour n = 3.

$$\begin{pmatrix} a_{11} & 0 & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{31} & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

Avec un algorithme:

- les éléments xi initialisés à 0, circulent de gauche à droite dans le réseau pour accumuler leurs produits partiels $a_{i1}\, x_1 + a_{12} x_2 \ldots\ldots + a_{i,i-1}\ x_{i-1}$.

Arrivé à la cellule ronde, sa valeur finale est calculée:

$$x_i = (b_i - x_i) / a_{ii}$$

Ensuite x_i est renvoyé dans le réseau et circule vers la gauche sans être modifié, afin de permettre aux composantes suivantes x_j, $j >= i$, d'accumuler leur produit partiel $a_{ji} \cdot x_i$.

Un nouvel x_i est délivré par la cellule ronde tous les deux temps de cycle, ce qui conduit à un temps de calcul égal à (2n-1) tops on a $T_{seq} = n^2/2 + O(n)$, alors l'efficacité obtenue est égale:

$$e = \frac{Tseq}{n.2n}$$

e= 1/4 pour n grand.

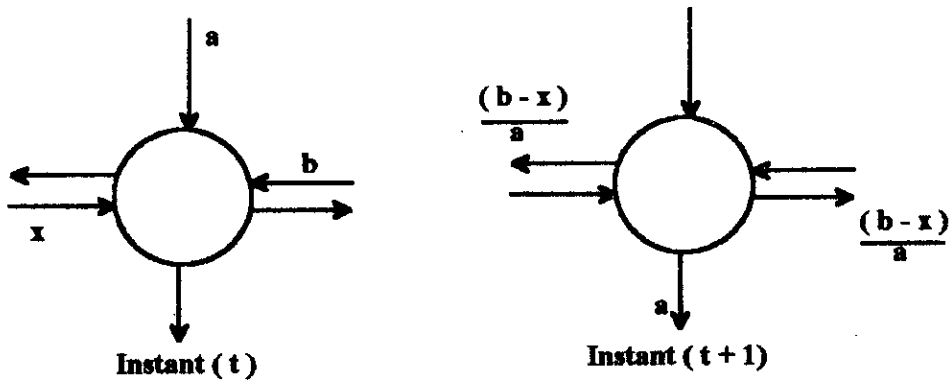
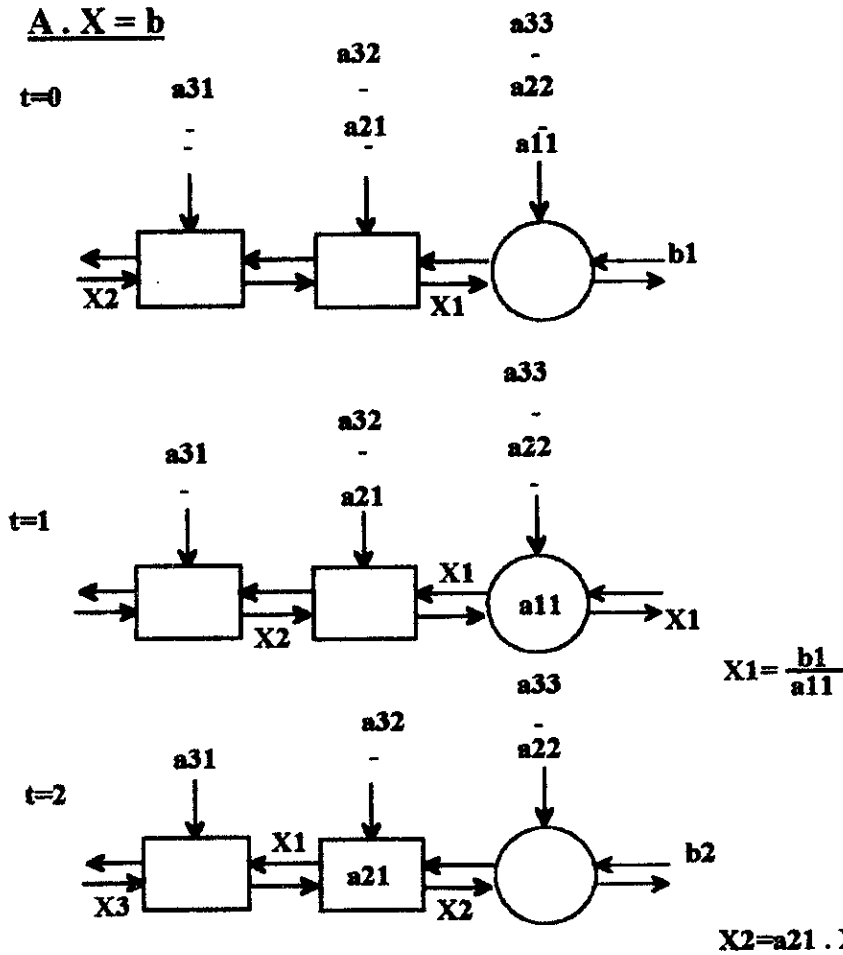
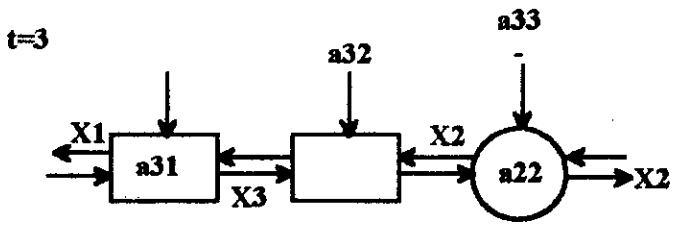


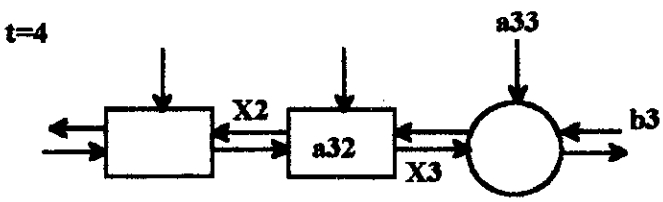
Figure 16 : FONCTIONNEMENT D'UNE CELLULE RONDE



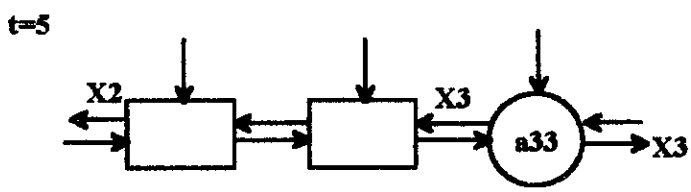


$$X3 = a31 \cdot X1$$

$$X2 = \frac{(b2 - X2)}{a22} = \frac{(b2 - a21 \cdot X1)}{a22}$$



$$X3 = a32 \cdot X2$$



$$X3 = \frac{(b3 - X3)}{a33}$$

avec : $X3 = a31 \cdot X1 + a32 \cdot X2$

Figure 17 : SYSTEME TRIANGULAIRE D'EQUATIONS (n=3)

III-3-3 MATRICES A STRUCTURE BANDE:

Dans ce paragraphe, on parlera des réseaux précédents adaptés au traitement des matrices à structure bande, qu'on rencontre fréquemment en calcul scientifique.

Soit une matrice A= (aij), une matrice carrée d'ordre n, à structure bande de largeur de bande w= p+q-1.

(A) possède (p) super-diagonale (y compris la diagonale) et (q) sous diagonales (y compris la diagonale), les autres éléments nuls

$$A=\begin{pmatrix} \overbrace{a_{11} \quad a_{12}}^{p} & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ 0 & a_{32} & a_{33} & a_{34} \\ 0 & 0 & a_{43} & a_{44} \end{pmatrix} \quad \text{- matrice à structure bande -}$$

$$p = q = 2$$

Soit le produit matrice-vecteur Y = A. x. avec x un vecteur à n composantes:

$$Y_i = a_{i,i-p+1}\, x_{i-p+1} \ldots\ldots\ldots + a_{i,i-1} \cdot x_{i-1} + a_{i,i} \cdot x_i + a_{i,i+1}\, x_{i+1} + \ldots\ldots\ldots a_{i,i+q-1}\, x_{i+q-1}.$$

Le réseau solution est identique au précédent (voir figure 18), mais ne comporte que W cellules, et le produit est calculé en temps 2 (n-1)+w.

De même pour la résolution du système A . x = b, seules q cellules sont nécessaires. Le temps de calcul reste égal à (2n-1) cycles.

Dans les deux cas, la propriété remarquable est que le nombre de processeurs ne dépend pas de n, la taille du problème traiter, mais plutôt de "w" la largeur de la bande de la matrice.

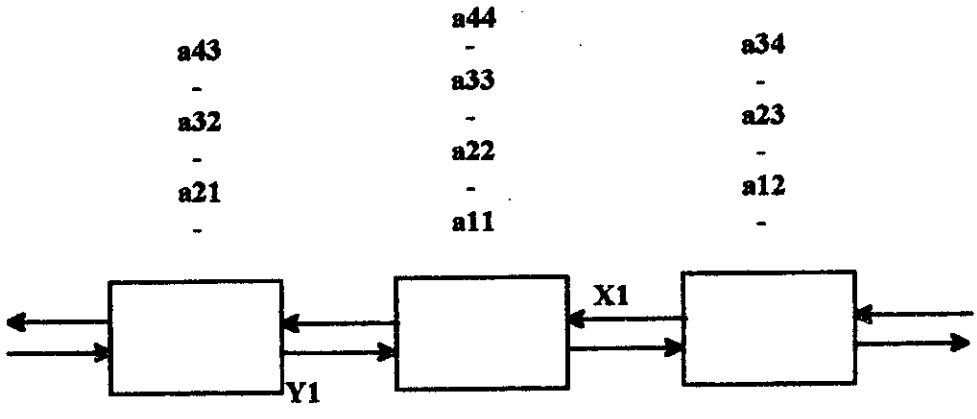


Figure 18 :

MULTIPLICATION MATRICE A STRUCTURE BANDE - VECTEUR
(n = 4) , (p = q = 2)

III-3-4 PRODUIT DE MATRICES DENSES

La multiplication des matrices est une opération usuelle . Pour celà nous avons opté pour deux solutions systoliques pour résoudre ce problème .

1- Première solution : réseau systolique rectangulaire.

2-Deuxième solution : réseau systolique carré.

A- RESEAU SYSTOLIQUE RECTANGULAIRE :

Soient A et B deux matrices dense d'ordre **n** . Nous allons décrire le réseau rectangulaire qui réalisera le produit matriciel :

$$\mathbf{A \,.\, B = C} \qquad \textbf{pour } \mathbf{n = 3}$$

- On juxtapose trois (03) cellules MAC , dont le fonctionnement est décrit par la figure 19 , pour former un réseau diagonal R_{diag} présenté par la figure 20 qui nous calculera C_{11}, C_{22} et C_{33}.

- On juxtapose ensuite deux (02) autres réseaux identiques à la droite de R_{diag} dans le but d'obtenir C_{12} , C_{13} et C_{23}.

- Puis , on juxtapose deux réseaux identiques à gauche de R_{diag} pour obtenir les éléments de la partie triangulaire inférieure du produit C.

Le réseau obtenu est décrit par la figure 21.

