Étude et réalisation d'un système microprocesseur pour le traitement des algorithmes parallèles

Sarwat Ragab

THESE

présentée à

L'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

pour obtenir le grade de
DOCTEUR INGENIEUR
"Génie Informatique"

par

Sarwat RAGAB

ÉTUDE ET RÉALISATION D'UN SYSTÈME MULTIMICROPROCESSEUR POUR LE TRAITEMENT DES ALGORITHMES PARALLÈLES

Thèse soutenue le 7 Juin 1983 devant la commission d'examen :

Président :	L. BOLLIET
Examinateurs :	C. BELLISSANT
	Y. DESWARTE
	G. MAZARE
	A. RECOQUE
Invité :	F.R. DAGALLIER

Je remercie tout particulièrement :

Monsieur L. BOLLIET, Professeur à l'Université de Grenoble 2, pour m'avoir accueilli dans son équipe de recherche ; je luis suis très reconnaissant d'avoir accepté de présider le jury,

Monsieur C. BELLISSANT, Professeur à l'Université de Grenoble 2, qui a bien voulu accepter de participer au jury,

Monsieur F.R. DAGALLIER, Directeur de la Société THOMSON TITN Rhône Alpes dont je fais parti, qui a bien voulu accepter de juger mon travail,

Monsieur Y. DESWARTE, Membre de l'Equipe de Direction de l'Action Permanente SURF, qui a bien voulu examiner en détail mon travail et m'a permis ainsi d'améliorer sensiblement mon manuscrit,

Monsieur G. MAZARE, Professeur à l'Ecole Nationale Supérieure d'Informatique et de Mathématiques Appliquées de Grenoble, qui a dirigé mes travaux et m'a permis grâce à ses conseils de spécialiste dans ce domaine, de mener à bien mon travail,

Madame A. RECOQUE, Conseiller Scientifique au Centre de Recherche CII-HB, qui a accepté de participer au jury,

Messieurs R. BOUTTAZ, P. BOREL et Ph. COLOVATTI, de l'Atelier de Micro Informatique de Grenoble, pour la part importante qu'ils ont prise dans la réalisation matérielle de la machine,

Mesdames C. CHALAND et M. BETTON pour leur amabilité et la réalisation matérielle de ce document, ainsi que le service tirage de l'IMAG.

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Président : Daniel BLOCH

Vice-Présidents : René CARRE
Hervé CHERADAME
Marcel IVANES

Année universitaire 1982-1983

Professeurs des Universités

ANCEAU	François	E.N.S.I.M.A.G.	LACOUME	Jean Louis	E.N.S.I.E.G.
BARRAUD	Alain	E.N.S.I.E.G.	LATOMBE	Jean Claude	E.N.S.I.M.A.G.
BAUDELET	Bernard	E.N.S.I.E.G.	LESIEUR	Marcel	E.N.S.H.G.
BESSON	Jean	E.N.S.E.E.G.	LESPINARD	Georges	E.N.S.H.G.
BLIMAN	Samuel	E.N.S.E.R.G.	LONGEQUEUE	Jean Pierre	E.N.S.I.E.G.
BLOCH	Daniel	E.N.S.I.E.G.	MAZARE	Guy	E.N.S.I.M.A.G.
BOIS	Philippe	E.N.S.H.G.	MOREAU	René	E.N.S.H.G.
BONNETAIN	Lucien	E.N.S.E.E.G.	MORET	Roger	E.N.S.I.E.G.
BONNIER	Etienne	E.N.S.E.E.G.	MOSSIERE	Jacques	E.N.S.I.M.A.G.
BOUVARD	Maurice	E.N.S.H.G.	PARIAUD	Jean Charles	E.N.S.E.E.G.
BRISSONNEAU	Pierre	E.N.S.I.E.G.	PAUTHENET	René	E.N.S.I.E.G.
BUYLE BODIN	Maurice	E.N.S.E.R.G.	PERRET	René	E.N.S.I.E.G.
CAVAIGNAC	Jean François	E.N.S.I.E.G.	PERRET	Robert	E.N.S.I.E.G.
CHARTIER	Germain	E.N.S.I.E.G.	PIAU	Jean Michel	E.N.S.H.G.
CHENEVIER	Pierre	E.N.S.E.R.G.	POLOUJADOFF	Michel	E.N.S.I.E.G.
CHERADAME	Hervé	U.E.R.M.C.P.P.	POUPOT	Christian	E.N.S.E.R.G.
CHERUY	Arlette	E.N.S.I.E.G.	RAMEAU	Jean Jacques	E.N.S.E.E.G.
CHIAVERINA	Jean	U.E.R.M.C.P.P.	RENAUD	Maurice	U.E.R.M.C.P.P.
COHEN	Joseph	E.N.S.E.R.G.	ROBERT	André	U.E.R.M.C.P.P.
COUMES	André	E.N.S.E.R.G.	ROBERT	François	E.N.S.I.M.A.G.
DURAND	Francis	E.N.S.E.E.G.	SABONNADIERE	Jean Claude	E.N.S.I.E.G.
DURAND	Jean Louis	E.N.S.I.E.G.	SAUCIER	Gabrielle	E.N.S.I.M.A.G.
FELICI	Noël	E.N.S.I.E.G.	SCHLENKER	Claire	E.N.S.I.E.G.
FOULARD	Claude	E.N.S.I.E.G.	SCHLENKER	Michel	E.N.S.I.E.G.
GENTIL	Pierre	E.N.S.E.R.G.	SERMET	Pierre	E.N.S.E.R.G.
GUERIN	Bernard	E.N.S.E.R.G.	SILVY	Jacques	U.E.R.M.C.P.P.
GUYOT	Pierre	E.N.S.E.E.G.	SOHM	Jean Claude	E.N.S.E.E.G.
IVANES	Marcel	E.N.S.I.E.G.	SOUQUET	Jean Louis	E.N.S.E.E.G.
JAUSSAUD	Pierre	E.N.S.I.E.G.	VEILLON	Gérard	E.N.S.I.M.A.G.
JOUBERT	Jean Claude	E.N.S.I.E.G.	ZADWORNY	François	E.N.S.E.R.G.
JOURDAIN	Geneviève	E.N.S.I.E.G.			

Professeurs associés

BASTIN	Georges	E.N.S.H.G.	GANDINI	Alessandro	U.E.R.M.C.P.P.
BERRIL	John	E.N.S.H.G.	HAYASHI	Hirashi	E.N.S.I.E.G.
CARREAU	Pierre	E.N.S.H.G.			

Professeurs Université des Sciences Sociales (Grenoble II)

BOLLIET	Louis	CHATELIN	Françoise

Professeurs E.N.S. Mines de Saint Etienne

RIEU	Jean	SOUSTELLE	Michel

Chercheurs du C.N.R.S.

FRUCHART	Robert	Directeur de recherche	HOPFINGER	Emil	Maître de recher
VACHAUD	Georges	Directeur de Recherche	JOUD	Jean Charles	Maître de recher
ALLIBERT	Michel	Maître de recherche	KAMARINOS	Georges	Maître de recher
ANSARA	Ibrahim	Maître de Recherche	KLEITZ	Michel	Maître de recher
ARMAND	Michel	Maître de recherche	LANDAU	Ioan-Dore	Maître de recher
BINDER	Gilbert		LASJAUNIAS	J.C.	
CARRE	René	Maître de recherche	MERMET	Jean	Maître de recher
DAVID	René	Maître de recherche	MUNIER	Jacques	Maître de recher
DEPORTES	Jacques		PIAU	Monique	
DRIOLE	Jean	Maître de recherche	PORTESEIL	Jean Louis	
GIGNOUX	Damien		THOLENCE	Jean Louis	
GIVORD	Dominique		VERDILLON	André	
GUELIN	Pierre				

Chercheurs du Ministère de la Recherche et de la Technologie
(Directeurs et Maîtres de recherche - E.N.S. Mines de Saint Etienne)

LESBATS	Pierre	Directeur de recherche	LALAUZE	René	Maître de recher
BISCONDI	Michel	Maître de recherche	LANCELOT	Francis	Maître de recher
KOBYLANSKI	André	Maître de recherche	THEVENOT	François	Maître de recher
LE COZE	Jean	Maître de recherche	TRAN MINH	Canh	Maître de recher

...

Personnalités habilitées à diriger des travaux de recherche
(Décision du Conseil Scientifique)

E.N.S.E.E.G.

ALLIBERT	Colette	DIARD	Jean Paul	NGUYEN TRUONG	Bernadette
BERNARD	Claude	EUSTATOPOULOS	Nicolas	RAVAINE	Denis
BONNET	Roland	FOSTER	Panayotis	SAINFORT	(CENG)
CAILLET	Marcel	GALERIE	Alain	SARRAZIN	Pierre
CHATILLON	Catherine	HAMMOU	Abdelkader	SIMON	Jean Paul
CHATILLON	Christian	MALMEJAC	Yves (CENG)	TOUZAIN	Philippe
COULON	Michel	MARTIN GARIN	Régina	URBAIN	Georges (Laboratoire des ultra-réfractaire ODEILLO)

E.N.S.Mines Saint Etienne

GUILHOT	Bernard	THOMAS	Gérard	DRIVER	Julien

E.N.S.E.R.G.

BARIBAUD	Michel	CHEHIKIAN	Alain	HERAULT	Jeanny
BOREL	Joseph	DOLMAZON	Jean Marc	MONLLOR	Christian
CHOVET	Alain				

E.N.S.I.E.G.

BORNARD	Guy	KOFMAN	Walter	MAZUER	Jean
DESCHIZEAUX	Pierre	LEJEUNE	Gérard	PERARD	Jacques
GLANGEAUD	François			REINISCH	Raymond

E.N.S.H.G.

ALEMANY	Antoine	MICHEL	Jean Marie	ROWE	Alain
BOIS	Daniel	OBLED	Charles	VAUCLIN	Michel
DARVE	Félix			WACK	Bernard

E.N.S.I.M.A.G.

BERT	Didier	COURTOIS	Bernard	FONLUPT	Jean
CALMET	Jacques	DELLA DORA	Jean	SIFAKIS	Joseph
COURTIN	Jacques				

U.E.R.M.C.P.P.

CHARUEL	Robert

C.E.N.G.

CADET	Jean	JOUVE	Hubert (LETI)	PERROUD	Paul
COEURE	Philippe (LETI)	NICOLAU	Yvan (LETI)	PEUZIN	Jean Claude (LETI)
DELHAYE	Jean Marc (STT)	NIFENECKER	Hervé	TAIEB	Maurice
DUPUY	Michel (LETI)			VINCENDON	Marc

Laboratoires extérieurs :

C.N.E.T.

DEMOULIN	Eric	GERBER	Roland	MERCKEL	Gérard
DEVINE	R.A.B.			PAULEAU	Yves

I.N.S.A. Lyon

GAUBERT	C.

****

إهداء

إلى والدي ووالدتي

A Mes Parents ...

SOMMAIRE

SOMMAIRE

AVANT - PROPOS

AVANT PROPOS

Le sujet traité dans cet ouvrage a été réalisé au sein de l'équipe Systèmes et Applications dans le cadre des travaux de recherches sur les MULTIMICROPROCESSEURS, à l'I.M.A.G.

L'objectif de notre travail était d'évaluer les performances des systèmes à multimicroprocesseurs non spécialisés, dans le traitement des algorithmes parallèles.

Pour atteindre notre objectif, nous sommes passés par les étapes suivantes:

1. L'étude détaillée des différentes architectures multiprocesseurs,

2. La définition et la réalisation d'un système à structure multimicroprocesseurs.

   A cette étape, le travail consistait en:

   a. La réalisation matérielle (hardware) de la maquette à base d'un microprocesseur existant,

   b. La réalisation logicielle (software) du système d'exploitation parallèle (S.E.P) et l'ensemble du logiciel de mise au point de cette architecture.

3. L'étude de certains algorithmes parallèles, l'exécution de ces programmes sur la machine et l'évaluation des performances du système réalisé dans le traitement de ces programmes.

ooOOOOOoo

CHAPITRE 1

ETUDE DES DIFFERENTES ARCHITECTURES MULTIPROCESSEURS

Tout au long de l'histoire de l'informatique, on n'a cessé de penser à cette idée: connecter plusieurs processeurs pour réaliser une machine plus puissante. Cette idée allait de pair avec le développement d'un logiciel parallèle d'exploitation de ces machines. C'est ainsi que furent développés les différents algorithmes parallèles et les divers langages du parallélisme. Cette étude nous permettra de situer précisément le type de système à multimicroprocesseurs réalisé.

1.1. CLASSIFICATION DES SYSTEMES A MULTIPROCESSEURS

Le classement des machines multiprocesseurs par séquencement des programmes nous permet de distinguer quatre types essentiels d'architectures (FLYNN 1966-1972) selon les flots d'instructions et les flots de données traitées. (FLY-72.)

1.1.1. *S.I.S.D.* (Single Instruction stream Single Data stream)

Les machines monoprocesseurs exécutant un seul flot d'entrée sur un seul ensemble de données selon le séquencement des instructions, représentent cette classe qui est la plus classique des machines.

1.1.2. *M.I.S.D.* (Multiple Instruction stream Single Data stream)

Ce sont des machines qui exécutent plusieurs instructions simultanément sur un seul flot de données. Les machines "Pipe-line" qui utilisent des techniques d'anticipation de traitement sont les meilleurs exemples de ce type d'architecture.

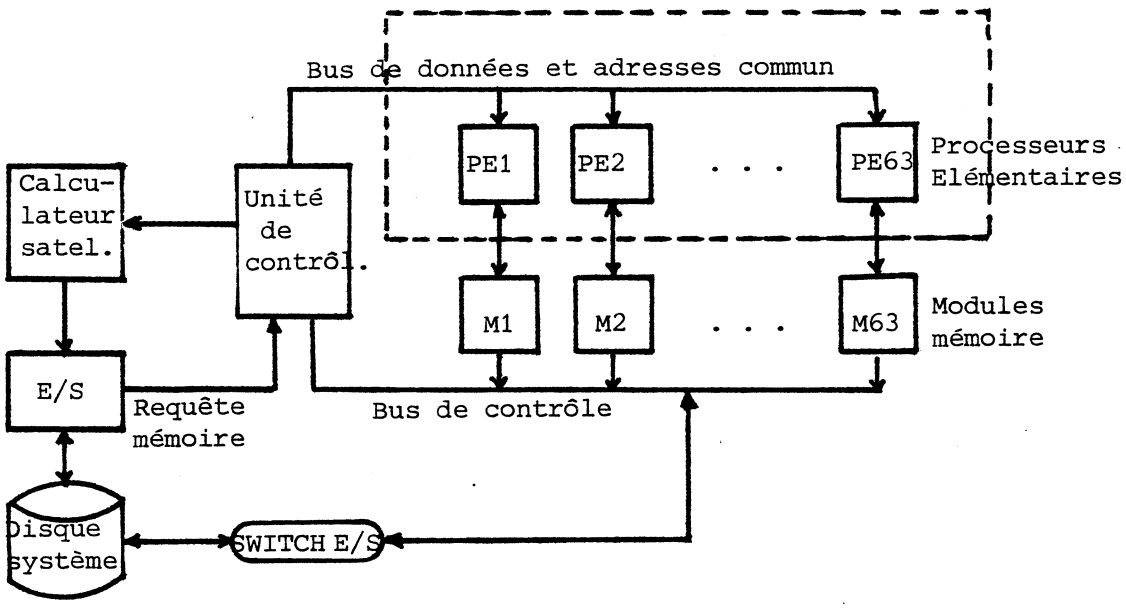
1.1.3. *S.I.M.D.* (Single Instruction stream Multiple Data stream)

L'architecture dans laquelle l'unité de contrôle fournit un seul flot d'instructions à exécuter par plusieurs processeurs, chacun avec un ensemble de données dans sa mémoire privée; ILLIAC IV est un exemple de ces machines.

Selon la complexité de son unité de contrôle, la puissance de traitement et le système d'adressage de ses processeurs élémentaires et de leurs interconnexions, on peut les regrouper en: Architecture matricielle, Architecture associative ou Architecture à contrôle hiérarchique.

Dans les multiprocesseurs à structure matricielle, les processeurs élémentaires communiquent en général chacun avec ses voisins, sinon c'est à travers l'unité de contrôle selon un protocole synchrone bien défini.

Dans les structures associatives, les communications se font à travers une mémoire de taille importante.



Bloc diagramme ILLIAC IV

Schéma 1.1.

1.1.4. *M.I.M.D.* (Multiple Instruction stream Multiple Data stream)

Ce sont les véritables machines multiprocesseurs. En principe, elles permettent l'exécution de plusieurs flots d'instructions sur plusieurs flots de données d'une manière asynchrone. Les différentes architectures de cette catégorie peuvent être classées selon deux critères de conception essentiels: (BAE 76)

- La nature des connexions entre les processeurs élémentaires et les modules mémoires,
- L'homogénéité des unités de traitement.

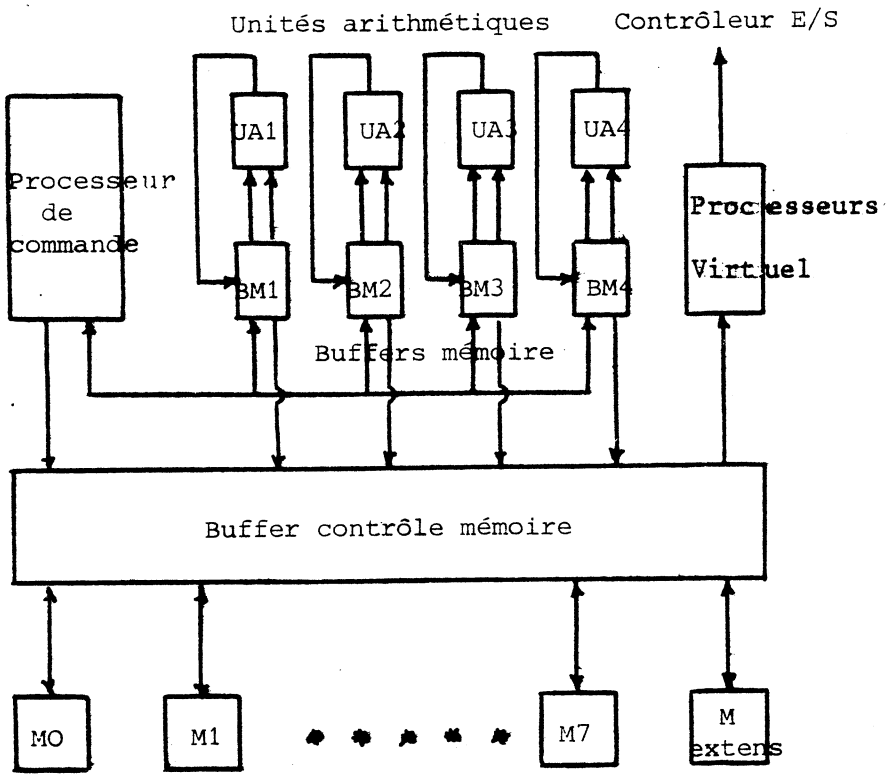
Le premier critère permet usuellement de distinguer entre deux types de structure M.I.M.D :

a. Les multiprocesseurs fortement couplés (Tightly Connected M/C),

b. Les multiprocesseurs faiblement couplés (Loosely Connected M/C).

Mais nous préférons réserver cette dernière classification à ce que résulte de l'utilisation du second critère, l'homogénéité des unités de traitement.

Dans le cas des multiprocesseurs fortement couplés le nombre des unités de traitement est fixe et elles opèrent sous la supervision d'une unité de contrôle dont le rôle est spécifique à chaque application. Ce contrôleur est en général un circuit matériel. Les processeurs élémentaires sont, en général, hétérogènes dans un sens qu'ils constituent des blocs fonctionnels différents. Ces unités fonctionnelles sont commandées par une autre unité qui décode les instructions, recherche les opérandes et leur distribue les ordres. Le schéma 1.2. représente un système multiprocesseur de ce type (TI ASC).



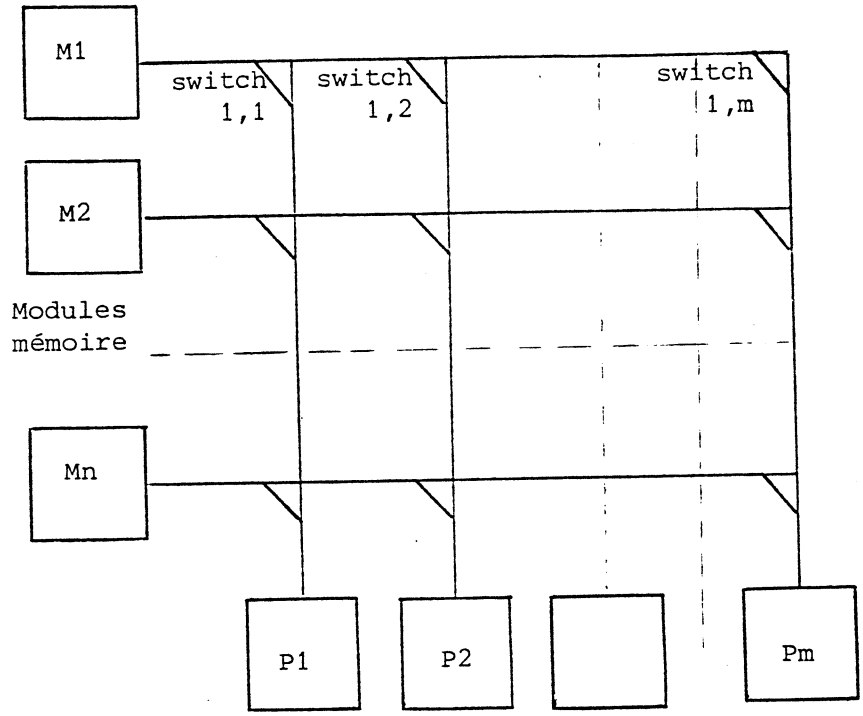
Structure calculateur TI-ASC

Schéma 1.2.

Ce type de connexion existe aussi dans les architectures matricielles (S.I.M.D) mais dans ce cas les unités de traitement sont identiques opérant chacune sur un ensemble de données privées.

Dans le cas des multiprocesseurs M.I.M.D faiblement connectés on constate une architecture plutôt modulaire. Les processeurs élémentaires sont en général très homogènes. Les connexions entre les processeurs et les modules mémoire sont réalisées selon les trois techniques suivantes:

A. *Commutateur "Cross Bar":* Ce type de connexion permet une liaison directe entre les processeurs élémentaires et les modules mémoire. Le schéma 1.3. représente ce type de connexion.

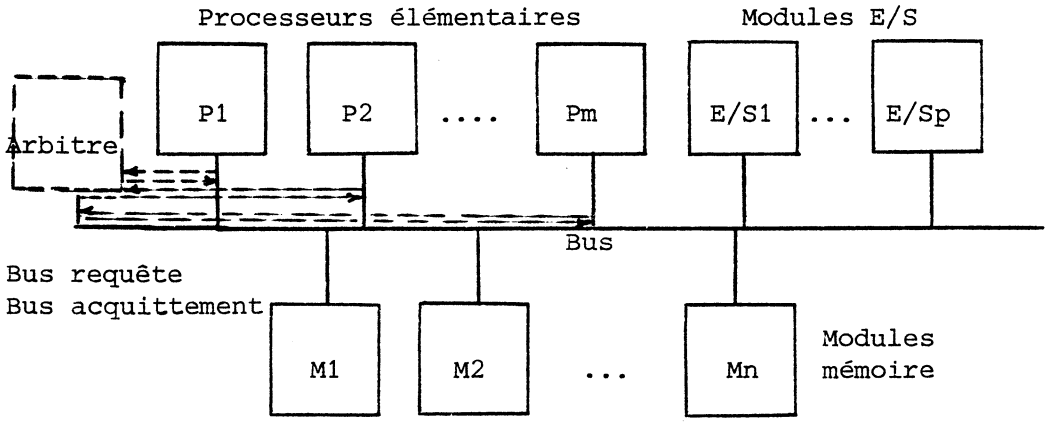


Connexion avec SWITCH CROSS BAR

Schéma 1.3.

Pour réaliser une connexion de ce type entre "m" processeurs et "n" modules mémoire, il nous faudra "m.n" commutateur point. Chaque commutateur point doit être capable de résoudre les conflits d'accès en mémoire sur ce point. Si le nombre des modules mémoire est égal à celui des processeurs de base qui les utilisent, le nombre de commutateurs point nécessaires s'élèvera à nxn. On s'aperçoit donc que la réalisation d'un commutateur Cross Bar est trop encombrante et coûteuse et par conséquent, on limitera le nombre des modules mémoire et des processeurs. Pourtant, on retrouve de telles connexions dans le Cmmp (Architecture multiprocesseur ayant 16 processeurs élémentaires -PDP11- et 16 modules mémoire, chaque processeur ayant une mémoire locale; les communications interprocesseurs s'effectuent à travers un module de connexion spécial et en utilisant un bus partagé).

B. *Bus partagé:* Les connexions entre processeurs et modules mémoire sont réalisés par un ou plusieurs bus partagés. Pour réaliser des transactions parallèles sur ce bus, encore faut-il ajouter des circuits d'arbitrage de bus. L'allocation du bus peut être effectuée selon un mode synchrone, avec ou sans priorités. Le bus peut être aussi accordé aux processeurs qui le demandent d'une manière asynchrone. Le schéma 1.4. représente ce type de connexion.



Connexion avec bus commun partagé

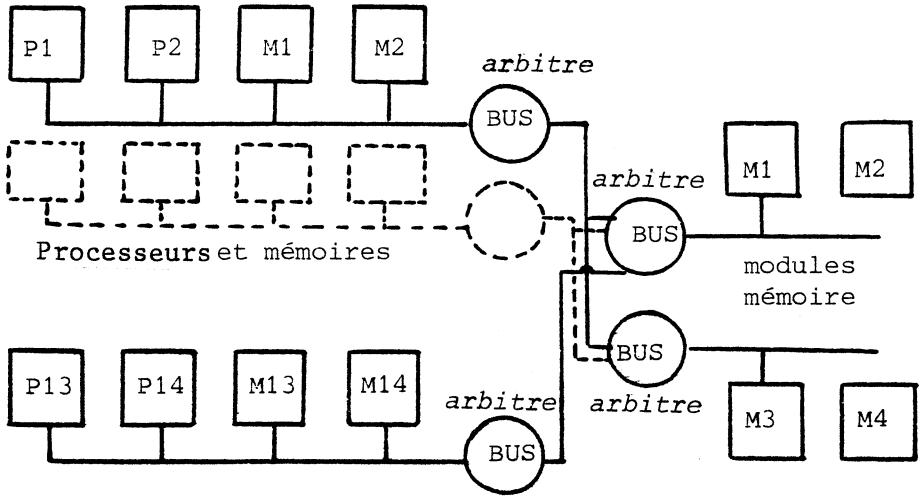
Schéma 1.4.

Pour réaliser plus de parallélisme, on pourra augmenter le nombre de ces bus; il est possible de les spécialiser à fonctionner en mode unidirectionnel, mais on augmentera ainsi la complexité et par conséquent le coût de la réalisation.

C. *Système de mémoires multiports.* Dans ce cas la commutation est concentrée au niveau des modules mémoire. Tous les processeurs auront accès à tous les modules mémoire via leurs propres bus. Les conflits d'accès seront résolus par des priorités fixes réalisées matériellement.

Il est possible d'utiliser une combinaison de ces techniques comme dans le schéma 1.5.

Avant de terminer le sujet de classification avec les quatre classes de multiprocesseurs définies par FLYNN ci-dessus, il est important de décrire une cinquième classe de multiprocesseurs qui est apparue avec les machines qu'on désigne par les supermultiprocesseurs, ce sont les multiprocesseurs à architecture reconfigurable.



Architecture multiprocesseur (7 bi-processeurs) ayant deux bancs de modules mémoire partagés

Schéma 1.5.