Notons , bien sûr , qu'un espace doit être inséré entre deux éléments consécutifs. Les éléments de la matrice C sont initialisés à zéro. Si une matrice C, non nulle, rentre dans le réseau , celui-ci effectuera l'accumulation :

$$\mathbf{C := C + A \,.\, B}$$

Un réseau de **(2n - 1) . n** cellules permet le calcul du produit , en un temps **(4n - 3)** que nous pouvons évaluer comme suit :

- C_{11} entre dans le réseau au **n** $^{\text{ième}}$ top.

- C_{nn} arrive à **2(n -1)** tops plus tard.

- **(n -1)** tops sont nécessaires pour compléter le calcul.

Notons qu'aprés **(4n - 3)** tops , les calculs sont terminés , néanmoins , il y a toujours certains coefficients qui continuent à circuler dans le réseau . D'autre part , rien ne se passe durant les **n** premiers tops , où a_{11} et b_{11} circulent à travers le réseau pour se rencontrer dans la cellule centrale .

Sachant que le produit séquentiel de deux matrices **n . n** se fait en n^3 tops , L'efficacité obtenue est :

$$e = \frac{n^3}{2n^2.4n}$$, pour **n** grand on aura **e = 1/8**

B- <u>RESEAU SYSTOLIQUE CARRE</u>:

Le réseau systolique carré qui est notre deuxième solution, est composé de cellules carrées, dont le registre interne est modifié par accumulations successives, jusqu'à l'acquisition de la valeur finale du produit.

Le fonctionnement est illustré par la figure 22.

Pour calculer un Cij, on utilise l'algorithme suivant:

$$C_{ij} = 0$$

pour k : = 1 a n

$$C_{ij} := C_{ij} + a_{ik} \cdot b_{kj}$$

La figure 23 nous décrit le calcul du premier élément C_{11}.

Le réseau obtenu est composé de n^2 cellules (une cellule par coefficient) comme nous le décrit la figure 24 (pour $n = 3$). Le calcul total du produit est effectué en ($3n - 2$) tops .

L'efficacité de ce réseau est :

$$e = \frac{n^3}{3n.n^2}$$

pour n grand on aura : $e = 1/3$

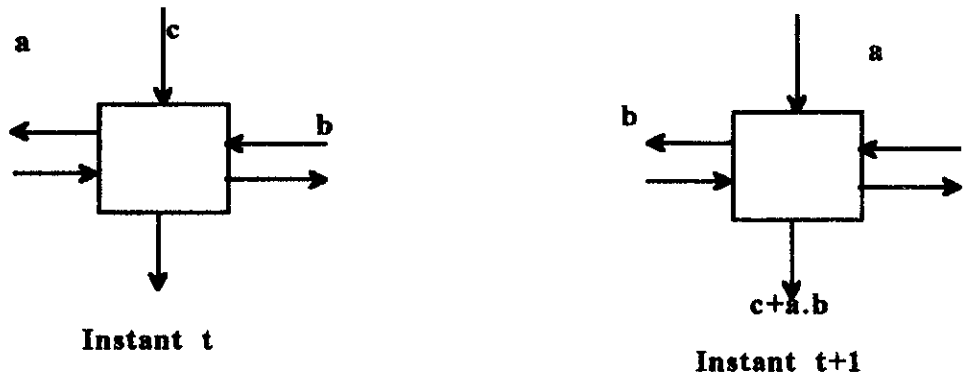


Figure 19: CELLULE MAC (première version)

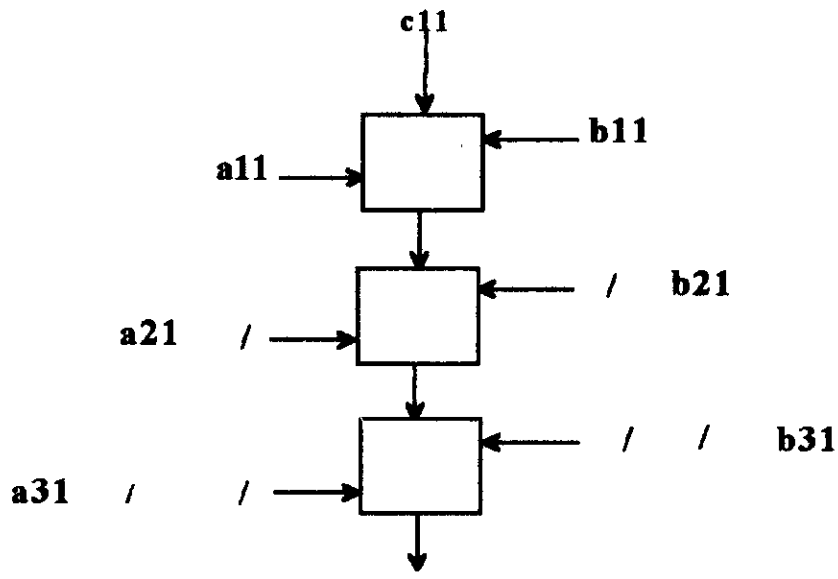


Figure 20: CALCUL D'UN PRODUIT SCALAIRE

(première version)

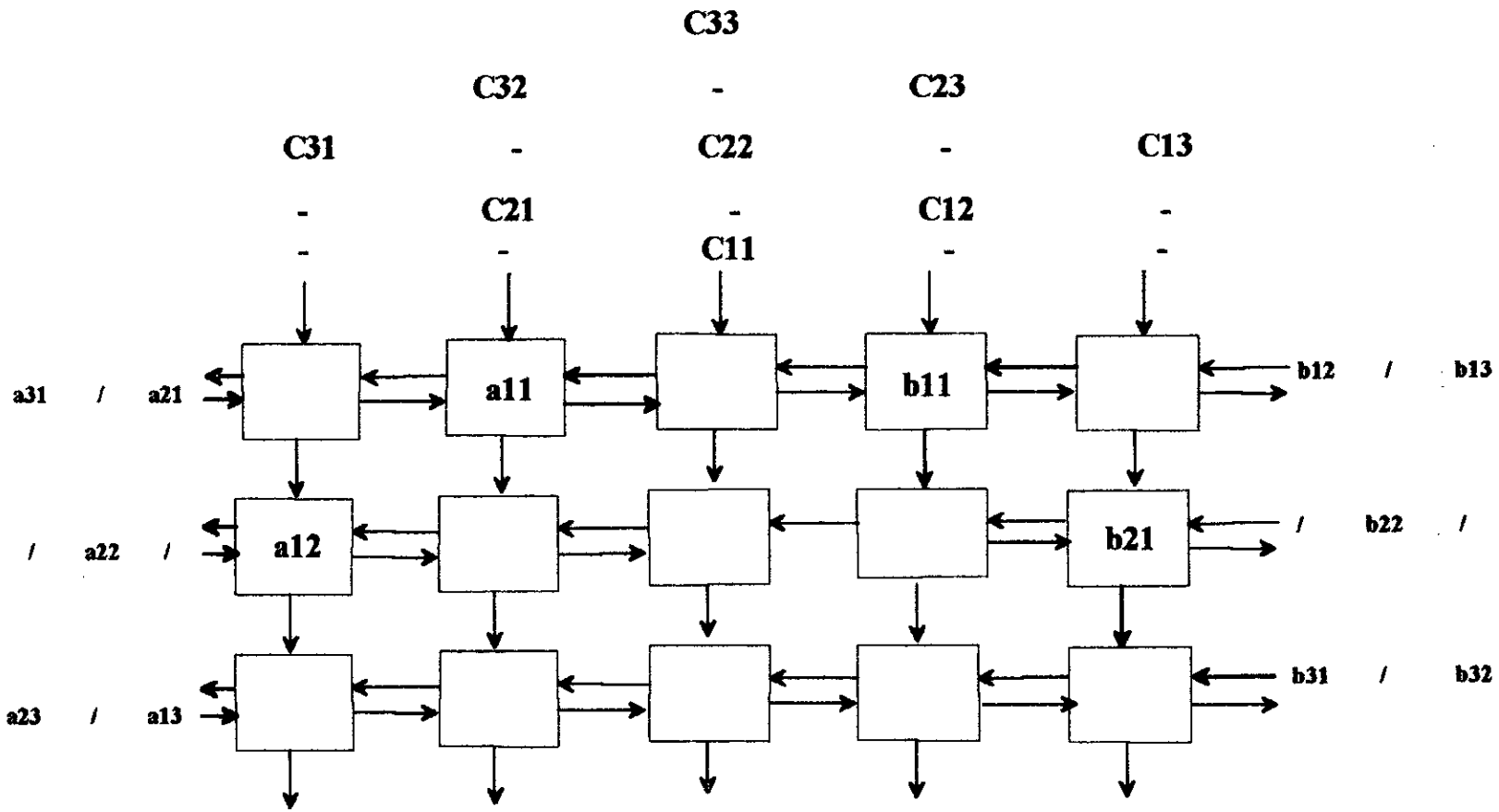


Figure 21: PRODUIT DE MATRICES DENSES DE TAILLE 3*3 (solution 1)

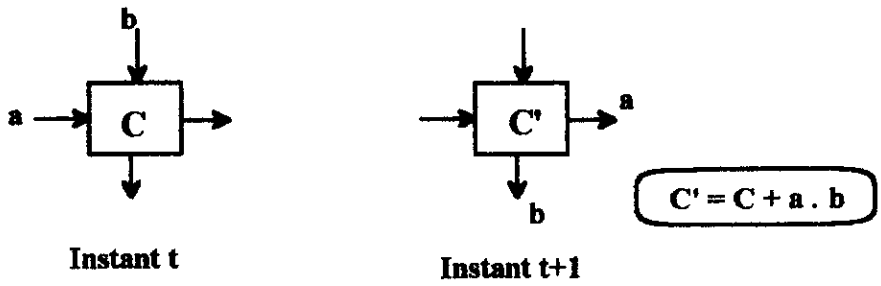


Figure 22: CELLULE MAC (deuxième version)

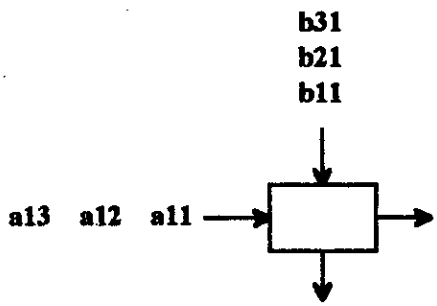


Figure 23: CALCUL D'UN PRODUIT SCALAIRE (deuxième version)

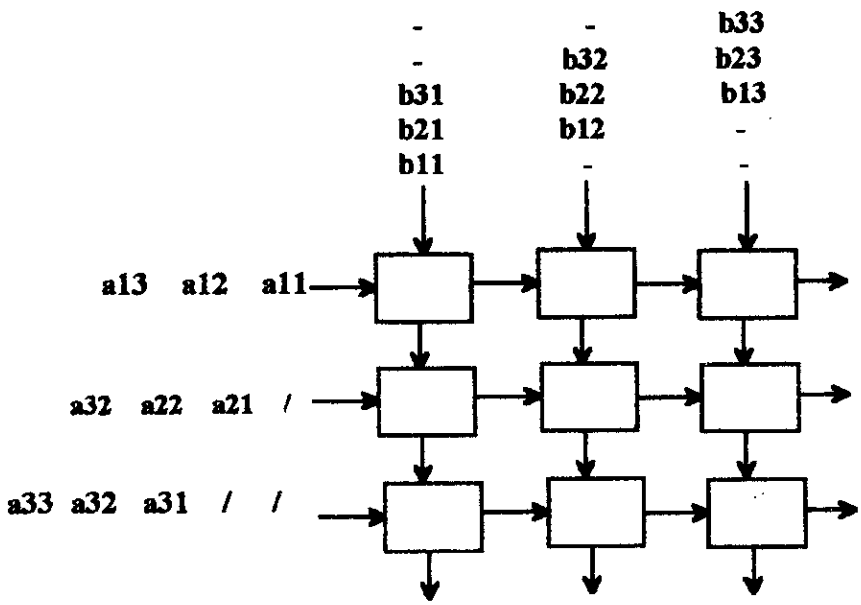


Figure 24: PRODUIT DE DEUX MATRICES DENSES DE TAILLE 3 * 3 (solution 2)

CHAPITRE 4

LE TRANPUTER

IV-1 INTRODUCTION

Une fois que les chercheurs sont arrivés à concrétiser l'idée du parallélisme dans différents modèles et architectures parallèles, le processeur conventionnel s'est averé inutilisable vu sa nature séquentielle entraînant le goulot d'étranglement. Les recherches se sont lancés, et ceci dans le but de trouver un nouvel élément adaptable aux réseaux parallèles avec un minimum de surface d'intégration.