1.1.5. Multiprocesseurs à architecture reconfigurable

Ce type d'architecture permet de redéfinir la taille du système et le type de connexion entre les processeurs élémentaires et les modules mémoire ainsi que les unités d'entrées et de sorties.

Parmi les raisons qui ont poussé les concepteurs à la réalisation de tels monstres: la volonté de réaliser une grande fiabilité de leurs systèmes et d'offrir aux différents utilisateurs la possibilité de travailler sur plusieurs types d'architecture multiprocesseur sans changer de machine; ainsi, ils pourront satisfaire leurs ambitions d'ouverture vers de nouvelles architectures.

Par reconfiguration on désigne la possibilité de modifier dynamiquement les interconnexions entre les différents modules constituant un système multiprocesseur. En termes de systèmes informatiques, cette reconfiguration consiste à reconnecter les processeurs aux ressources pour obtenir un système global répondant aux besoins d'une application donnée.

L'avantage de cette architecture par rapport aux autres classes citées précédemment, repose sur le fait que le degré de parallélisme varie d'un programme à un autre et qu'une structure rigide ne permet pas d'exécuter avec la même efficacité deux types de programmes différents sur la même machine. On peut énumérer ces avantages dans les points suivants:

- L'utilisateur n'aura plus besoin de restructurer ses programmes selon l'organisation de la machine.

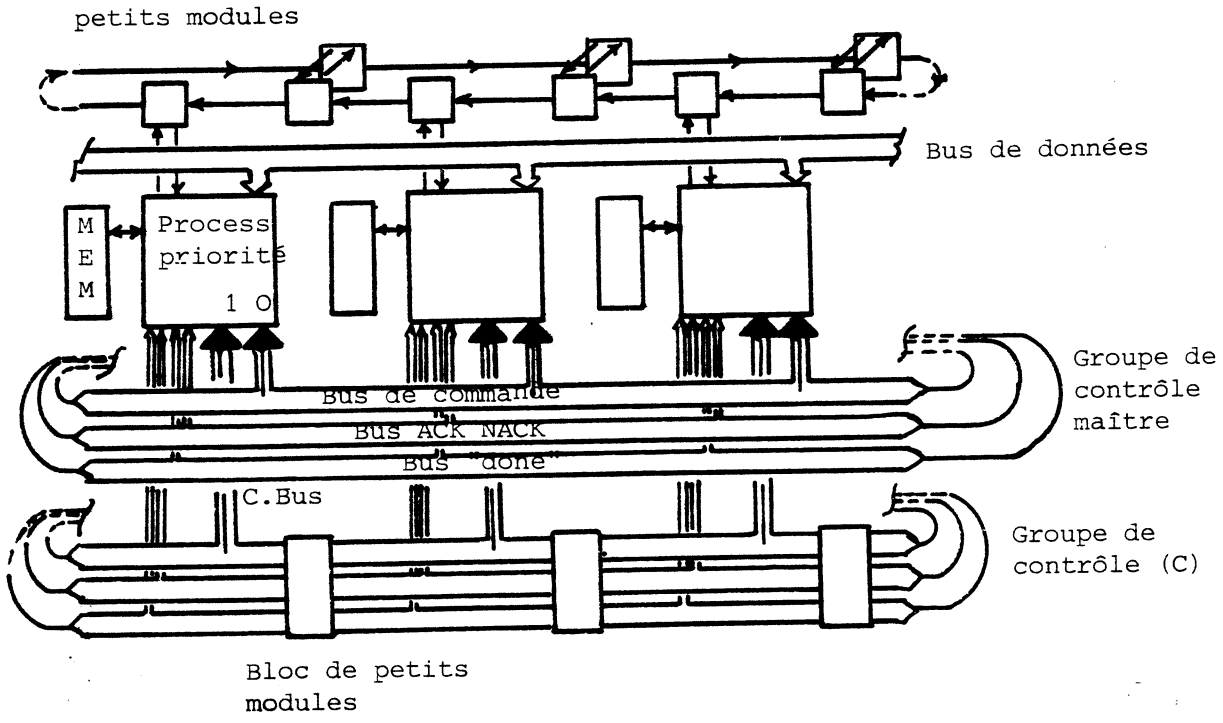
- La modularité et la souplesse des interconnexions des ressources rendra possible l'augmentation ou la réduction de la puissance selon les besoins des utilisateurs. Ceci a pour effet aussi de réaliser une augmentation considérable de la fiabilité du système simplement par une exclusion des modules défaillants de la configuration.

Les multiprocesseurs à architecture reconfigurable peuvent être classés en groupes selon les reconfigurations qu'ils peuvent réaliser. On citera quelques types de machines reconfigurables avec des exemples à l'appui sur des projets réalisés ou en cours de réalisation.

I. Architecture multiprocesseur à contrôle hiérarchique (M/C réalisée par ARNOLD et PAGE): (ARN 76)

Dans cette architecture sont connectés des processeurs en tranches par un système de bus. Avec ce type de M/C on peut configurer plusieurs sortes d'architectures multiprocesseurs: matricielle, associative, etc...

Un processeur (qui sera désigné par l'utilisateur) joue le rôle du chef d'orchestre dans le système. Les communications interprocesseurs sont réalisées par deux types de bus: un bus partagé et un bus circulaire. Le schéma 1.6. représente la structure de cette machine.

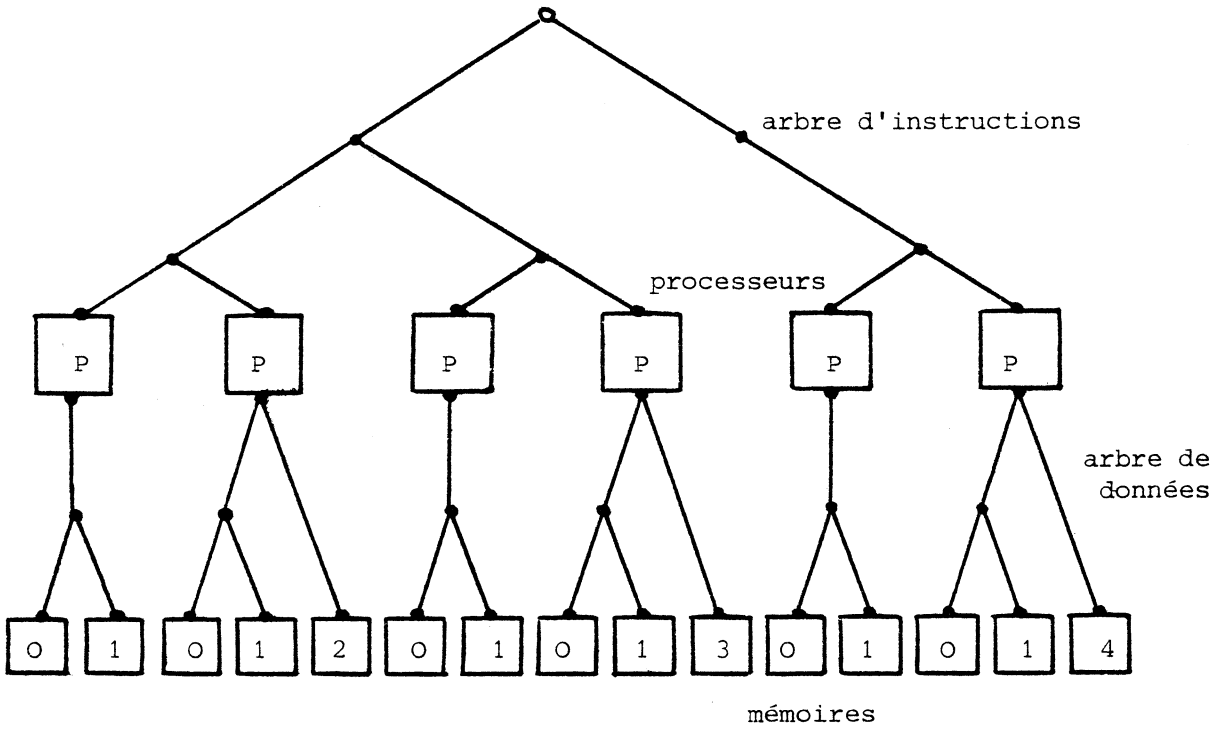


Structure de la m/c multiprocesseur à contrôle hiérarchique

Schéma 1.6.

II. Multiprocesseur Matriciel à structure variable:

Architecture proposée par LIPOVSKI & TRIPATHI. Ce multiprocesseur multimatriciel partitionable ayant une structure arborescente est représenté par le schéma 1.7. (LIP-77)



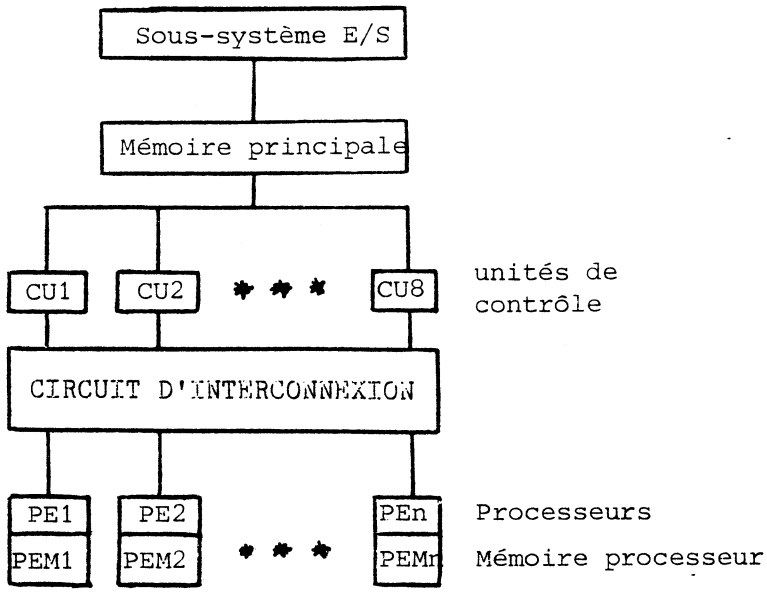
Processeur à structure vectorielle reconfigurable

Schéma 1.7.

III. Architecture Multiassociative (Multiassociative Processor MAP d'ARNOLD & NUTT): (NUT-75)

Dans cette architecture, les unités de contrôle s'affectent un nombre variable de processeurs élémentaires selon la taille de la tâche à réaliser.

Chaque unité de contrôle a un accès direct à l'ensemble des modules mémoire via un bus partagé. Les processeurs élémentaires ayant chacun sa mémoire privée fonctionnent en mode S.I.M.D. Le système est classé sous les M.S.I.M.D. reconfigurables. Le schéma 1.8. représente ce type d'architecture.



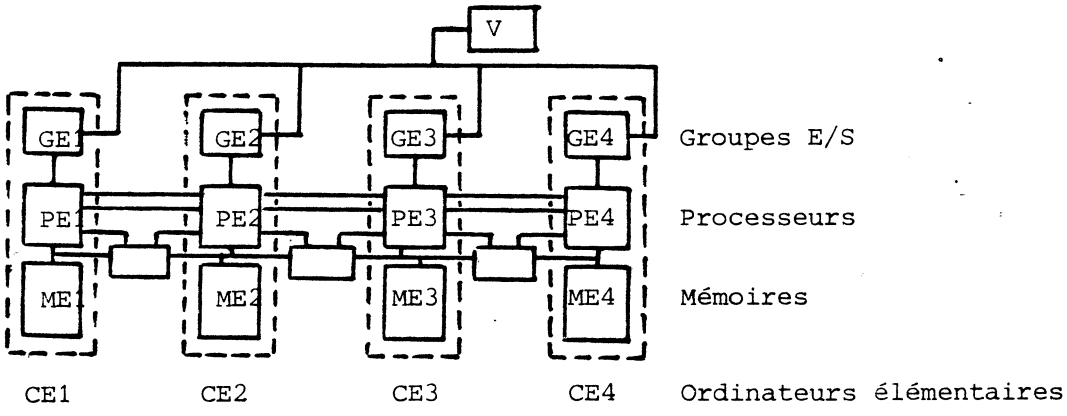
Structure du M.A.P.

Schéma 1.8.

IV. Architecture Multiordinateur dynamique (Dynamic Computer des KARTASCHEV): (KRT-78)

Architecture qui regroupe tout simplement des ordinateurs (unités de traitement, mémoires, unités d'entrée et de sortie).

Chacun de ces ordinateurs est équipé d'un circuit dont le rôle est d'activer ou de désactiver le module d'interconnexions. Ce multiprocesseur est représenté par le schéma 1.9.

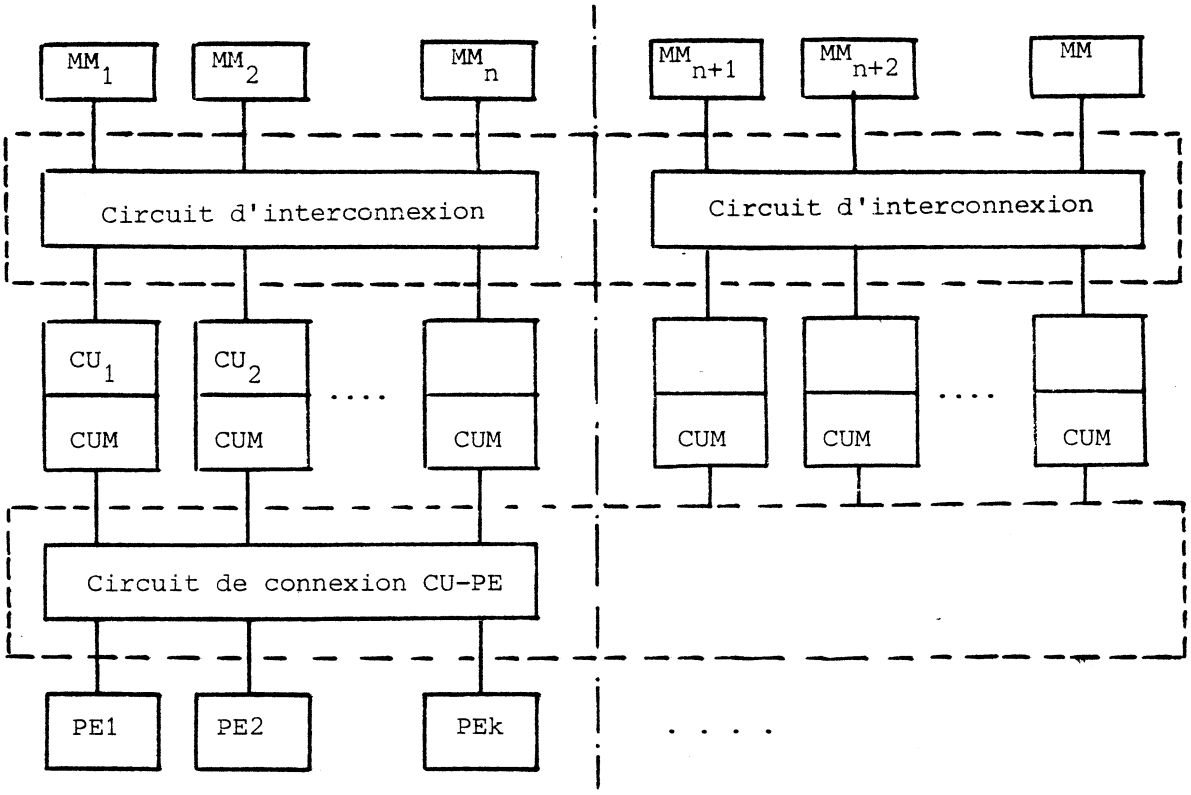


Structure du D.C.