Ce n'est qu'en Octobre 1985 que **BOB BARTON** est arrivé à concevoir le **TRANSPUTER** qui a été mis sur le marché par la compagnie INMOS.
Ce microprocesseur qui est conçu pour des systèmes hautement concurrent, est l'élément de base des systèmes complexes, à l'analogue du transistor dans les réalisations électroniques analogiques, ce qui traduit sa dénomination. C'est un **comPUTER** monoboitier et un composant silicium comme un **TRANsistor**.

IV-2 DEFINITION:

Le transputer est un composant VLSI programmable, orienté au traitement parallèle, conduisant de manière remarquablement simple à des performances élevées et à une adaptabilité extrême aux besoins des utilisateurs.

Son architecture interne révolutionnaire autorise l'implémentation de la concurrence des différentes architectures sur " puce".

Le transputer est un microprocesseur à 16 ou 32 bits avec un jeu d'instruction de type RISC (jeu d'instruction réduit), interconnectable à quatre autres transputers au moyen de canaux de communication série unidirectionnels intégrés appelés liens. Il peut par ailleurs accéder à des ressources propres ou partagées à l'aide d'un bus parallèle configurable conventionnel (figure 25). Ses performances sont de l'ordre de 10 MIPS (Million d'instructions par seconde) pour un processeur de 20 MHZ.

Le transputer peut être utilisé comme un microprocesseur conventionnel de haute performance, mais il est particulièrement destiné à la réalisation des réseaux de processeurs de topologies variées (quelques exemples sont illustrés par la figure 26).

La conception du tranputer est le résultat d'une collaboration étroite entre le concepteur du circuit et le créateur du langage occam .Ceci n'empêche pas son utilisation par d'autres langages parallèles tel que le **"C parallèle"** qui sera l'outil de notre simulation.

L'avantage des réseaux à base de transputer est d'autoriser l'obtention de performances notablement accrues et permettre la réalisation d'applications jusqu'ici difficilement imaginables.[1,3]

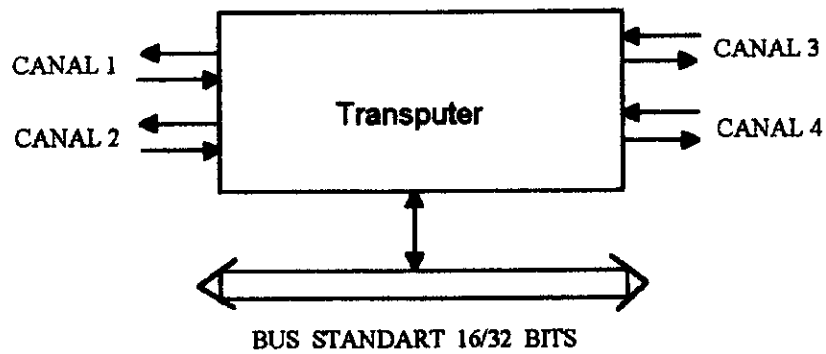


Figure 25: REPRESENTATION FONCTIONNELLE DU TRANSPUTER

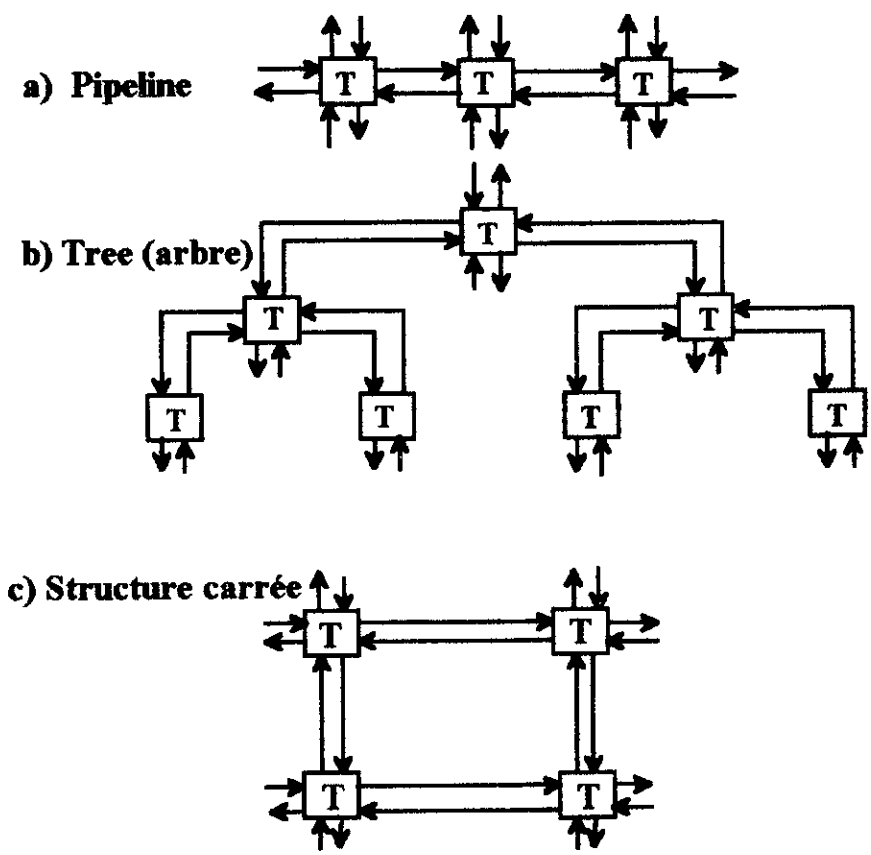


Figure 26: DIFFERENTES TOPOLOGIE A BASE DE TRANSPUTER

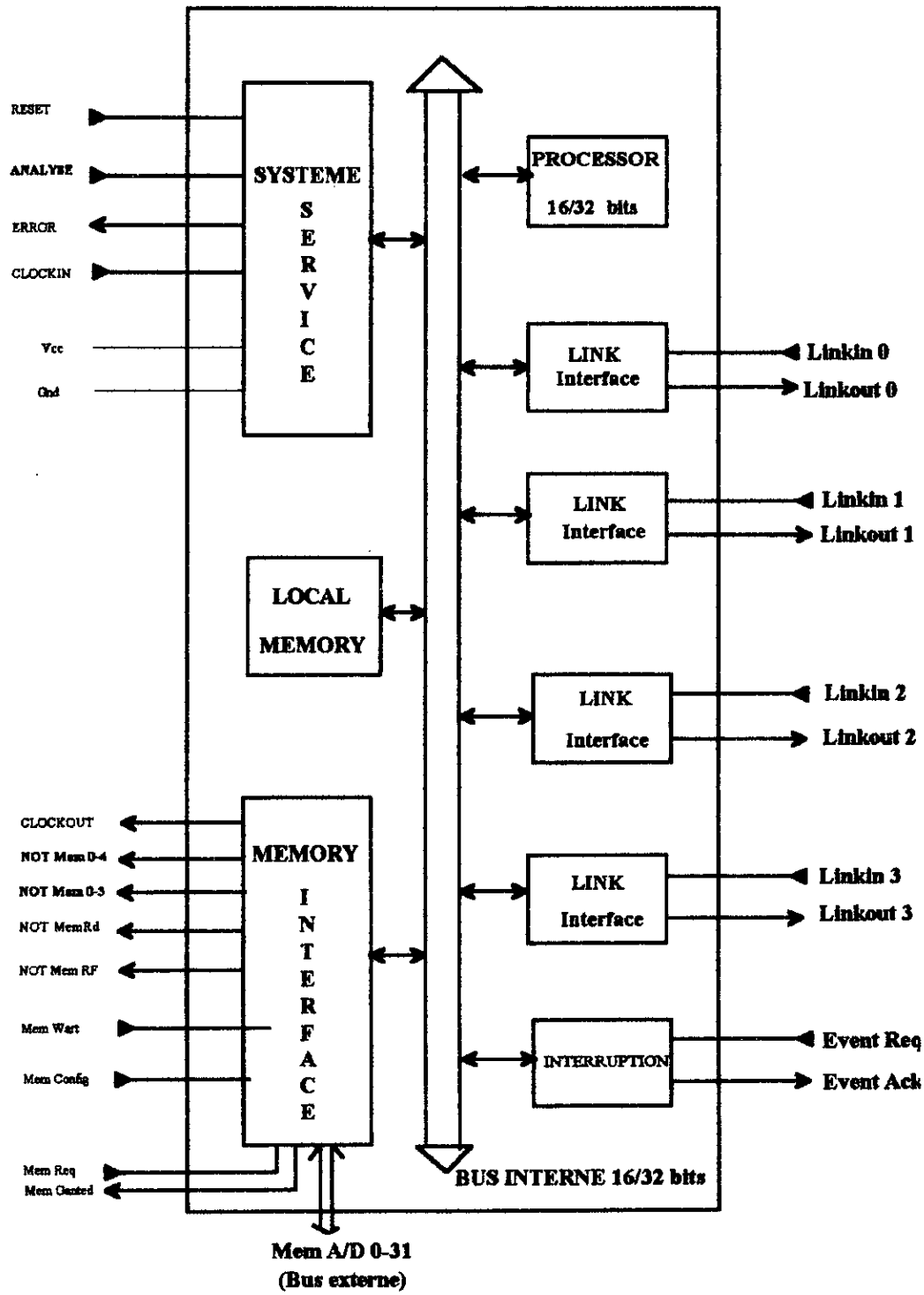


Figure 27: ARCHITECTURE DU TRANSPUTER

IV-3 STRUCTURE INTERNE DU TRANSPUTER:

Le transputer est un microprocesseur destiné aux systèmes concurrents de haut niveau. Ce produit qui a révolutionné le monde par ses utilisations rentables est un circuit qui contient (voir Figure 27) :

* Un processeur à 16/32 bits.

* Une mémoire locale.

* Des liens de communication par les liaisons point à point.

* Une interface mémoire programmable permettant de l'adapter à un usage particulier.

Notons que le transputer possède quatre liens de communication point à point, dont chacun est constitué de deux lignes unidirectionnelles, l'une vers le transputer (LinKin) et l'autre vers l'extérieur (LinKout), véhiculant des données et des bits de contrôles. Grâce à ces liens on peut obtenir aisément des réseaux parallèles de taille arbitraires. Un exemple simple est decrit par la figure28.