Schéma 1.9.

V. Architecture à Pool de Ressources Reconfigurables
(PRRM de MEHRA & MAJITHA): (MEH-79)

C'est une architecture proche de celle du PASM. Elle suppose un pool de ressources qui peuvent être partagées par des programmes qui demandent des environnements de traitement différents. Le schéma 1.10 représente le bloc diagramme de cette M/C.



Structure du P.R.R.M.P.

Schéma 1.10.

Cette machine se compose en effet de deux pools: un pool S.I.M.D. et un autre M.I.M.D. Les unités de contrôle sont des minicalculateurs ayant une mémoire locale. Elles fournissent aux processeurs de base les contextes programmes. D'autre part, les processeurs élémentaires peuvent s'activer et se désactiver selon une technique associative. En mode M.S.I.M.D les unités de contrôle peuvent s'allouer un nombre variable de processeurs élémentaires à travers un circuit d'interconnexions.

Ayant toutes les ressources sous forme modulaire cette machine est une M.I.M.D. et M.S.I.M.D partitionnable.

VI. Architecture MSIMD/MIMD .Partitionnelle (PASM)

Ce type de multiprocesseurs est capable de se reconfigurer en plusieurs S.I.M.D ou en un M.I.M.D. Un exemple de ce type de machine PASM sera décrit dans les pages suivantes. (SIG-81)

CONCLUSIONS

La classification donnée par FLYNN étant un peu ambiguë en ce qui concerne les M.I.S.D. d'autres classifications sont apparues parmi lesquelles nous citons l'ECS (Erlangen Classification System). (HAN 81) D'un moindre intérêt que celle de FLYNN, elle sera présentée en Annexe.

Après cette classification détaillée des architectures multiprocesseurs, on peut situer notre système comme un M.I.M.D spécialisé dans le traitement des algorithmes parallèles qui peut être reconfigurable statiquement pour se comporter comme un S.I.M.D.

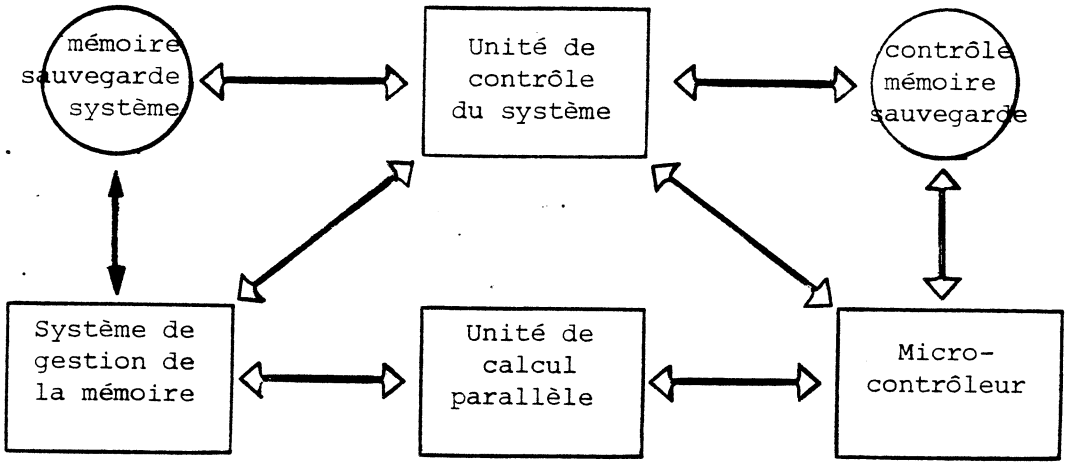
1.2. REVUE DES PROJETS MULTIPROCESSEURS

Ce travail étant considéré comme une continuité des travaux de l'ancienne équipe MULTIMICROPROCESSEURS dirigé par MM. BOLLIET L. et MAZARE G. dans les domaines du parallélisme et des architectures multiprocesseurs, nous nous sommes contentés de présenter quelques projets récents apparus après la dernière revue publiée par l'équipe. Ces projets étant de tailles différentes, on les présentera dans un ordre d'importance décroissant.

1.2.1. *PROJET PASM* (Système Partitionnable M.S.I.M.D/M.I.M.D)

Ce projet a été conçu à l'Université de Michigan (U.S.A.): (SIG 81)

Ce système multiprocesseur est constitué d'un très grand nombre de processeurs reconfigurables. Il a été réalisé spécialement pour l'exploitation du parallélisme dans le traitement de l'image et de la reconnaissance des formes. Il peut être aussi utilisé dans le traitement de la parole et du signal biomédical. Le schéma 1.11. montre le bloc diagramme du système.



Blocs fonctionnels de PASM

Schéma 1.11.

Structure du système

1. *PCU* (Unité de Calcul Parallèle): Elle est composée de $N = 2^n$ processeurs, de N modules mémoire et des circuits d'interconnexion.

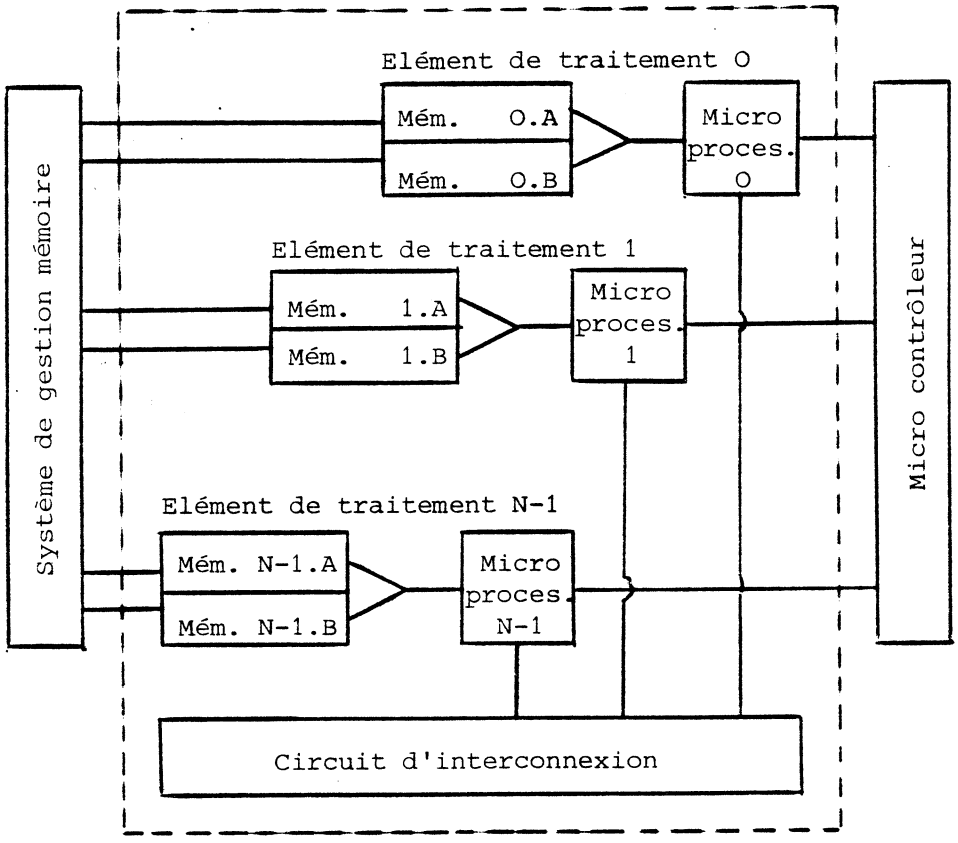
   a. Les modules mémoire représentent la mémoire de données dans les S.I.M.D et les données programme dans les structures M.I.M.D.

   b. Les circuits d'interconnexion permettent les communications entre les PCU et les modules mémoire. Les interconnexions entre les processeurs d'instructions et les modules mémoire peuvent prendre les deux formes suivantes:

      - Connexions PE-PE où chaque processeur dispose d'un accès direct à deux modules mémoire (pour permettre de charger un module pendant que l'autre est utilisé par le processeur). Mais pour accéder aux modules mémoire des autres processeurs il lui faudra passer à travers le circuit d'interconnexion.

      - Connexions Processeurs-Mémoire où tous les processeurs peuvent accéder à tous les modules mémoire via ce circuit d'interconnexion. Ceci est intéressant dans le cas où s'effectuent des échanges importants d'informations entre les différents processeurs de base.

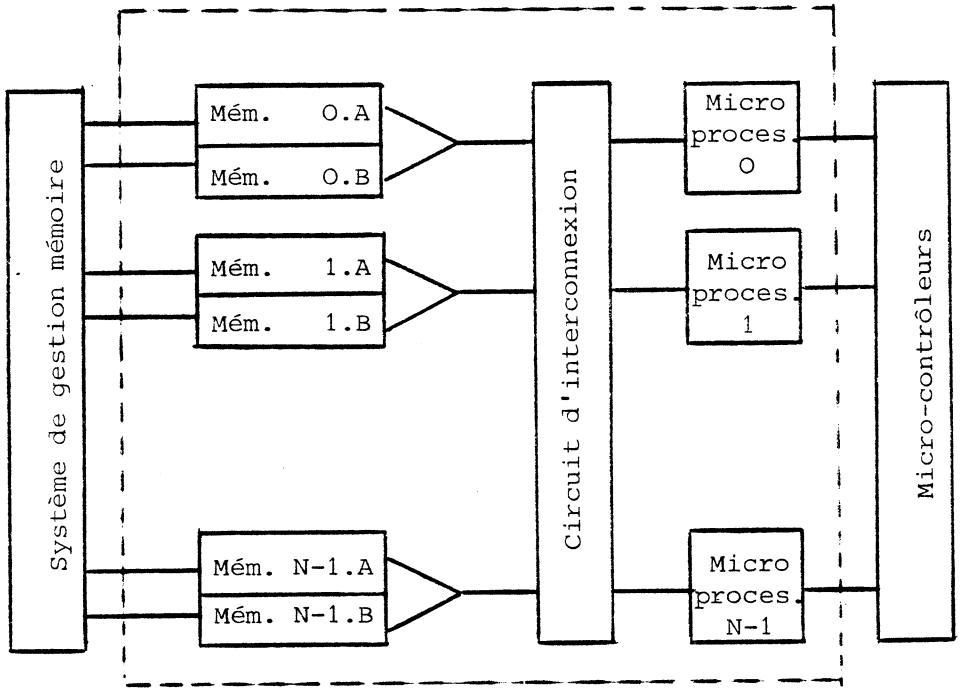
Les deux schémas 1.12. et 1.13. représentent ces deux types de connexions possibles.



Configuration PE-PE de l'unité de calcul parallèle

Schéma 1.12.

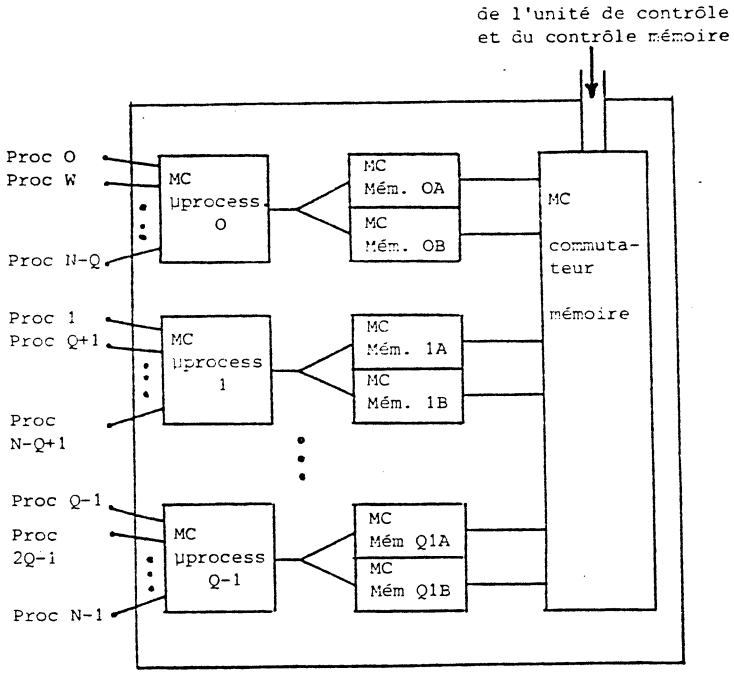
14



Configuration Processeur-Mémoire de l'unité de calcul parallèle

Schéma 1.13.

2. *Les Microcontrôleurs:* Constitués d'un ensemble de microprocesseurs qui ont les rôles des unités de contrôle dans les configurations S.I.M.D. Par contre, dans les configurations M.I.M.D. ils se chargent du contrôle des activités des processeurs élémentaires.



Micro-contrôleurs de PASM

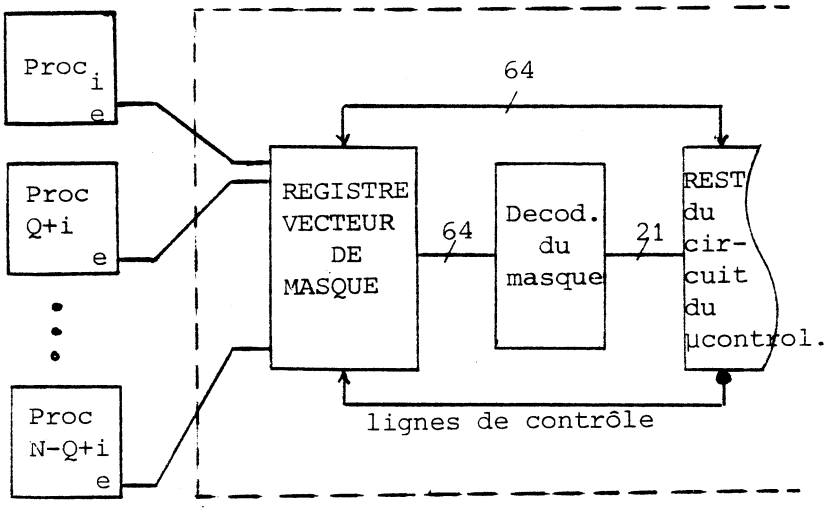
Schéma 1.14.

Ces microcontrôleurs permettent aussi l'activation ou la désactivation sélective des processeurs élémentaires. Cette possibilité est réalisée par un registre qui contient un vecteur de masque des processeurs. Dans ce vecteur chaque bit représente un processeur de base, sa valeur détermine son inclusion ou son exclusion de l'ensemble des PCU dans la configuration. Le circuit réalisant cette fonction dans les microcontrôleurs est donné par le schéma 1.15.

3. *Unité de contrôle du système:* Elle est constituée par un calculateur PDP-11. Elle est responsable des activités des autres constituants du système.

4. *Mémoire secondaire:* Cette mémoire secondaire contient les fichiers donnés ou programmes du système. Elle est composée de plusieurs modules mémoire.

5. *Système de contrôle de la mémoire secondaire:* Il permet de contrôler le chargement des modules de la mémoire secondaire par les différents périphériques.



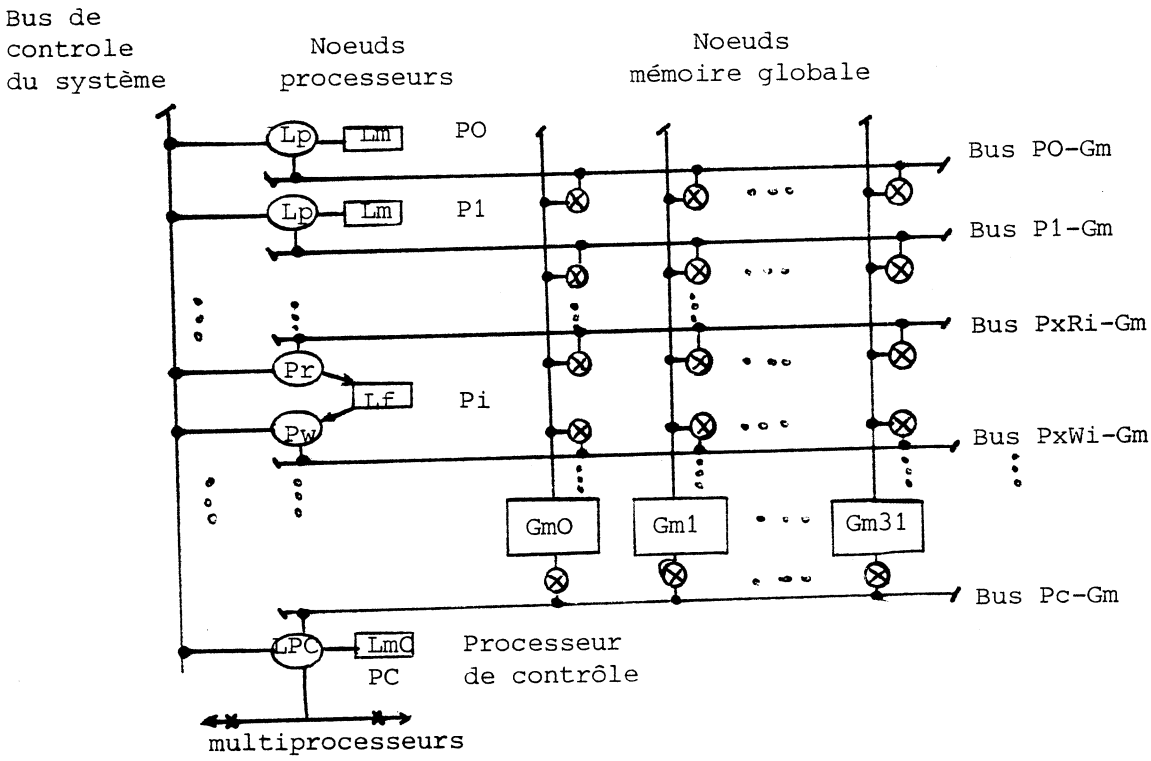
Réalisation du masque dans les micro-contrôleurs

Schéma 1.15.

1.2.2. *Système Multiprocesseur Entièrement Reconfigurable* (Fully Reconfigurable Multiprocessor)

Ce système est en cours de développement au Département Informatique du Laboratoire National de "LOS ALMOS" au Nouveau Mexique. (TRU 82).

Ce projet consiste à réaliser un système multimicroprocesseur construit d'un très grand nombre de microprocesseurs et de modules mémoire commercialement disponibles. L'architecture de base est de type M.I.M.D et elle supporte une reconfiguration de tous ses éléments. L'objectif de cette réalisation est de disposer d'un outil pour la recherche dans l'évaluation des algorithmes de traitement parallèle sur les différentes architectures multiprocesseurs.



Architecture du système (F.R.P.)

Schéma 1.16

Structure de la machine

1. Cette structure est caractérisée par un ensemble de nœuds de processeurs élémentaires et un ensemble de nœuds de modules mémoire dans une connexion matricielle ayant les caractéristiques suivantes:

   a. Un ensemble de bus reliant les processeurs élémentaires aux modules mémoire.

   b. Des mémoires multiports utilisant des commutateurs "Cross Bar" entre les nœuds processeurs et les nœuds mémoires.

   c. La reconfiguration entre les nœuds processeurs et les nœuds mémoires est réalisée par un "mapping" de la mémoire dans chaque nœud processeur.

2. On distingue trois types de nœuds processeurs:

   a. Processeur de contrôle du système (PC) qui se charge du télé-chargement des sites de développement des programmes, du contrôle de la reconfiguration, l'initialisation du système, les mesures de performances, la gestion des interruptions des microprocesseurs, etc...

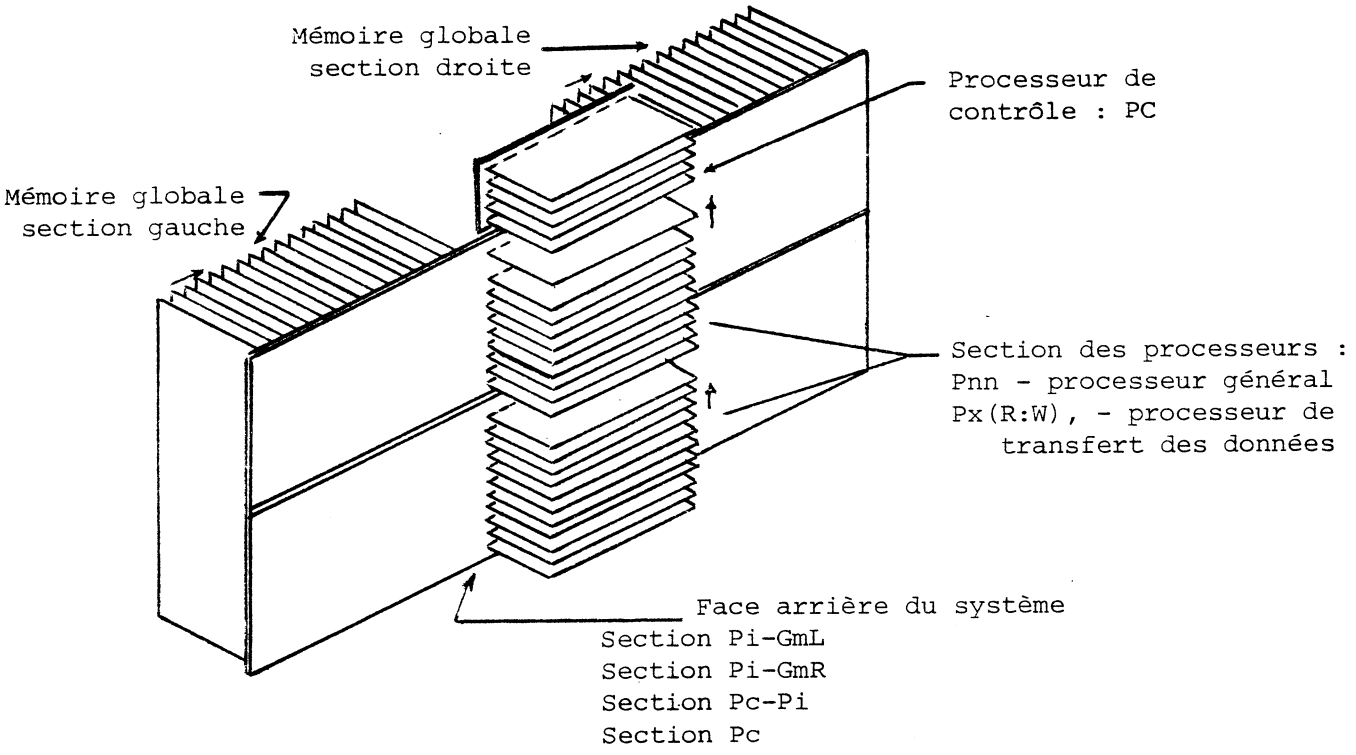
   b. Processeurs pour opérations flottantes (Pi) qui sont des IAPX 86/87 d'intel utilisant une mémoire locale de 48 K octets de ROM/RAM. Ils ont des circuits de mapping de la mémoire permettant de leur allouer 61 segments de la mémoire globale de 16 K octets chacun.

   c. Processeurs de transfert des données entre les segments de la mémoire globale Pxi.

3. La mémoire globale est décomposée en nœuds de 256 k octets de RAM chacun. Ces nœuds sont accessibles par le processeur de contrôle du système et des contrôleurs de la mémoire multiports. Les nœuds de la mémoire peuvent être alloués aux processeurs par le circuit de mapping. Ces segments mémoire seront utilisés comme mémoire globale ou comme mémoire privée aux processeurs.

   Un circuit d'arbitrage et un mécanisme de TEST & SET permettent une exclusion mutuelle d'accès aux modules de la mémoire commune.

4. L'assemblage orthogonal des nœuds mémoires et des nœuds processeurs (schéma 1.17.) permet d'utiliser des lignes de bus physiquement très courts réalisant ainsi des meilleures interconnexions.



Assemblage orthogonal du système (F.R.P.)

Schéma 1.17.

1.2.3. Architecture multiprocesseur (S.I.M.D.) partageant deux bus pour le traitement du signal en temps réel

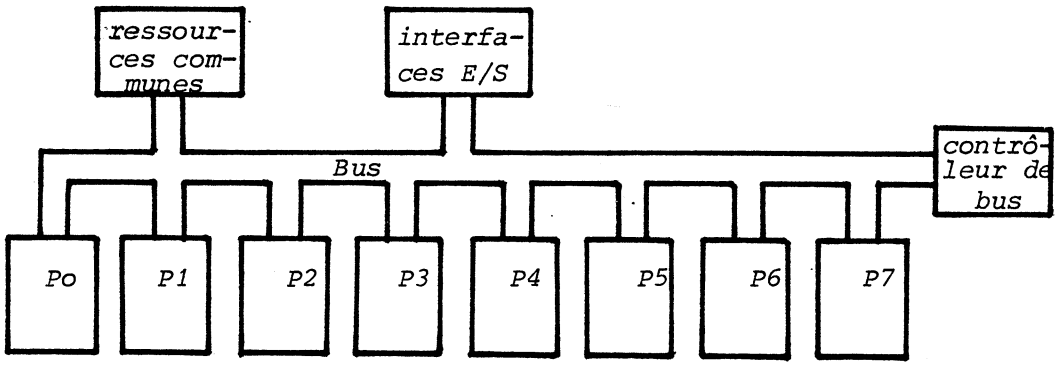
Ce projet réalisé à l'université libre de Bruxelles V.U.B en 1978, représente une alternative limitée de notre réalisation en S.I.M.D. Le logiciel d'exploitation et d'évaluation du système est en cours de développement.

La machine est construite autour de 8 microprocesseurs de type Z.80 (8 bits, 2 MhZ). Chaque processeur possède 24 K octets de mémoire locale. Ils se partagent 2 bus communs:

1. Le premier bus (Bus Rapide) est alloué à un des processeurs pour un cycle horloge. Il permet aux différents processeurs l'accès aux ressources communes.

2. Le deuxième bus (Bus Lent) est alloué à un des processeurs pendant plusieurs cycles horloge. Il permet à un instant donné à un des processeurs d'accéder à la mémoire privée d'un autre processeur. Ce dernier sera bloqué par une demande d'accès direct en mémoire DMA.