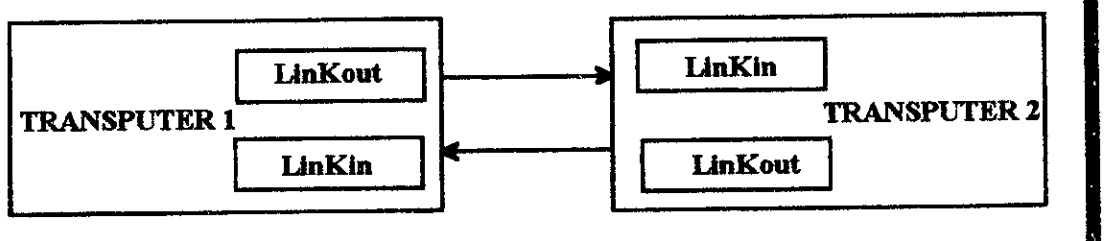


Figure 28: CONNEXION DES LIENS DE DEUX TRANSPUTERS

L'avantage des liens point a point par rapport au bus multiprocesseurs est multiple:

- L'utilisation n'a pas à se préoccuper du mécanisme de communication, celui-ci étant intégré au niveau du composant.

- La charge capacitive de chaque lien ne devient pas pénalisante lorsqu'on fait croître la taille d'un réseau.

- La bande passante des transferts n'est pas saturée lorsque le nombre de transputer augmente. Plus le nombre est grand, plus la capacité de transfert est grande, et, quelle que soit la taille du réseau, les liaisons entre transputers restent locales et courtes.

IV-4 PROTOCOLE DE COMMUNICATION:

L' implémentation physique des liens est telle que lurs utilisation devient évidente. La communication proprement dite est de type série avec une vitesse standard de 10 M bits/s. Cette vitesse est cependant programmable et peut aller de 5 M bits/s à 20 M bits/s selon le type du circuit. Elle ne dépend pas par contre des performances du processeur.

Cependant, la communication à travers des liens doit obéir à un simple protocole, qui permet un dialogue fiable entre les deux processeurs, en assurant une bonne synchronisation.

Notons, qu'avant la communication, il faut d'abord établir un lien entres les deux transputers. Ce lien qui est composé de deux liaisons électroniques unidirectionnelles, est le résultat d'une liaison matérielle entre deux interfaces liens des deux transputers comme nous l'a déjà décrit la figure 28.

Ainsi le protocole de communication permet la transmission séquentielle de bytes à travers une ligne de transmission.Chaque byte est composé d'un bit de départ (start bit) suivi d'un autre bit, d'un octet de données et, finalement de bit de fin (stop bit). Après émission complète du byte de données, l'émetteur attend l'arrivée d'un accusé de réception avant de continuer sa tâche. Ce dernier qui est envoyé par le récepteur, est constitué d'un bit start suivi d'un bit zéro (voir figure 29), et joue deux rôles:

1- il prouve que le récepteur a été capable de recevoir le message (information)

2- il informe la prédisposition du lien récepteur d'accepter un autre message.

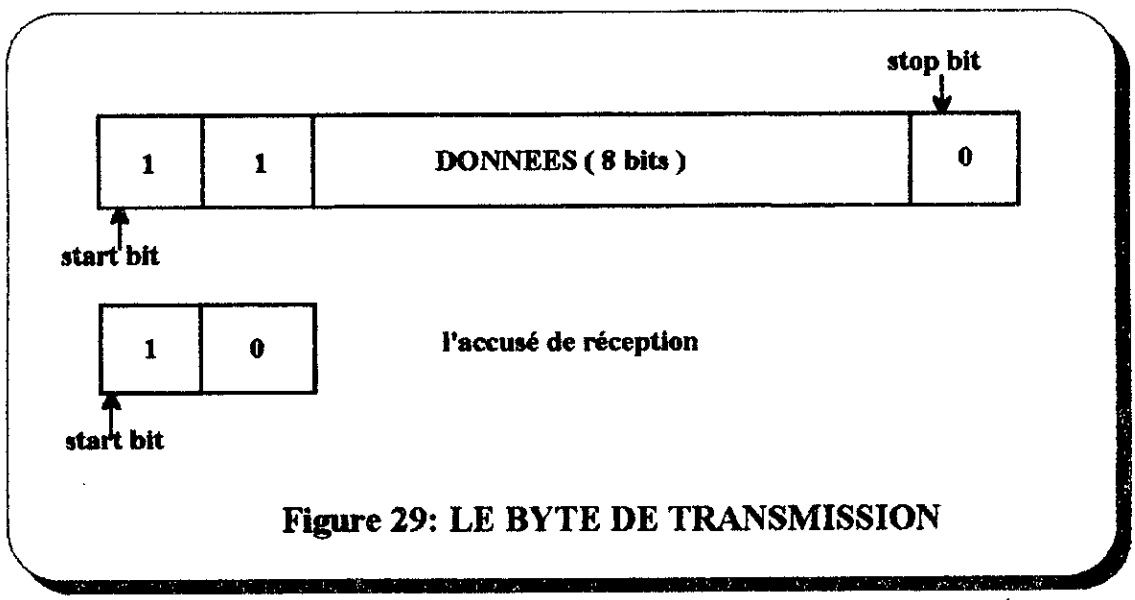
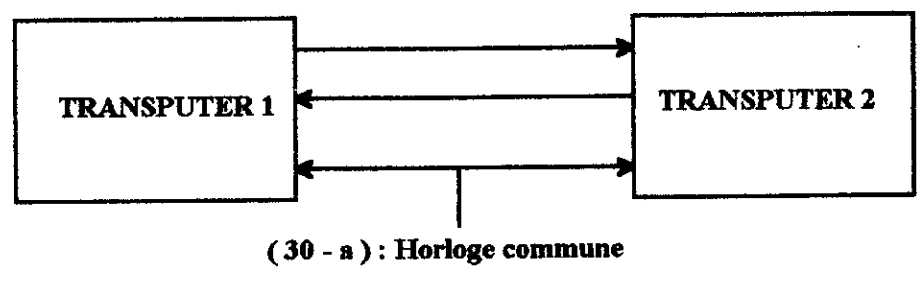


Figure 29: LE BYTE DE TRANSMISSION

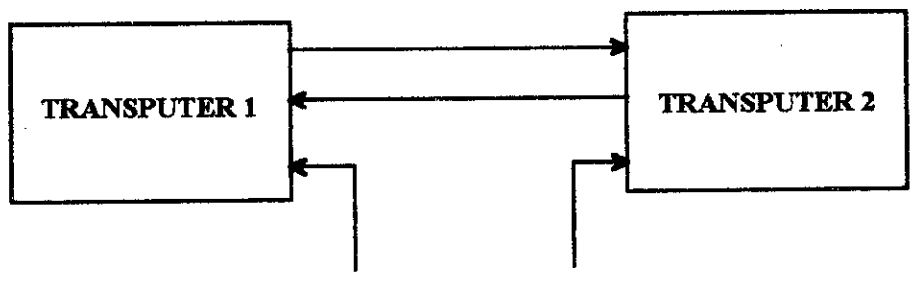
En conséquence, grâce à l'accusé de réception qui est envoyé dés que la réception d'un byte est engagée, la transmission est continue, et se fait sans délai entre les messages envoyés.

L'autre avantage de la communication par lien, est qu'elle n'est pas sensible à la phase de l'horloge du système.Ce qui permet d'assurer un échange d'informations par lien entre composant utilisant des horloges distinctes, à condition toutefois que la fréquence de communication soit la même. D' où les deux possibilités décrites par la figure 30.

L'intérêt de ces liens demeure dans le sens où ils permettent l'échange des informations entre des programmes différents s'exécutant en parallèle sur des transputers distincts.[1,8]



(30 - a) : Horloge commune



(30 - b) : Deux horloges de même fréquence

Figure 30: CADENCEMENT DES TRANSPUTERS

IV-5 LA CONCURRENCE DANS UN TRANSPUTER:

Le transputer a été conçu pour des réseaux parallèles de hat niveau. Néanmoins, une des plus fortes caractéristiques qui particularise le transputer vis-à-vis du processeur conventionnel, est son autorisation de la concurrence. C'est à dire, que dans un seul transputer deux ou plusieurs processus peuvent être exécutés en parallèle. Ces processus forment un système qu'on appel **"système concurrent"**. Un exemple est illustré par la figure 31.

Un processus est une tâche élémentaire et autonome, avec ses propres données et ses propres opérations, et qui peut communiquer à l'aide des canaux avec d'autres processus s'exécutant en même temps.

Un des avantages de la concurrence sur un transputer et que ce dernier nous permet de simuler un réseau parallèle complexe, en utilisant le nombre nécessaire de processus, et cela par défaut de manque de transputers ou pour autre raison.

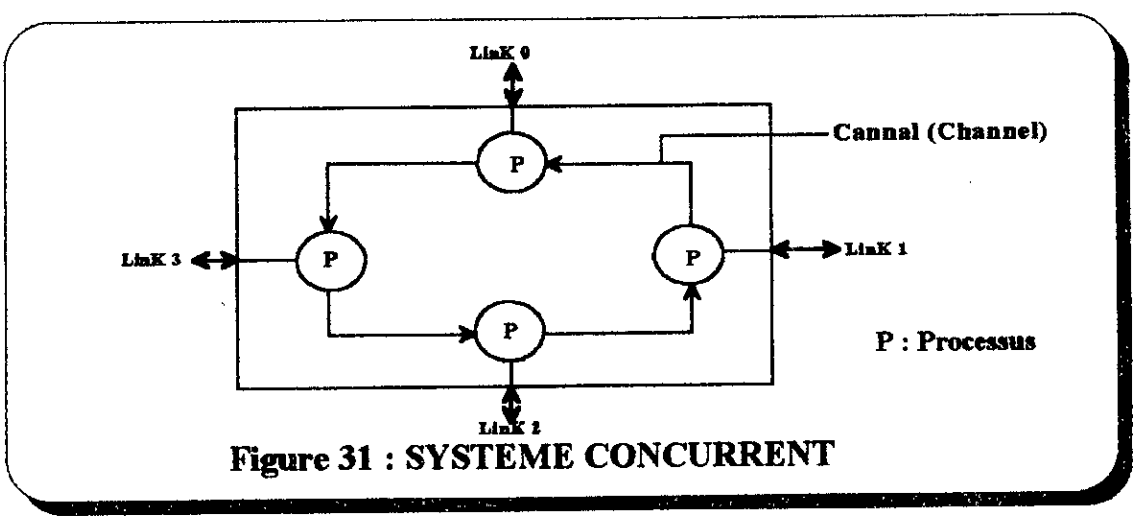


Figure 31 : SYSTEME CONCURRENT

IV-6 APPLICATION DES TRANSPUTERS:

Les produits réalisés à partir de transputer sont multiples. Ils vont de la simple carte d'extension aux machines complètes à base de transputers. La carte d'extension est destinée à être insérée dans un micro-ordinateur classique, et va se comporter comme un processeur spécialisé vis-à-vis de ce dernier. Elle permet de s'initier au multiprocessing et aux langages parallèles comme l'occam ou le "C parallèle", et ceci à partir d'un environnement de programmation classique. Ces extensions peuvent être utilisées dans le but d'augmenter les performances du micro-ordinateur. Pour ce qui est des machines à base de transputer, il existe plusieurs types pour diverses applications.Certains de ces dispositifs sont destinés aux traitement d'image et la synthèse d'image, d'autrse à la simulation pour la conception des circuits intégrés, comme on trouve des applications dans le traitement de signal. Récemment on a vu l'entrée de l'intelligence artificielle dans les applications transputers. [2,3]

CHAPITRE 5

LE "C PARALLELE"

V-1 MODELE ABSTRAIT:

Le modèle parallèle dans le transputer est basé sur l'idée du CSP (communicating sequential processes) developpée par le professeur C.A.R. Hoare.