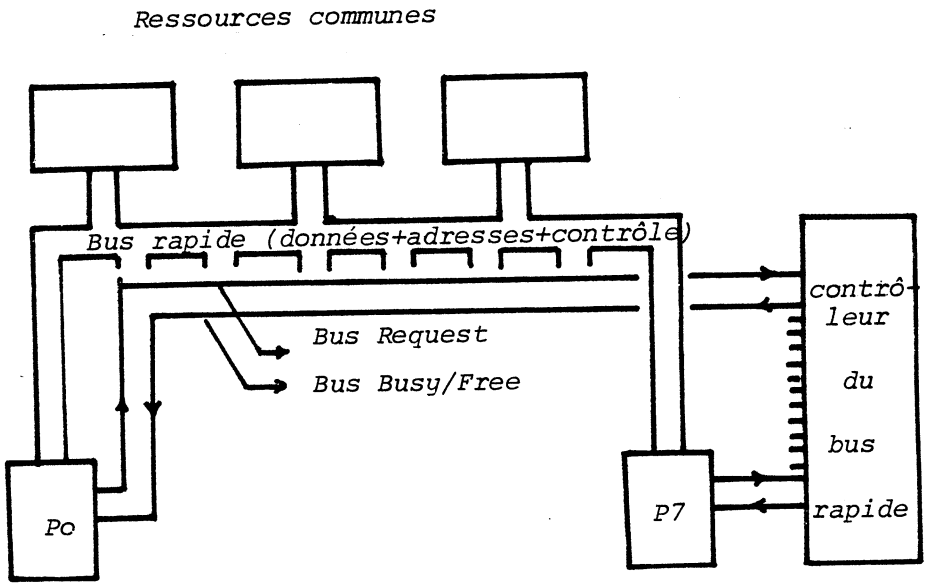
Les processeurs sont banalisés, mais chaque processeur est identifié par un numéro sur 3 bits.



ARCHITECTURE GENERALE

Schéma 1.18.

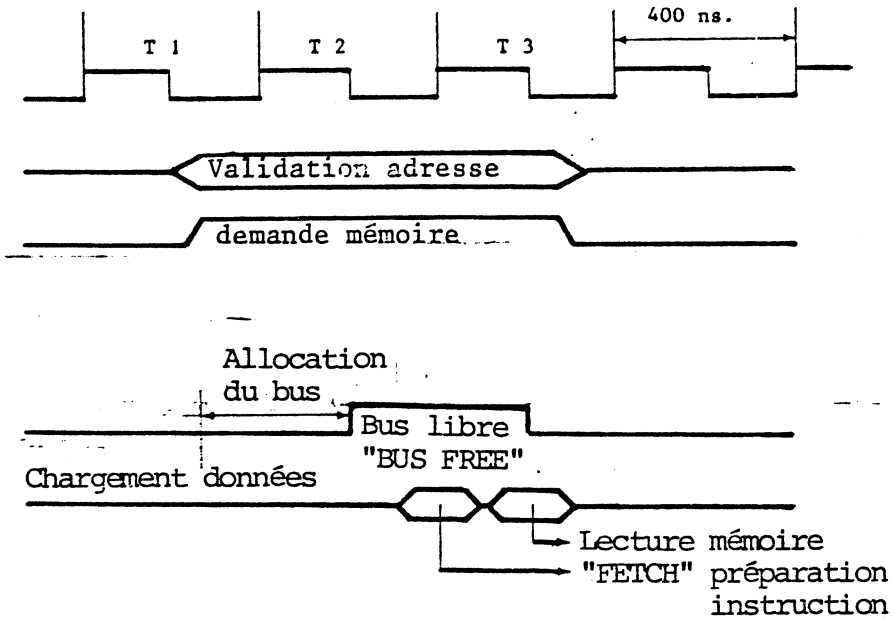
A. *Bus d'accès aux ressources communes (Bus Rapide)*



ORGANISATION DU BUS RAPIDE

Schéma 1.19

A la réception d'un signal "BUS BUSY" le processeur restera en attente par insertion des cycles wait jusqu'à réception du signal "BUS FREE". Le bus lui sera donc alloué pour une durée de 400 NS nécessaire au chargement des données dans son buffer. Les données lui seront ainsi disponibles même après la libération du bus.



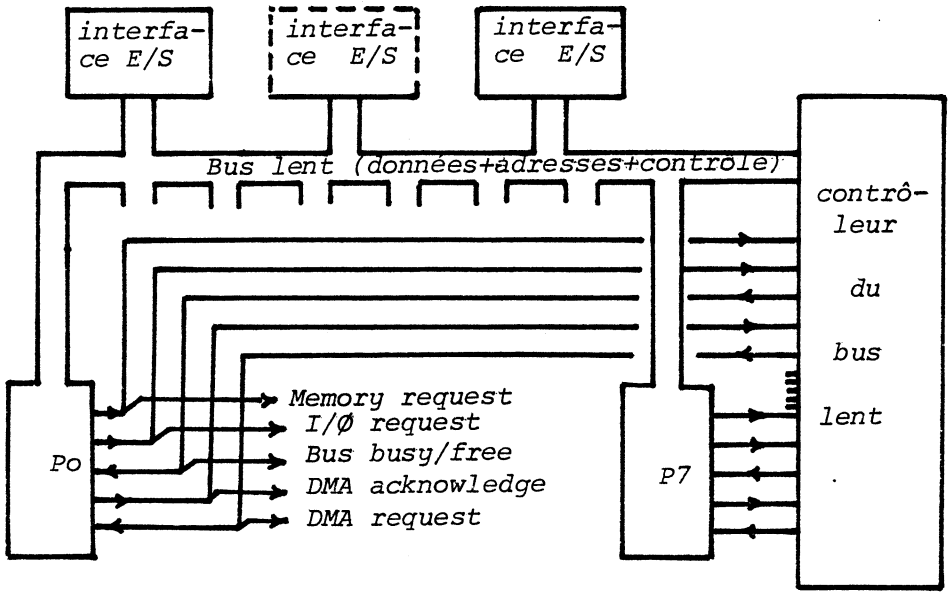
Evolution chronologique (TIMING) d'une lecture mémoire ou préparation d'une instruction sur le bus rapide

Schéma 1.20.

B. *Bus d'accès aux mémoires locales et aux interfaces d'E/S (Bus Lent)*

Chaque processeur désirant accéder à la mémoire privée d'un autre processeur signale au contrôleur de bus le numéro du processeur dont il recherche l'accès à la mémoire locale. Cette demande sera suivie d'une semande d'accès en mémoire. Le contrôleur de bus adresse au processeur ayant cette mémoire une demande d'accès direct en mémoire DMA et à la réception de l'acquittement sur cette demande, il envoie au processeur demandeur le signal "BUS FREE". Ce signal déclenchera l'opération de transfert des blocs de données voulus. L'allocation de ce bus est réglementée par une priorité circulaire.

On remarque dans ce projet que le choix du microprocesseur était le même que le nôtre. On justifiera ce choix dans la description de notre architecture.



ORGANISATION DU BUS LENT

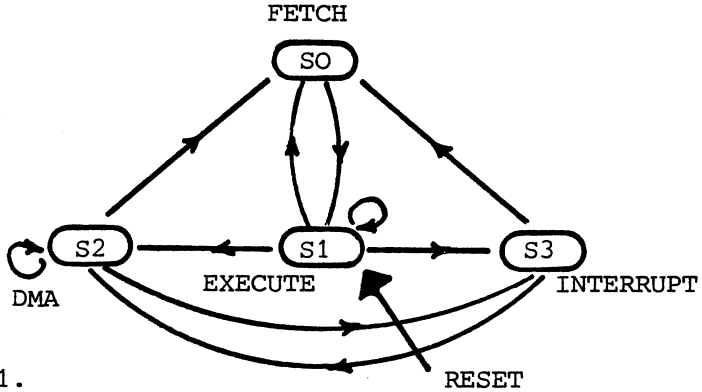
Schéma 1.21.

1.3. *RELATION ENTRE L'EVOLUTION DES CIRCUITS INTEGRES ET LE DEVELOPPEMENT DE PROJETS MULTIPROCESSEURS*

Avec le développement des circuits intégrés de coût toujours décroissant, les concepteurs ont essayé de fournir aux utilisateurs quelques facilités pour construire des systèmes multimicroprocesseurs. Le microprocesseur CDP 1802 de RCA (le COSMAC) donne l'exemple de cette tendance.

	Code état	
	SCO	SC1
Fetch SO	0	0
Execute S1	1	0
DMA S2	0	1
Interrupt S3	1	1

Table de vérité décrivant les états du CDP 1802 par rapport aux signaux en sortie SCO et SC1.

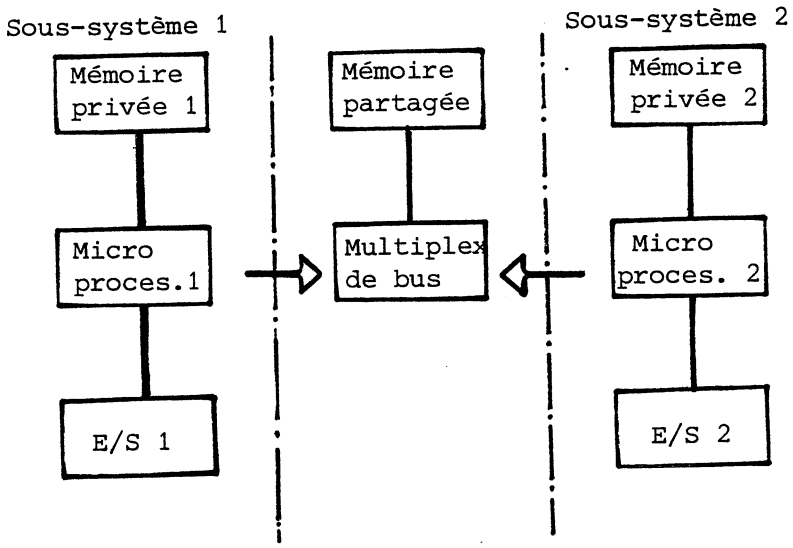


Automate état du Cosmac

Schéma 1.22.

Dans ce microprocesseur, deux codes d'états SCO et SCI sont à la disposition de l'utilisateur. Ces deux sorties du microprocesseur reflètent l'état du CPU selon le graphe et la table de vérités (schéma 1.22.).

Dans une configuration multiprocesseur de type S.I.M.D. à mémoire partagée comme celle du schéma 1.23, l'utilisateur peut exploiter l'état du microprocesseur pour résoudre les problèmes du conflit d'accès au bus commun avec le minimum de circuits de synchronisation possible.



Système à multiprocesseur avec une mémoire partagée

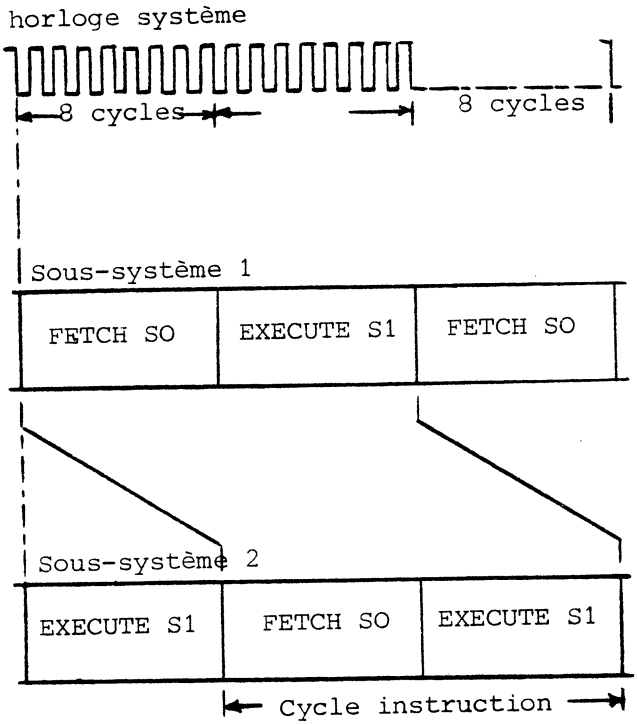
Schéma 1.23.

Cette configuration comporte une mémoire programme privée pour chaque processeur et une mémoire de données commune aux deux processeurs.

Sachant que dans l'état de recherche "FETCH" les instructions sont lues de sa mémoire programme et que c'est seulement dans l'état d'exécution "EXECUTE" que la mémoire de données est accédée, avec un simple décalage de 180° entre les phases "FETCH-EXECUTE" des deux microprocesseurs, on peut résoudre le conflit d'accès à la mémoire commune selon le schéma 1.24.

Le décalage de phases peut être conservé tant qu'on n'exécute pas quelques rares instructions qui insèrent des cycles d'exécution additionnels (par exemple l'instruction NOP et les instructions de branchements longs).

La réalisation de ce décalage peut être fait par logiciel en insérant des instructions NOP en début des programmes exécutés par un des deux microprocesseurs.

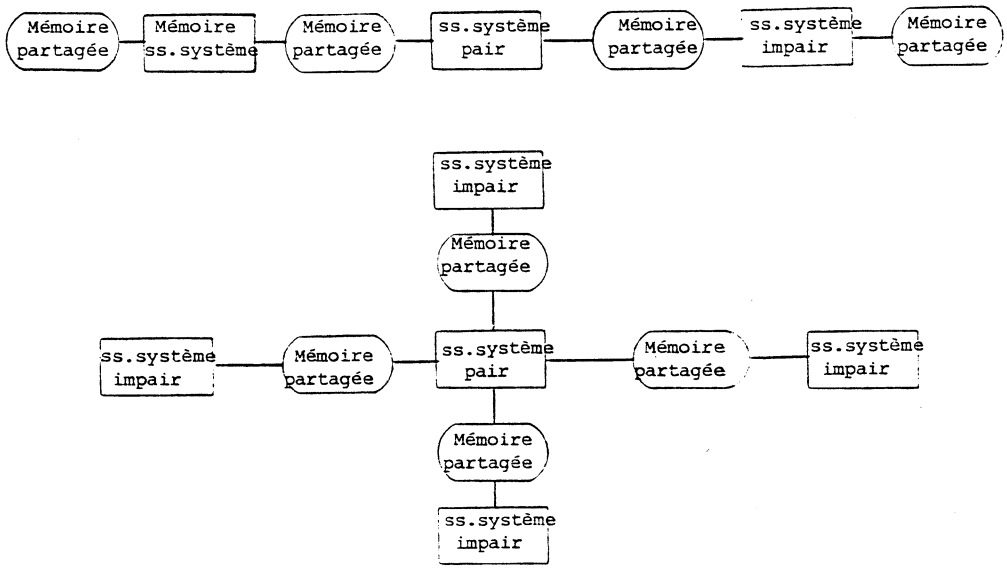


Décalage des phases FETCH EXECUTE
de 180° entre les sous-systèmes

Schéma 1.24.

Il est possible de généraliser ce principe et construire un système multimicroprocesseur constitué par deux sous-ensembles qu'on notera pair et impair. Ces deux sous-ensembles sont caractérisés par un décalage de 180° entre leurs phases "FETCH-EXECUTE". Cette architecture est représentée par le schéma 1.25.

En regardant l'architecture interne de ce microprocesseur, on remarquera que la partie opérative du circuit est probablement conçue par ordinateur. Cette conception automatique utilise un répertoire de modules pré-définis. La réalisation des microprocesseurs est considérablement accélérée, diminuant ainsi les prix des unités de traitement et favorisant ainsi leurs utilisations dans les systèmes multimicroprocesseurs.



Deux configurations multi-sous-systèmes à base du microprocesseur 1802 utilisant une mémoire partagée. Un décalage de 180° entre les phases FETCH - EXECUTE des sous-systèmes pairs et des sous-systèmes impairs.

Schéma 1.25.

CONCLUSIONS

L'évolution dans le domaine des circuits intégrés est une des principales causes de développement des projets multiprocesseurs. Malgré le fait que les constructeurs poursuivent leur courses pour proposer des microprocesseurs plus puissants (allant jusqu'aux tailles des mini-calculateurs), il sera toujours intéressant de construire des machines multiprocesseurs, car elles sont plus fiables, plus puissantes et moins chères que la machine de la même taille en monoprocesseurs. Nous prenons, par exemple, le rapport des prix entre un microprocesseur 8 bits et un microprocesseur 16 bits: il est de 1:10; on peut vérifier que la puissance d'un multimicroprocesseur ayant 10 processeurs de 8 bits est plus élevée que celle d'un seul de 16 bits. Ceci sans parler de la fiabilité qui est très importante dans certaines applications.

ooOOOOOoo

CHAPITRE 2

REALISATION D'UN SYSTEME MULTIMICROPROCESSEUR POUR LE TRAITEMENT DES ALGORITHMES PARALLELES

Dans ce chapitre on présente les caractéristiques du système multimicroprocesseur réalisé. La première partie de ce chapitre (chapitre 2-A) décrit l'architecture et les composants matériels du système. Dans la deuxième partie (chapitre 2-B) on développe les outils logiciels d'adaptation de la machine au traitement des algorithmes parallèles. Le prototype a été défini, réalisé et mis au point en collaboration avec Monsieur MIRZAVAZIRI HAMID qui soutiendra une thèse de Docteur Ingénieur sur ce sujet. Il représente 6 cartes au format 16.5 x 11 cm comprenant environ 15 circuits chacune. Le logiciel se trouve partie en mémoire ROM et partie en RAM chargée depuis les diskettes; il représente environ 2^7 k octets.

Chapitre 2-A

REALISATION MATERIELLE

DU SYSTEME

MULTIMICROPROCESSEUR

Dans la réalisation matérielle d'une architecture M.I.M.D. pour l'expérimentation des algorithmes parallèles nous recherchions la simplicité et la modularité.

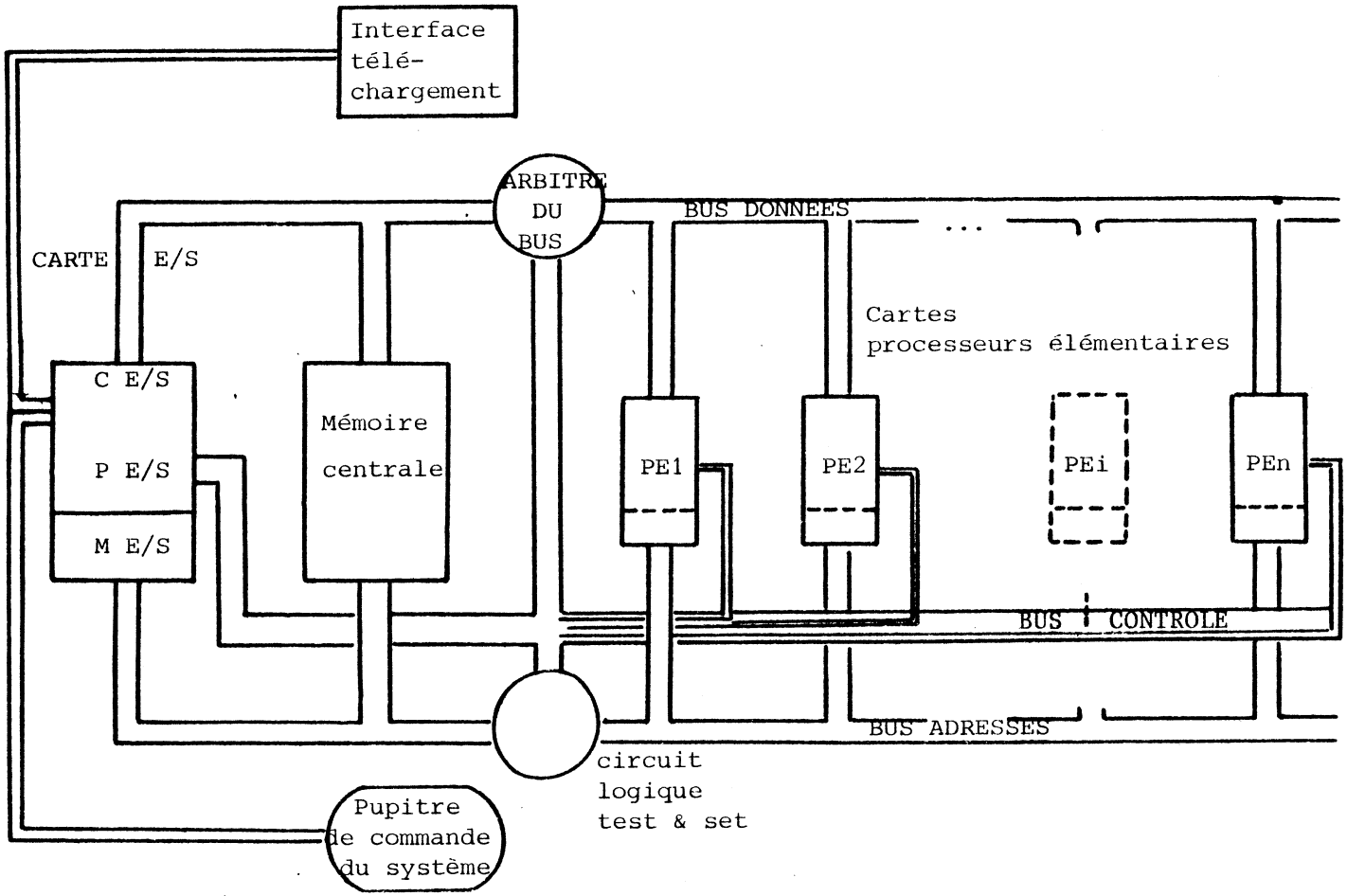
L'unité de traitement se compose d'un certain nombre de modules. Chaque module représente une carte processeur élémentaire construite autour d'un microprocesseur 8 bits du type "Z.80".

Ces processeurs élémentaires ou processeurs d'instructions communiquent entre eux (via un bus d'adresses et un bus de données standards) par l'intermédiaire d'une mémoire centrale.

Un processeur spécialisé aussi du type "Z.80" se charge des communications du système avec l'extérieur, via des liaisons standards séries asynchrones "RS.232".

Le schéma 2.1 représente le type d'architecture du système ainsi réalisé.

Schéma 2.1.



Bloc diagramme du système multiprocesseurs

2.A.1. *CARTES PROCESSEURS D'INSTRUCTIONS*

2.A.1.1. Structure

Ces cartes processeurs sont entièrement banalisées, ce qui nous permet d'augmenter ou de diminuer, à volonté, le nombre des processeurs élémentaires, sans aucune modification logicielle ou matérielle du système fournissant ainsi au fonctionnement une grande fiabilité.

Pourtant, il est évident que le nombre de modules influence les performances du système.

Chaque carte est constituée par un microprocesseur Z 80. Chaque processeur élémentaire peut interrompre le processeur d'Entrées/Sorties ou les autres processeurs grâce à ces portes d'E/S. Des circuits logiques leur permettent de disposer des données lues de la mémoire centrale après la libération du bus dans le contexte multiprocesseur.

Dans chaque carte, on dispose d'un ensemble de diodes d'affichage permettant ainsi le contrôle des activités des processeurs d'instructions.

2.A.1.2. Choix du microprocesseur

Il n'existe aucun microprocesseur commercialement disponible et qui puisse répondre au besoin des systèmes multimicroprocesseurs traitant des algorithmes parallèles. On essaiera de définir les fonctions nécessaires à un microprocesseur pour convenir parfaitement à ce type d'application. De toutes façons, on était contraint d'implémenter les fonctions non existantes par des moyens matériels et/ou logiciels afin de pouvoir exécuter ce type de programmes.

Le choix du microprocesseur a été fait en fonction des systèmes de développement dont on dispose et des critères suivants:

1. *La puissance:* Partant de l'idée que le microprocesseur sera utilisé dans une configuration multimicroprocesseur dans laquelle la puissance peut être augmentée en multipliant le nombre de processeurs élémentaires, ce critère n'a pas joué dans le choix du microprocesseur d'instructions.

2. *Le coût de la réalisation:* Comme on l'a déjà cité dans le chapitre précédent, on prendra le moins coûteux.

3. *Le rafraîchissement de la mémoire dynamique:* Le cycle de rafraîchissement fourni par certains microprocesseurs sera intéressant dans le cas de l'utilisation d'une mémoire dynamique qui est moins chère et qui occupe moins de place dans les cartes. Cette option est valable dans le cas du processeur d'E/S pour adresser sa mémoire locale d'E/S. Mais dans le cas des processeurs d'instructions, s'il n'utilisent pas de mémoires locales, ils utiliseront uniquement la mémoire commune. Cette mémoire commune doit être accessible par le processeur d'E/S qui ne peut pas la rafraîchir puisqu'il ne peut la modifier que par des instructions d'E/S à travers ces ports d'E/S. Donc ce critère sera exclu.

4. *Les registres internes des microprocesseurs:* Il était intéressant de choisir un microprocesseur qui puisse nous fournir une zone de mémoire interne "Registres Scratchpad"; cette zone pourra être utilisée pour le stockage des informations concernant le parallélisme, dans le cas du fonctionnement sans mémoire locale. Le Z 80 n'ayant pas cette zone, dispose de 3 paires de registres internes (B'C', D'E', H'L'). L'utilisation de ces registres n'est permise que par une instruction spéciale (EXX). Ces 6 mots sont suffisants pour satisfaire ce besoin.

5. *Le système d'interruption:* Pour les besoins de notre réalisation, il nous fallait deux systèmes d'interruption:

   a. Un système d'interruption masquable (I.R.Q.). Cette interruption sera utilisée pour alerter les processeurs d'instructions dans le cas d'éclatement d'un processus parallèle.

   b. Un système d'interruption non masquable (N.M.I.). Cette interruption est nécessaire dans le cas d'une demande d'arrêt des processus frères et descendants d'un processus donné par une primitive de S.E.P (FSAIS - Chapitre 2.B.3.3.).

Ces deux systèmes d'interruption sont fournis par la plupart des microprocesseurs. Cependant, dans le Z 80 à l'occurence d'une interruption, les processeurs, sans aucun chargement au préalable de vecteurs d'interruption, iront exécuter tous les mêmes routines d'interruptions (à l'adresse 38 H pour les I.R.Q. et à l'adresse 66 H pour les N.M.I.).

6. *Signaux du CPU facilitant l'arbitrage du bus:* Pour un fonctionnement parallèle des microprocesseurs dans le système multiprocesseur , il a fallu réaliser un circuit logique permettant de résoudre les conflits d'accès des différents processeurs à la mémoire commune. Ce circuit logique qu'on désigne par l'arbitre de bus, peut être réalisé selon des techniques différentes qui sont basées essentiellement sur les signaux de CPU. Parmi ces techniques, nous citons quelques unes en ce qui suit:

a. Méthode utilisant le signal BUS request ou DMA

En agissant sur ce signal on pourra bloquer le bus du microprocesseur (en haute impédance) au moins pour un cycle d'instruction complet. Cette méthode est utilisée (à côté d'un autre moyen d'arbitrage plus performant comme dans le projet cité dans le paragraphe 1.2.3.) pour accéder à la mémoire privée d'un autre processeur. Elle ne permet pas un véritable parallélisme puisqu'elle bloque les processeurs.

b. Méthode utilisant le signal MEMORY request

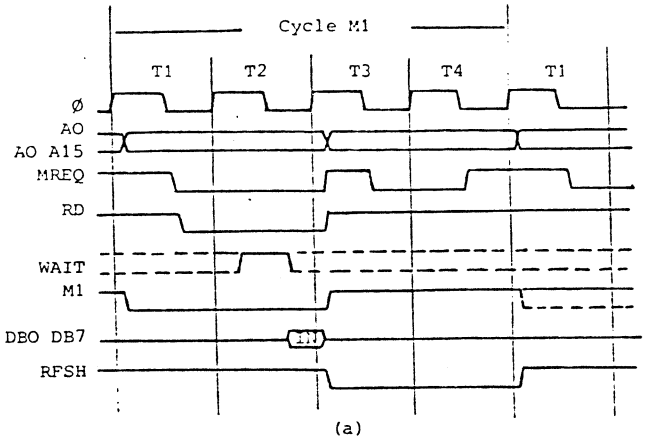
Au lieu de bloquer les processeurs pendant tout leur cycle instruction sachant que l'accès en mémoire occupe la moitié ou le tiers du cycle instruction, un signal donné par le CPU indique que le processeur va accéder en mémoire. Si l'on peut arrêter ou bloquer les processeurs jusqu'à libération du bus par le processeur qui l'utilise, on pourra faire fonctionner ces processeurs en parallèle. L'allocation du bus aux différents processeurs pour accéder à la mémoire commune peut être réalisé d'une façon synchrone ou asynchrone.