Le CSP est un modèle général abstrait de la concurrence, basé sur l'idée de processus indépendants qui dialoguent entre eux par l'échange des données à travers des cannaux.

Le traitement concurent dans le "C parallèle" est conforme au modèle CSP où les processus sont indépendants, peuvent être emboités avec d'autres, et sont liés entre eux par des cannaux.

La figure (32) illustre les principaux éléments du modèle CSP. Les processus peuvent être emboités dans un autre, et peuvent communiquer à l'aide d'une voie unidirectionnelle pour le passage des données, ou de deux voies unidirectionnelles pour l'échange des données. [12]

V-1-1 PROCESSUS:

Les processus sont les éléments principaux d'un modèle CSP.Un processus est une entité autonome capable d'éffectuer une série d'opérations. Une application peut être partagée en un nombre donné de processus. Ces derniers peuvent être appliqués sur un réseau donné de transputers.

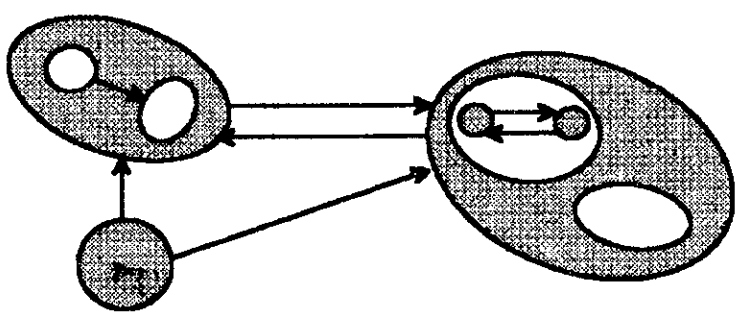


Figure 32

V-1-2 CANNAUX:

Les cannaux sont des connections entre processus, à travers lequels des informations et des données sont échangées. Chaque connection point à point relie seulement deux processus, où le transfert de données se fait sur une ligne unidirectionnelle. L'échange de données entre deux processus, exige une paire de cannaux.

Dans le modèle CSP, les cannaux ont deux fonctions:

* ils assurent la communication entre deux processus s'**exécutant indépendants**.

* ils assurent la communication synchronisée entre ces deux processus.

V-2 CHOIX D'UN LANGAGE PARALLELE:

La programmation parallèle pose deux problèmes dont le langage doit tenir compte:

* Le découpage d'un programme en tâches concurrentes, partageant des informations.

* La réception des tâches entres les diverses unités de traitement de la machine.

Le premier langage du transputer qui approche de manière très élégante ces problèmes, est le langage occam qui a été conçu en 1982 par May. Ce langage est basé sur trois processus primitifs: entrée, sortie, assignation.

ENTREE:

EST ! message

signifie que la valeur de la variable message est émise sur le canal "EST".

SORTIE:

OUEST ? message

signifie qu'une valeur est reçue par le canal OUEST et est stockée sous le nom de message.

ASSIGNATION: (classique de la forme)

variable : = expréssion

A partir de ces trois processus primitifs et de leurs combinaisons, on peut construire tous les autres.

Le langage Occam est le plus proche de l'architecture du transputer. Il est considéré comme le langage assembleur de celui-ci. Mais pour notre simulation, nous avons choisi un autre langage parallèle qui est évolué et qu'on nomme le **"C parallèle"**. [1, 3, 8]

V-3 LE "C PARALLELE":

Le "C parallèle " a tiré un tas d'avantages du transputer pour son caractere concurrent et du langage Occam pour son haut niveau d'implémentation du modèle CSP. La concurrence est supportée par l'extension des librairies qui vont contenir deux nouveaux types de données et un ensemble de fonctions. Les nouvelles fonctions sont obtenues de la même manière que toutes les fonctions du C en incluant à l'entête du programme le fichier correspondant . **h** avec la directive correspondante "include". Les fonctions processus et cannales sont déclarées par deux fichiers . **h** différents.

V-3-1 LES NOUVEAUX TYPES DE DONNEES:

On a deux types de données:

Process:de type structure, il réserve les informations de chaque processus déclaré.

Channel: de type pointeur, il est utilisé pour implémenter les cannaux . chaque variable cannal, représente une communication unidirectionnelle entre deux processus. Channel est un pointeur de type Voïd.

V-3-2 LA PROGRAMMATION "C PARALLELE":

On retrouve deux méthodes:

a) - Programmation avec configuration

La confuguration est constituée d'un réseau Software représenté par des processus joints par une déscription adaptative (Mapping), à un réseau Hardware, représenté par des processeurs. Cette méthode est recommandée dans le cas où on dispose d'un réseau de transputer.

b) - Programmation sans confuguration

On retrouve un groupe de processus qui communiquent à travers des cannaux, et un programme principal où toutes les allocations mémoires sont faites avant l'exécution du programme. Cette méthode est utilisée dans le cas d'une programmation sur un seul transputer (cas de notre simulation). Ce sera l'objet du chapitre suivant.

V-4 PROGRAMMATION SANS CONFIGURATION:

Les processus parallèles sont crées en liant les fonctions définies aux structures processus prédéclarées, puis exécutés en utilisant les instructions adéquates.

Les cannaux entre processus, sont créés simplement en déclarant des variables de type Channel * dans le programme. Les cannaux d'entrées et de sorties sont utilisés pour le passage des données. C'est la responsabilité du programme de s'assurer que les données envoyées par le processus soient reçues par l'autre. Des fonctions séparées pour les émissions et réceptions existent et doivent être utilisées pour la communication entre les deux processus.

Les fonctions principales dans le traitement parallèle sont déclarées dans les fichiers **process . h** et **channel . h**

Le paragraphe si dessous, illustre les fonctions processus et cannales.

V-4-1 PROCESSUS:

Les processus sont définis de la même manière que toutes les fonctions du C, mais avec un premier paramètre fixe. Ce premier paramètre est un process pointeur déclaré dans sa propre structure processus. Les paramètres de la fonction processus qui suivent le processus pointeur sont déclarés de façon normale.

Les processus sont installés par un appel de la fonction **ProcAlloc**, en utilisant le nom de la fonction comme un lien à la structure processus. Une fois installé, le processus est entamé, en utilisant les fonctions **ProcRun**, **ProcPar** ou **ProcPripar**.

Notons que le compilateur génère un message d'avertissement indiquant l'inutilisation d'un process pointeur, chaque fois qu'il est passé à une fonction. Pour éviter la génération d'un tel message, le process pointeur doit être assigné à lui même à l'interieur de la fonction, utilisant l'instruction **p = p** .

ProcAlloc réserve un éspace mémoire pour le processus. Cette fonction possède les paramètres suivants:

- **void (* func)** : la fonction processus de type void.

- **int size** : on retrouve deux paramètres de type entier.

a) - Le premier est déstiné à l'éspace mémoire . La valeur zéro indique un éspace mémoire de 4 K pour 32 bits et 1 K pour 16 bits. En cas d'insuffisance, on utilisera la valeur adéquate.

b) - Le deuxième est réservé au nombre de paramètres que possède la fonction processus.

- **int param** : on retrouve les paramètres de la fonction processus.

Tout appel à la fonction **ProcAlloc** doit être suivis d'un test d'allocation. En cas d'echec, le résultat NULL est envoyé.

Notons que tous les processus doivent toujours être alloués avant toute utilisation.

A. L'EXECUTION PROCESSUS/

Un ensemble de fonctions est prévu pour l'exécution des processus asynchrones (**ProcRun**) ou synchrones (**ProcPar** et autre).

Fonctions	Paramètres
void ProcKun	(Process * P)
void ProcPar	(process * P1, Process * 2,..)
void ProcPriPar	(Process * P high, process * plow)

ProcRun exécute des processus indépendants (asynchrones). Tandis que **ProcPar** est souvant utilisé pour forcer la synchronisation d'un groupe de processus en les exécutant à priorité concurrente.

Notons que **ProcPar** peut être utilisé pour des processus indépendants. **ProcPriPar** exécute deux processus synchronisés avec priorité. Le premier processus possède la priorité supèrieure.

V-4-2 COMMUNICATION CANNAL:

Des routines sont prevues pour:

- le passage de bytes et des entiers dans des cannaux (**ChanIn**, **ChanOut** et d'autre).
- l'implémentation en sureté du protocole de communication.
- l'allocation et la représentation des cannaux.

La communication entre processus est éffectuée par le passage des données dans des variables de type Channel. Des fonctions sont prevues pour l'allocation et l'emission des données de type entier, caractère ou tout autre. Ainsi un protocole de communication fiable est établi. Un cannal d'entrée et un autre de sortie doivent être utilisés par couple pour une communication et un échange de données entre deux processus.

V-2-1-a ALLOCATION CANNAL ET INITIALISATION:

Fonctions	Paramètres
Channel * ChanAlloc	( )
void ChanInit	(Channel * C)

ChanAlloc réserve un éspace mémoire pour le cannal, puis l'initialise. Si l'éspace mémoire ne peut pas être alloué, **ChanAlloc** renvoie le message NULL. Tout appel à **ChanAlloc** doit être suivi d'un test d'allocation.

ChanInit : initialise le cannal.

Notons que les cannaux entre processus exécutés sur le même transputer doivent toujours être loués avant toute utilisation.

V-4-2-b CANNAUX D'ENTREE ET SORTIE:

Fonction	paramètres
void ChantOut	(Channel * C, void * cp, int cnt)
void ChanIn	(Channel * C, void * cp, int cnt)
void ChanOutChar	(Channel * C, Char ch)
void ChanInChar	(Channel * C)
void ChanOutInt	(Channel * c, int n)
int ChanInInt	(Channel * C)

ChanOut et **ChanIn** passent des données de type quelconques et de dimenssions quelconques, mais le nombre de bytes doit être spécifié (dans le troisième paramètre).

ChanOutChar et **ChanInChar** sont similaires aux précédents, sauf qu'ils font passer uniquement des caractères. Le nombre de bytes n'est pas demandé.

ChanOutInt et **ChanInInt** sont aussi similaires sauf qu'ils font passer uniquement des entiers. [12]

V-4-3 EXEMPLES DE PROGRAMMES PARALLELES:

Dans ce paragraphe trois (03) exemples sont présentés et ce dans le but de démontrer comment :

* exécuter en parallèle des processus.
* communiquer entre deux processus à travers des cannaux synchronisés .
* faire passer des données d'un processus à un autre.

a- Exemple 1: **Processus parallèles asynchrones.**

Cet exemple montre comment déclarer et exécuter des processus parallèles avec le "C parallèle" . **ProcWait** (10000) qui est une routine qui produit un temps mort avant l'exécution du processus, est utilisée dans le but de montrer l'indépendance des processus.

```
# include  <stdio . h>
# include <stdlib . h>
# include <process . h>

void hello (Process *p)
 {
     p = p;
      ProcWait (10000) ;
      printf (" \nHello, \n") ;
 }

void world (Process *p)
 {
      p = p;
      printf (" \nWorld\n") ;
 }

int main ( )
 {

       Process *pl, *p2;

       p1  =  ProcAlloc (hello, 0, 0) ;
      if  (p1 == NULL)
       abort ( ) ;
      p2  =  ProcAlloc (world, 0, 0) ;
      if  (p2 == NULL)
       abort ( ) ;

      ProcPar (p1, p2, NULL) ;
 }
```

L'exécution du parogramme nous donne: **World**
Hello ,

b- Exemple 2: **Processus synchronisés par un cannal**

Cet exemple montre comment les deux processus de l'exemple 1 peuvent être synchronisés en utilisant un cannal. L'utilisation d'un cannal force le processus world d'attendre l'emisssion complète du processus Hello. Ce qui crée une dépendance entre les deux processus. Le cannal lie les deux processus sans communiquer aucune donnée.

```
# include <stdio.h>
# include <stdlib.h>
# include <process.h>
# include <channel.h>

void hello (Process *p, Channel *ready)
 {
   p = p;
   ProcWait (10000) ;
   printf (" \nHello, \n") ;
   ChanOutInt (ready, 1) ;
 }

void world (Process *p, Channel *ready)
 {
   p = p;
   ChanInInt (ready) ;
   printf (" \nWorld \n") ;
 }

int main ( )
 {

   Process *p1, *p2;
   Channel *ready;

   ready = ChanAlloc ( ) ;
   if (ready == NULL)
     abort ( ) ;

   p1 = ProcAlloc (hello, 0, 1, ready) ;
   if (p1 == NULL)
     abort ( ) ;
   p2 = ProcAlloc (world, 0, 1, ready) ;
   if (p2 == NULL)
     abort ( ) ;

   ProcPar (p1, p2, NULL) ;

 }
```

L'exécution du programme nous donne:

```
Hello,
World
```

c- *EXEMPLE 3:* **Communication de données à travers un cannal.**

Cet exemple montre comment deux processus peuvent synchroniser leurs comportements et communiquer des données en utilisant des cannaux.

Dans cet exemple le processus input lit du clavier un certain nombre de caractères (<20), grâce à la fonction gets, puis transmet ce message à travers le cannal **ChanOut** (chan). Le processus display reçoit ce message à travers le cannal d'entrée **ChanIn** (chan) puis l'affiche sur l'écran.

```
# include <stdio.h>
# include <string.h>
# include <stdlib.h>
# include <process.h>
# include <channel.h>

void input (Process *p, channel *chan)
 {
   char message [20];

   p = p;
   printf (" \nplease type your name (20 letters max) : ");
   gets (message) ;
   ChanOut (chan, message, 20) ;
 }

void display (Process *p, Channel *chan)
 {
   char name [20] ;

   p = p;
   ChanIn (chan, name, 20) ;

   printf (" \nHello %s \n", name) ;

 }

int main ( )
 {

   Process *p1, *p2;
   Channel *chan;

   chan = ChanAlloc ( ) ;
   if (chan == NULL)
     abort ( ) ;

   p1 = ProcAlloc (input, 0, 1, chan) ;
   if (p1 == NULL)
     abort ( ) ;
   p2 = ProcAlloc (display, 0, 1, chan) ;
   if (p2 == NULL)
     abort ( ) ;

   ProcPar (p1, p2, NULL) ;

 }
```

V-4-4 EXECUTION D'UN PROGRAMME EN "C PARALLELE":

Après avoir établi notre programme source, on doit passer à l'étape d'exécution; Pour cela , nous suivons les étapes citées ci dessous:

1- Compiler le programme source:

Cette compilation se fait avec l'instruction suivante:

icc nom . c / t 425

2- LinKer l'unité compilée:

On linK en utilisant l'instruction suivante:

ilink nom . tco /f startup . lnk /t 425

3- Configuration du programme:

Dans le cas où, nous avons un réseau de transputers, la configuration doit être conçue dans le but de linker ces derniers. L'instruction adéquate est:

icconf nom . cfs

4- Génération du fichier exécutable:

Le fichier généré après utilisation de **"icconf"** est lu par une instruction :

icollect nom . cfb

Cette instruction nous génère le fichier exécutable **(btl)**.

Dans le cas d'un seul transputer, le fichier exécutable est généré à partir du fichier linké, en utilisant l'instruction:

icollect nom . lku /t

5- Exécution du programme:

Le fichier exécutable est chargé et est exécuté sur transputer en utilisant l'instruction:

iserver /sb nom . btl

Une fois chargée, l'exécution est entamée immédiatement mais la communication avec le Host est toujours conservée. [12]

La figure 33 illustre les différentes étapes du développement du programme en "C parrallèle".

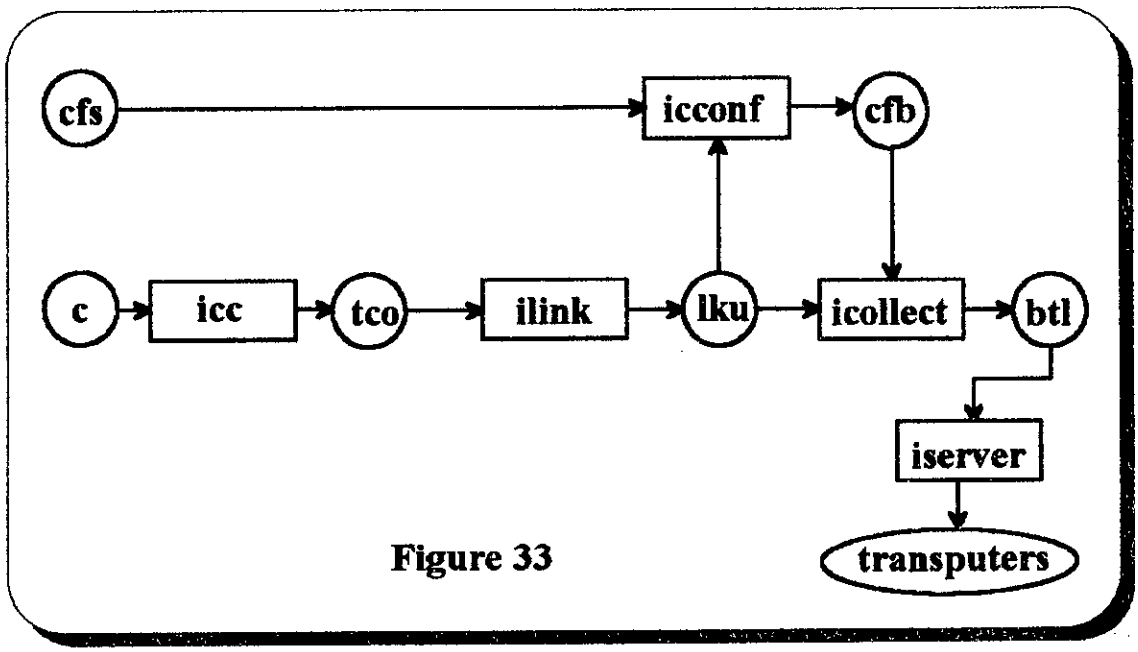


Figure 33

V-5 LOGICIEL UTILISE:

Le logiciel que j'ai utilisé au niveau du laboratoire de recherche de l'INELEC pour ma simulation est le ANSI C tools et installé sur PC. L'utilisation raissante de ce logiciel m'a affronté à de nombreux problèmes. Parmis ces problèmes la programmation avec configuration qui est très interressante, mais jusqu'ici non réalisable.

Grâce à la programmation sans configuration, on est arrivé à utiliser ce logiciel pour une simulation qui se fera sur le transputer (t 425)

CHAPITRE 6

SIMULATION

VI-1 INTRODUCTION

Après avoir élaborer cette modeste étude sur :

- les stuctures systoliques et caractéristiques
- quelques algorithmes sur des réseaux systoliques
- Le transporteur : qu'on va utiliser pour simuler quelques applications
- Le " C parallèle " : l'outil de notre programmation.

Nous arrivons à l'étape finale : **simulation** . Pour celà nous avons choisi deux applications .

Une application numérique : **La mutiplication matricielle**.

Une application non - numérique : **Le tri des nombres.**

Avant de passer à la simulation , nous allons traiter un programme parallèle classique appelé **Producteur Consommateur**.

VI-2 Premier programme : Producteur Consommateur

Ce programme élémentaire montre comment se fait la communication entre processus.

Il possède deux processus et un cannal .

Premier processus : Le processus producteur demande à lire une donnée (chiffre)grâce à la fonction "scanf" qu'il transmettera à travers le cannal de sortie "chan" vers le processus consommateur.

Deuxième processus : Le processus consommateur qui va reçevoir cette donnée à travers le cannal d'entrée "chan" , la transmettra vers l'écran pour l'affichage et ceci grâce à la fonction " printf".

VI-3 Deuxième programme : Produit de deux matrices denses.

Pour notre simulation , nous avons chosi un réseau systolique carré pour effectuer le produit **A . B = C** pour n = 2.

Notre programme possède cinq (05) processus :

1- un processus master : son rôle est de reçevoir les données du clavier, puis de les transmettre vers les quatres autres processus.

2- quatre processus identiques : job1 , job2 , job3 , job4 :

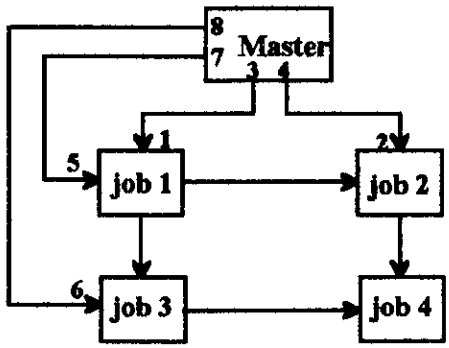
Chaque processus effectue les opérations suivantes :

- Il reçoit des données à travers deux cannaux d'entrée .
- Il effectue :
  - * une multiplication de deux données.
  - * une accumulation avec le résulta précédent.
- Il passe les données reçues aux processus voisins à travers les cannaux de sortie correspondants.

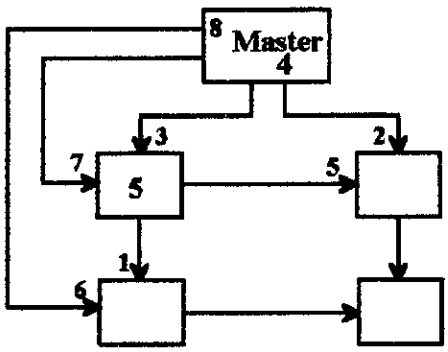
L'exemple suivant illustre le fonctionnement .

soient de matrices denses **A** et **B** tellles que:

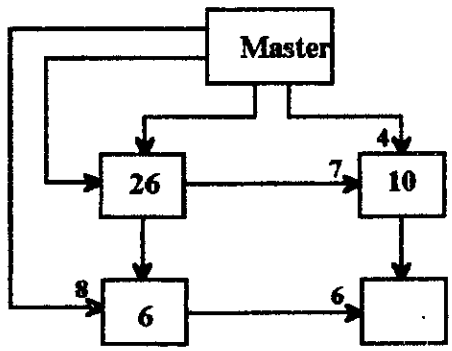
$$A = \begin{pmatrix} 5 & 7 \\ 6 & 8 \end{pmatrix} \qquad \text{et} \qquad B = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$


8
7
Master
3
4
1
2
5
job 1
job 2
6
job 3
job 4

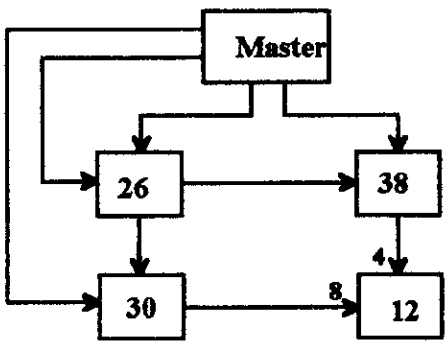
(1)


8
Master
4
3
2
7
5
5
1
6

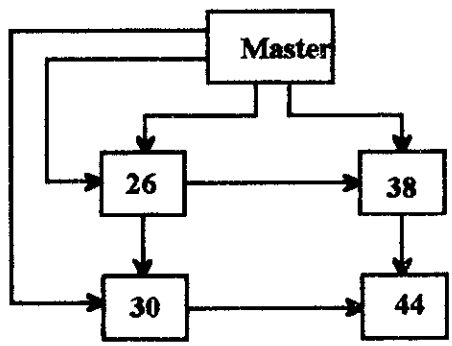
(2)


Master
4
26
7
10
8
6
6

(3)


Master
26
38
4
30
8
12

(4)


Master
26
38
30
44

(5)

VI-4 Troisième programme: Tri des nombres par ordre croissant.

Notre deuxième simulation est le tri des nombres par ordre décroissant. Une structure pipeline décrite par la figure (34) est la solution du problème.

Une structure pipeline pour le tri de cinq valeurs

Figure : 34

L'Algorithme est illustré ci-dessous :

Pour ordonner un nombre M de valeurs , il faut une structure à (M-1) processus.

Ces valeurs sont transmises à partir du clavier (Host) vers la structure pipeline , où chaque processus possède deux variables R et S. La première donnée qui rentre dans chaque processus est affectée à la variable R , tandis que toutes les autres sont affectées à la variable S.

Chaque valeur affectée à la variable S , subit une comparaison à celle qui est affectée à la variable R.

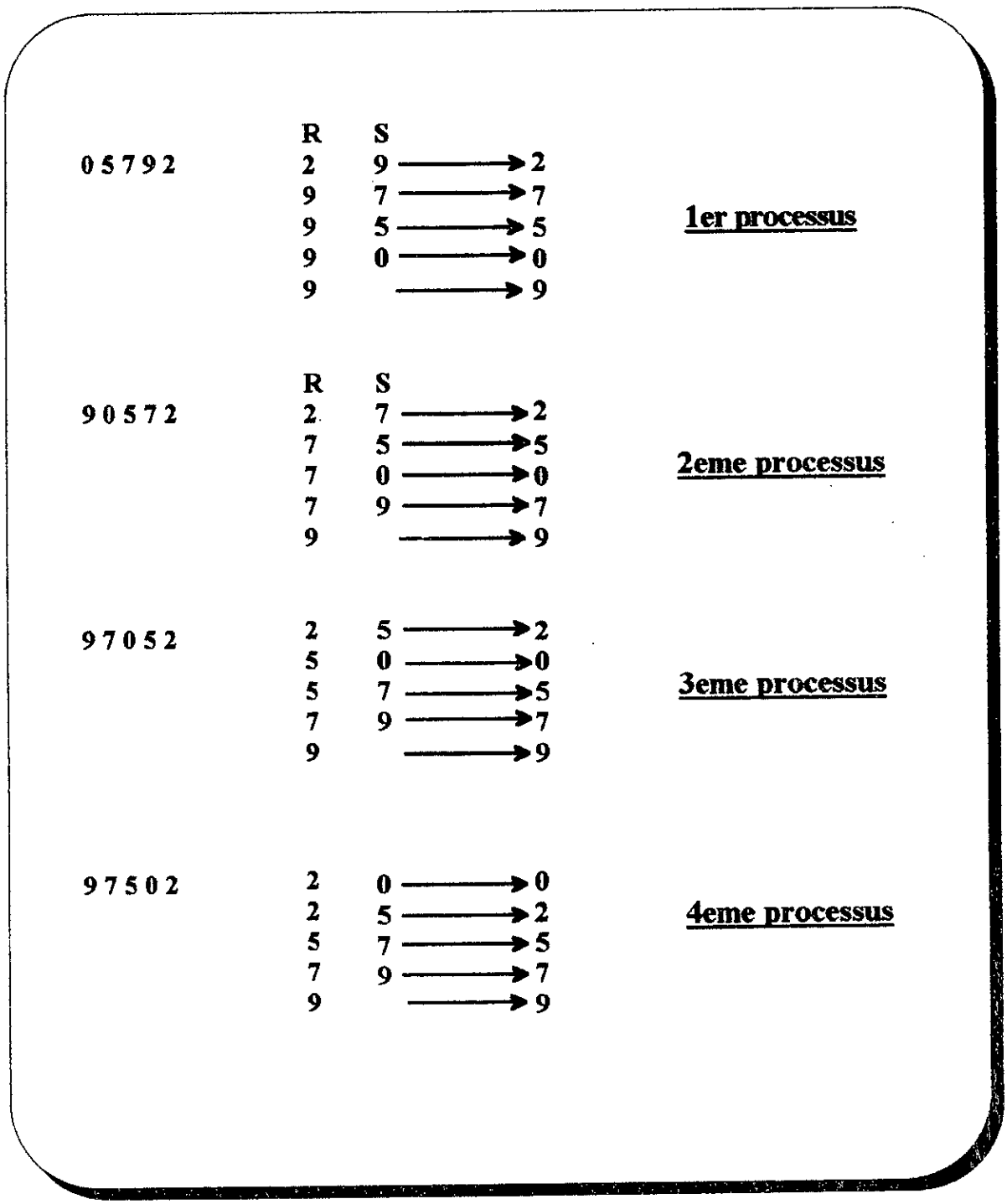
- Si S < R ==> la valeur de S va être transmise au prochain processus.
- Si S > R ==> la valeur de S sera affectée à R , et l'ancienne valeur de R sera transmise au prochain processus.

Quand toutes les valeurs ont traversé le processus , la valeur finale de R sera transmise à son tour.

Avec cette méthode on aura un tri croissant tel que :

- Dans le premier processus , la plus grande valeur ira à la dernière position dans le tri.
- Dans le second processus , la deuxième grande valeur ira à la position avant dernière du tri , et ainsi de suite.

L'exemple suivant illustre le mode d'évolution pour n = 5



```
/*************************************************************************** */
/*                TITRE :   " PRODUCTEUR-CONSOMATEUR "                       */
/*       ********************************************                        */
/*            CE PROGRAMME POSEDE DEUX PROCESSUS:                            */
/*LE PREMIER PROCESSUS RECOIT UNE DONNEE DU CLAVIER,PUIS IL L'A TRANSMEE     */
/*       A TRAVERS LE CANNAL "CHAN"                                          */
/*LE DEUXIEME PROCESSUS RECOIT LA DONNEE A TRAVERS LE CANNAL "CHAN",PUIS     */
/*       IL L'A TRANSMEE VERS L'ECRAN                                        */
/*                L'OUTIL DE PROGRAMMATION EST:                              */
/*                 *******************************                           */
/*                        LE "C PARALLEL"                                    */
/*                       *****************                                   */
/*************************************************************************** */
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<process.h>
#include<channel.h>
  void producteur (Process *p,Channel *chan)
{
    int i ;
    p=p;
     printf("S'IL VOUS PLAIT ENVOYER UNE VALEUR :");
    scanf("%d",&i);
    ChanOutInt(chan,i);
}
  void consommateur (Process *p,Channel *chan)
{
    int s ;
    p=p;
    s=ChanInInt(chan);
     printf("LA VALEUR RECUE EST :%d\n",s);
}

    int main()
     {
        Process *p1 , *p2;
        Channel *chan;
          chan=ChanAlloc();
           if (chan==NULL) abort();
          p1=ProcAlloc(producteur,0,1,chan);
           if (p1==NULL) abort();
          p2=ProcAlloc(consommateur,0,1,chan);
           if (p2==NULL) abort();
          ProcPar(p1,p2,NULL);
     }
```

```
/*****************************************************************/
/*       CE PROGRAMME  REALISE LA MULTIPLICATION DE DEUX MATRICES  N=2     */
/*                                  A*B=C                                  */
/*                       L'OUTIL DE PROGRAMMATION EST:                     */
/*                     ***********************************                 */
/*                             LE  "C PARALLELE"                           */
/*                           ********************                          */
/*****************************************************************/
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<process.h>
#include<channel.h>
static int second1(int,int,int);
static int second2(int ,int ,int);
static int second3(int ,int ,int);
static int second4(int ,int ,int);

void master(Process*p,Channel*chan1,Channel*chan2,Channel*chan3,Channel*chan4
    {
       int i ,x ,y ,z ,w;
       p=p;
        for(i=0 ; i<=1 ; i++ )
         {
              printf("DONNER LES b ET LES a :\n");
           scanf("%d%d%d%d",&x,&y,&z,&w);
           ChanOutInt(chan1 ,x);
           ChanOutInt(chan2 ,y);
           ChanOutInt(chan3 ,z);
           ChanOutInt(chan4 ,w);
         }
     }
 int t ;
void job1(Process *p,Channel *chan1,Channel *chan3,Channel*out1,Channel*out2)
     {
       int i ,r ,s;
       p=p;
        for (i=0 ; i<=1 ; i++ )
         {
           s=ChanInInt(chan1);
           r=ChanInInt(chan3);
             printf("LE RESULTAT TOTAL 1 EST:%d\n",second1(r,s,t));
           ChanOutInt(out1,s);
           ChanOutInt(out2,r);
         }
     }
int k;
void job2(Process *p,Channel *out2,Channel *chan2,Channel *out3)
     {
       int i, m ,n ;
       p=p;
        for(i=0 ; i<=1 ; i++ )
         {
           m=ChanInInt(chan2);
           n=ChanInInt(out2);
             printf("LE RESULTAT TOTAL 2 EST:%d\n",second2(m,n,k));
           ChanOutInt(out3,m);
         }
     }
 int h;
 void job3(Process *p,Channel *chan4,Channel *out1,Channel *out4)
     {
       int i,l,e;
       p=p;
```

```
        for(i=0 ; i<=1 ; i++ )
         {
            l=ChanInInt(out1);
            e=ChanInInt(chan4);
              printf("LE RESULTAT TOTAL 3 EST:%d\n",second3(l,e,h));
            ChanOutInt(out4,e);
         }
     }
 int g;
 void job4(Process *p,Channel *out3,Channel *out4)
     {
       int i ,a,b;
       p=p;
        for(i=0 ; i<=1 ; i++ )
         {
            a=ChanInInt(out3);
            b=ChanInInt(out4);
              printf("LE RESULTAT TOTAL 4 EST:%d\n",second4(a,b,g));
         }
     }
int main()
     {
            Process *p1,*p2,*p3,*p4,*p5;
            Channel *chan1 ,*chan2,*chan3,*chan4,*out1,*out2,*out3,*out4;
            chan1=ChanAlloc();
            if (chan1==NULL)  abort();
            chan2=ChanAlloc();
            if (chan2==NULL)  abort();
            chan3=ChanAlloc();
            if (chan3==NULL)  abort();
            chan4=ChanAlloc();
            if (chan4==NULL)  abort();
            out1=ChanAlloc();
            if(out1==NULL)    abort();
            out2=ChanAlloc();
            if(out2==NULL)    abort();
            out3=ChanAlloc();
            if(out3==NULL)    abort();
            out4=ChanAlloc();
            if (out4==NULL)   abort();
            p1=ProcAlloc(master,0,4,chan1,chan2,chan3,chan4);
            if (p1==NULL)     abort();
            p2=ProcAlloc(job1,0,4,chan1,chan3,out1,out2);
            if (p2==NULL)     abort();
            p3=ProcAlloc(job2,0,3,out2,chan2,out3);
            if (p3==NULL)     abort();
            p4=ProcAlloc(job3,0,3,chan4,out1,out4);
            if (p4==NULL)     abort();
            p5=ProcAlloc(job4,0,2,out3,out4);
            if (p5==NULL)     abort();
            ProcPar(p1,p2,p3,p4,p5,NULL);
     }
              static  second1 (int i,int e,int v)
                       { t=(i*e)+v;
                         return(t); }
                static  second2 (int i,int e ,int v)
                       { k=(i*e)+v;
                         return(k); }
                static second3 (int i,int e, int v)
                       { h=(i*e)+v;
                         return(h); }
              static second4 (int i, int e, int v)
                       { g=(i*e)+v;
                         return(g); }
```

```
/*************************************************************************/
/*      CE PROGRAMME  REALISE LE TRI DE CINQ VALEURS PAR ORDRE   CROISSANT */
/*                      L'OUTIL DE PROGRAMMATION EST :                    */
/*                    ********************************                    */
/*                            LE "C PARALLELE"                            */
/*                          ******************                            */
/*************************************************************************/
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<process.h>
#include<channel.h>
void job1 (Process *p,Channel *chan)
{
   int i ,s ,r ,x1;
   p=p;
   printf("donner la valeur 1:\n");
   scanf("%d",&r);
      for(i=0  ; i<=3;  i++)
      {
        printf("donner la valeur %d:\n",i+2);
        scanf("%d",&s);
          if(r<s)
           { x1=r;
             r=s;
             s=x1; }
       ChanOutInt(chan,s);
      }
       ChanOutInt(chan,r);
}
void job2 (Process *p,Channel *chan, Channel *chon)
{
   int i ,s ,r ,x2 ;
   p=p;
   r=ChanInInt(chan);
      for (i=0 ; i<=3  ; i++)
      {
        s=ChanInInt(chan);
           if (r<s)
            { x2=r;
              r=s;
              s=x2; }
       ChanOutInt(chon,s);
      }
       ChanOutInt(chon,r);
}
void job3(Process *p, Channel *chon ,Channel *chin)
{
   int i ,s ,r ,x3;
   p=p;
   r=ChanInInt(chon);
      for (i=0 ; i<=3 ; i++)
      {
        s=ChanInInt(chon);
           if (r<s)
            { x3=r;
              r=s;
              s=x3; }
       ChanOutInt(chin,s);
      }
       ChanOutInt(chin,r);
}
 void job4 (Process *p ,Channel *chin )
```

```
{
   int i, s ,r ,x4;
   p=p;
   r=ChanInInt(chin);
      for (i=0 ; i<=3 ; i++)
      {
        s=ChanInInt(chin);
           if(r<s)
            { x4=r;
              r=s;
              s=x4; }

      printf("LE RESULTAT %d EST : %d\n",i+1,s);
     }

      printf("LE RESULTAT  5 EST : %d\n",r);
}
   int  main()
          {
              Process *p1 , *p2, *p3 ,*p4  ;
              Channel *chan ,*chon , *chin ;
                chan=ChanAlloc();
                  if (chan==NULL) abort();
                chon=ChanAlloc();
                  if (chon==NULL) abort();
                chin=ChanAlloc();
                  if (chin==NULL) abort();
                p1=ProcAlloc(job1,0,1,chan);
                  if (p1==NULL) abort();
                p2=ProcAlloc(job2,0,2,chan,chon);
                  if (p2==NULL) abort();
                p3=ProcAlloc(job3,0,2,chon,chin);
                  if (p3==NULL) abort();
                p4=ProcAlloc(job4,0,1,chin);
                  if (p4==NULL) abort();
                ProcPar(p1,p2,p3,p4,NULL);
          }
```

CONCLUSION

CONCLUSION:

Le but de notre projet est d'étudier les structures systoliques et de simuler quelques applications.

* Notre étude théorique sur ces structures, nous a permis de bien voir le grand changement que peut apporter l'idée révolutionnaire du parallélisme. Ces architectures parallèles, simples, régulières et efficaces, à coût raisonnable, ont un mode de fonctionnement très élégant, basé sur la synchronisation. Pour cela, ils sont adaptables à différents algorithmes.

* La simulation de quelques applications, nous a mené, à étudier au préalable le transputer. Ce magnifique microprocesseur a la potentialité de communiquer avec quatre autres transputeurs au moyen de liens, et ce dans le but de réaliser des réseaux parallèles de hauts niveaux. Cet élément qui peut supporter la concurrence a été notre outil de simulation à l'aide du langage , le "C Parallèle".

Ce riche et passionnant langage, nous a permis de programmer deux applications, dans le but de remplacer une structure matérielle par une structure logicielle. Ces deux applications sont, une multiplication matricielle réalisée par un réseau systolique carré et le tri des membres réalisé par un réseau pipeline.

Cette simulation nous a permis, de nous initier au traitement parallèle en le touchant de près, de manipuler le "C Parallèle" et de programmer sur transputer.

Il serait important de noter qu'à partir du système de programmation mis à notre disposition durant notre étude (PC) on a pu faire exécuter différents programmes en "C Parallèle, qui simulent des réseaux systoliques. Mais la notion du parallélisme, n'est jamais réellement réalisée à cause de la structure séquentielle de la machine utilisée. Donc on n'a pas réalisé un vrai parallélisme, mais une simulation qui est bien le but de notre projet.

Notons que lorsqu'un programme concurrent s'exécute sur une machine séquentielle, il s'exécutera bien évident après, sur une machine concurrente.

ANNEXES

ANNEXE I

IV-7 DESCRIPTION DU BROCHAGE :

1- **Système service**:

Reset: Ce signal d'entrée initialise le transputer.

Analyse: Ce signal d'entrée, après activation, force le transputer à arrêter toutes ses activités, pour être ensuite analysé, et cela en cas de correction. La ligne s'utilise conjointement avec reset.

Error: C'est un signal de sortie, son activation indique la survenue d'une erreur durant une exécution du programme. Ce signal reste actif jusqu'à la prochaine remise à zéro du circuit.

ClocKin: Horloge externe à partir de laquelle sont dérivées toutes les horloges internes, sa fréquence est de 5MHz pour tous les transputers, quelles que soient leur vitesse et la taille de l'information manipulée.

Vcc: Tension d'alimentation de 5v qui doit être découplée à la masse.

GND: masse, référence du "0" logique.

2- **Interfaces Link**:

LinKin i (i=0...3): bornes d'entrée pour les liens de communications. Elles reçoivent des flux de bits constitués de bits de données et de contrôle.

L'utilisation d'une telle ligne nous oblige automatiquement à la reliée à une ligne LinKout du transputer concerné.

LinKout i (i=0...3): bornes de sortie pour les liens de communication, émettant les flux de bits. De même, une telle ligne doit être reliée à une et une seule ligne LinKin pour assurer la communication entre les deux transputers.

3- **Interruption**:

EventReq: Signal de demande de service par un dispositif externe. Il peut aussi signaler l'occurrence d'un événement extérieur pour la synchronisation transputer/environnement. Il se comporte comme une ligne d'interruption.

Event Ack: Signal de reconnaissance émis par le transputer lorsque le service est exécuté en réponse à une activation de la ligne Event Req. Ensuite EventAck redevient inactif dés que la demande de service est relâchée.

4- **Interface mémoire**:

Clockout: Sortie reproduisant l'horloge interne du processeur et synchronisée sur l'activité de l'interface mémoire. Elle peut être utilisée pour synchroniser des dispositifs externes.

Not Mem S 0-4: Ces signaux de sortie sont utilisés pour le contrôle des boitiers mémoires externes que l'on peut configurer.

Not Mem WB 0-3: Signaux d'écriture associés chacun à un des 4 octets des mots de 32 bits que traite le transputer.

Not Mem Rd: Signal de lecture de la mémoire externe.

Not Mem RF : Signal de contrôle pour le rafraîchissement automatique des mémoires externes. Ce rafraîchissement n'est pas réalisé si le cycle de transfert est indûment prolongé à l'aide de la broche Nem Wait.

Nem Wait: Cette entrée permet de ralentir le cycle de lecture/écriture lorsque le temps d'accès de la mémoire est plus long que le cycle correspondant du transputer.

Mem Config: Cette entrée permet de choisir la manière de configurer l'interface mémoire.

Mem Req: Cette entrée est activée par un dispositif extérieur (Exemple. DMA) demandant le contrôle du bus. En réponse à cette demande, le transputer place les sorties de l'interface mémoire en haute impédance.

Mem Granted: Ce signal, émis par le transputer, indique au circuit ayant généré Mem Req qu'il peut disposer du Bus adresse/Donnée et des lignes de contrôle.

Mem A/D 0-31: Bus externe de 32 bits. Données et adresses sont multiplexées.

ANNEXE II

Concurence: Cas où des opérations sont effectuées simultanement par différents, ou des processus différents.

Pipeline: traitement en chaine des données. Une série de données subit un même tritement, en circulant d'un processeur à un autre.

Simulation: remplacement d'un système par un modèle plus simple ayant un comportement semblable.

Processus indépendants: des processus qui excluent toute forme de partage. Un objet appartient à un seul processus et ne peut être commun à plusieurs.

BIBLIOGRAPHIE

BIBLIOGRAPHIE:

[1] Réalisation d'un logiciel pour le calcul parallèle de la convolution à base de transputer.
réalisé par: Mr FERADJI
Mr LATROUS USTHB 1993

[2] Réseaux systoliques: programmation en langage Occam réalisé par :
Melle LAROUI (ENP) 1993

[3] Alirson Carling
Parllel processing : Occam and the transputer.
Eddition Sigma 1988

[4] Patrice QUINTON et Yves POBERT
Algorithmes et architectures systoliques.
Eddition Masson 1989

[5] José FORTE et Benjamin WAH
Systoliques array concept to implementation.
IEEE 1987

[6] HT. Kung et Caroegie. Mellon University
Why systolic architectures?
IEEE 1982 (Janvier).

[7] SY. Kung, Jean et J.H. Wang
Wave front array.
Processors - concept to implementation
IEEE - 1987

[8] Roger Dettmer
Occam and the transputer.
IEEE - 1985

[9] S. Chen, F. Fuller, L. Loomis, M. Magleby, S. Whitrey
Introduction to computer architectures

[10] Carver. Mead et Lynn Conway
Introduction aux systèmes VLSI

[11] Matrix compultions on systolic type array.

[12] INMOS limited
ANSI C. Toolset
User manual August 1990