La méthode synchrone consiste à réaliser un décalage de phase de l'horloge de chaque microprocesseur. Ils peuvent accéder à la mémoire selon un ordre bien défini. Cependant, le nombre de processeur sera limité par le rapport cycle instruction/temps d'accès à la mémoire.

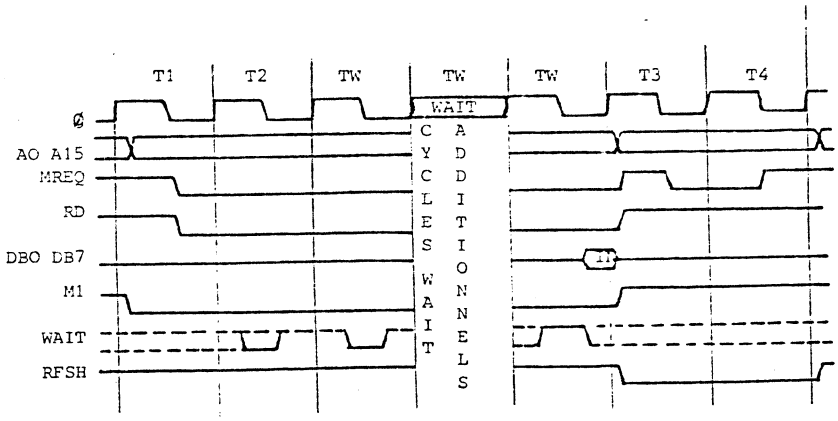
La méthode asynchrone permet un accès aux processeurs à leurs demandes (par le signal Memory Request) avec une priorité circulaire. Le blocage des autres processeurs se fera uniquement pendant l'accès à la mémoire et il est réalisé soit par arrêt de l'horloge des processeurs pendant la phase d'accès en mémoire, soit par le signal WAIT s'il est disponible dans le microprocesseur utilisé.

Les schémas 2.2. et 2.3. représentent l'utilisation du signal wait dans le Z 80 pour bloquer les processeurs en attendant la libération de la mémoire.

---- Schéma 2.2. ----

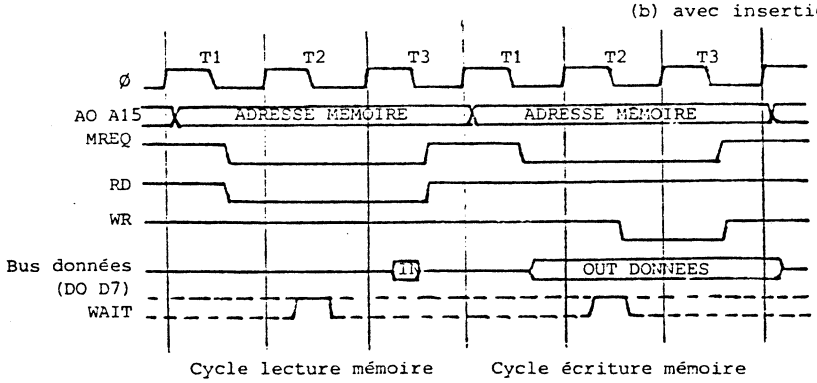


(a)

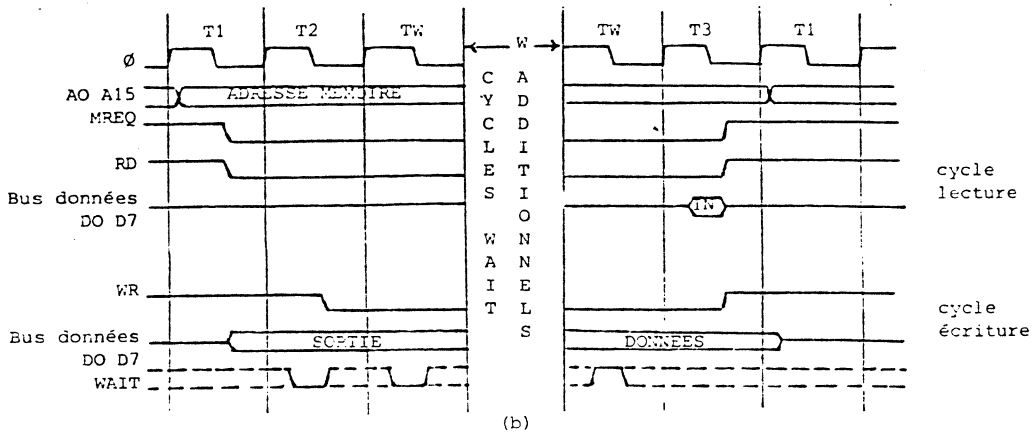


(b)

Séquencement de l'opération de recherche (FETCH) - (a) normale
(b) avec insertion cycles WAIT



(a)



(b)

Séquencement des cycles écriture et lecture mémoires - (a) normal
(b) avec insertion des cycles WAIT

Schéma 2.3.

---

7. *Instructions TEST & SET:* Ces instructions n'existent que dans les microprocesseurs 16 bits mais qui peuvent être réalisées par un circuit logique comme on le verra dans les paragraphes suivants.

Le microprocesseur Z 80 a été retenu comme processeur d'instruction pour son système d'interruption, ses registres internes et les signaux du CPU. La méthode asynchrone d'arbitrage du bus a été choisie afin de pouvoir utiliser un nombre variable de processeurs d'instructions dans un système multiprocesseur plus performant.

2.A.1.3. Mémoire locale

Il était possible, même préférable, d'utiliser dès le début de la réalisation une mémoire locale dans la carte processeur d'instruction. Cette mémoire pourrait contenir les routines correspondantes aux primitives du système d'exploitation parallèle, les variables locales et certains registres d'expression du parallélisme, ainsi que certaines zones de la mémoire centrale. Mais on a voulu évaluer les performances du système dans les deux cas.

La mémoire locale n'était pas indispensable puisque dans le microprocesseur retenu on disposait de trois paires de registres internes qu'on pouvait réserver comme registres de parallélisme; on verra l'usage de ces registres dans la description du système d'exploitation parallèle (chapitre 2.B).

L'utilisation d'une mémoire locale dans la carte processeur pourrait donc diminuer l'accès à la mémoire centrale et par suite améliorer les performances du système.

2.A.2. *PROCESSEUR D'ENTREES/SORTIES*

Cette carte permet au système de communiquer avec l'extérieur. Elle est constituée d'un microprocesseur Z 80 et deux interfaces de communications du type "SY6551".

2.A.2.1. Pupitre de commande du système

Cette interface permet à un opérateur de contrôler le fonctionnement du système. Un langage de commande lui offre la possiblité de réaliser une des fonctions suivantes:

1. Visualiser et/ou modifier le contenu de la mémoire centrale.

2. Initialiser le système, interrompre un ou plusieurs microprocesseurs d'instructions et fournir un compte rendu de l'état du système à n'importe quel moment.

3. Déclencher le téléchargement des programmes (via le deuxième U.A.R.T.) d'un calculateur hôte vers la mémoire centrale.

2.A.2.2. Téléchargement

Les programmes utilisateurs sont développés à l'extérieur sur une machine hôte capable de générer du code exécutable par les processeurs d'instructions. On disposait au laboratoire d'un système de développement de programmes TEKTRONIX LAB8002, on l'a utilisé comme calculateur hôte. Ce code est téléchargé en mémoire centrale à travers une liaison série asynchrone.

Le processeur d'Entrée/Sortie peut nous fournir à n'importe quel instant les adresses et la taille des différentes zones de la mémoire centrale qui sont libres pour le chargement des programmes d'applications. L'architecture du système nous permet d'utiliser un microprocesseur d'Entrée/Sortie de type différent de celui des processeurs d'instructions.

Si on veut augmenter la fiabilité du système, on peut utiliser un ensemble de processeurs d'Entrées/Sorties au lieu d'un seul.

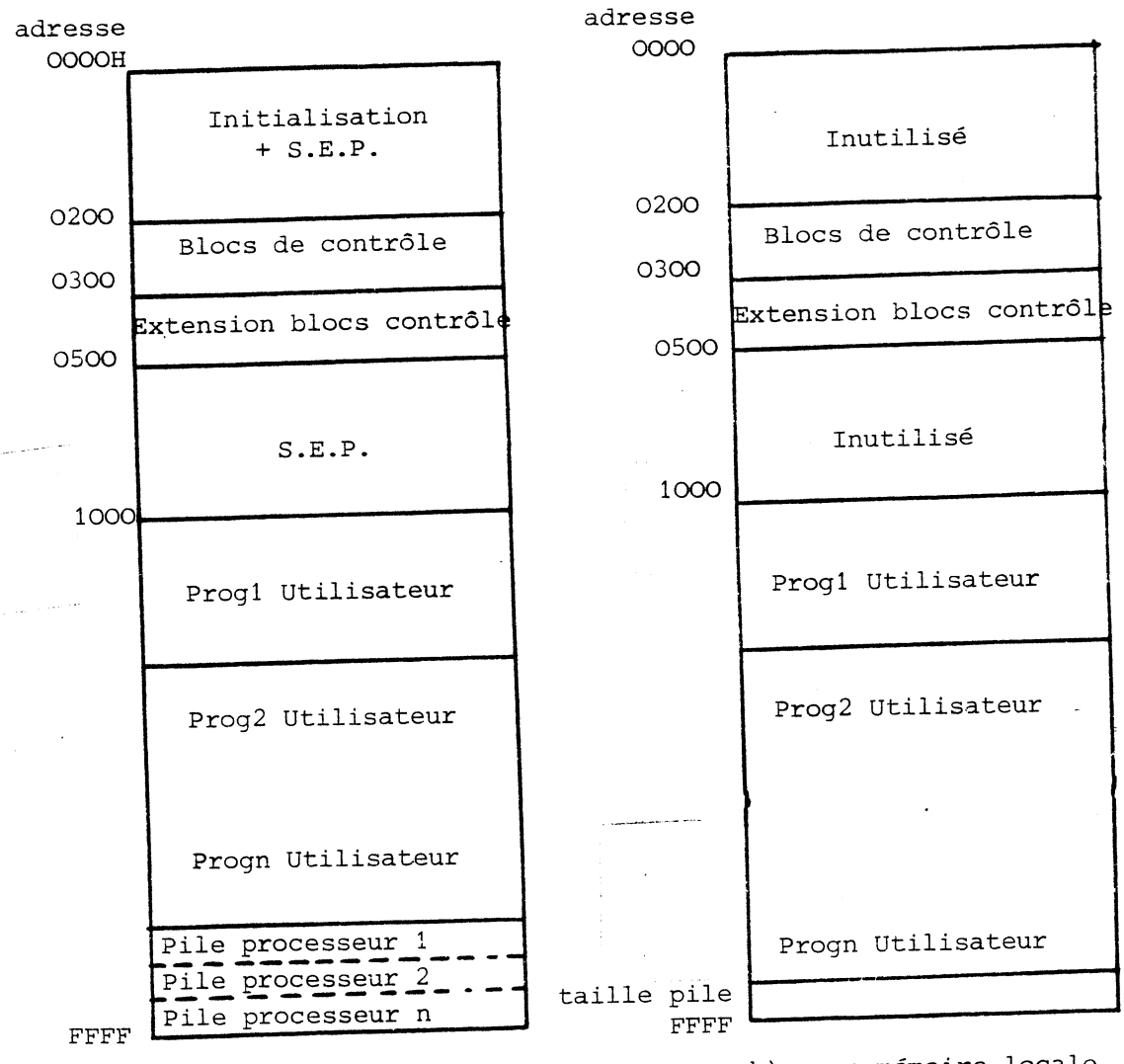
2.A.3. *MEMOIRE CENTRALE*

Les processeurs d'instructions communiquent entre eux par l'intermédiaire de 64 K octets de mémoire centrale. Le découpage des zones mémoires (schéma 2.4.) est fonction de l'option mémoire locale existante ou non.

Dans le cas d'utilisation d'une mémoire locale dans chaque carte processeur élémentaire, le Système d'Exploitation Parallèle (SEP) est résident dans une mémoire morte privée, une mémoire RAM est utilisée pour l'usage de la pile utilisateur par processeur ainsi que pour les variables locales.

La mémoire centrale est aussi accessible au processeur d'E/S pour le chargement des programmes. Elle contient les blocs de contrôle du parallélisme. Un mécanisme de TEST & SET permettant l'accès exclusif à cette section critique est réalisé par un circuit logique et utilisé par un test logiciel.

La réalisation de ce circuit est simple et basée sur un décodage d'une adresse de la mémoire et d'une génération d'une valeur dans cette zone (par hardware). Cette fonction existe dans le répertoire des instructions de la majorité des microprocesseurs 16 bits (68000 et Z8000).



Organisation de la mémoire centrale

Schéma 2.4.

CONCLUSIONS

Cette réalisation pourrait être améliorée par l'utilisation d'une mémoire modulaire avec des processeurs qui réalisent le cache-mémoire, des mémoires locales pourront réduire l'accès à la mémoire commune, et des sites de développement de programmes avec des langages de haut niveau qui feront appel aux primitives du S.E.P.

ooOOOOOoo

Chapitre 2-B

SYSTEME D'EXPLOITATION PARALLELE

Le système d'exploitation parallèle constitue l'ensemble des logiciels de base nécessaire au traitement des programmes parallèles. Il assure la synchronisation entre les différentes tâches logicielles des microprocesseurs de notre machine multimicroprocesseur.

Avant d'exposer les fonctions du système d'exploitation, il est nécessaire d'aborder le thème du parallélisme et des programmes parallèles.

2.B.1 *PARALLELISME ET PROGRAMMES PARALLELES*

La définition classique d'un programme est un ensemble d'instructions. Chaque instruction représente une action sur un ensemble de données qui définissent le contexte. Ces actions se déroulent selon un ordre défini à priori et qui peut être modifié selon certaines conditions relatives au contexte. Dans cette suite d'actions il est évident qu'une seule action s'exécute en un instant donné.

Il est à noter que selon le langage de programmation utilisé une instruction peut représenter un sous-ensemble d'actions qu'on se permet de définir comme une seule action. Selon l'approche de programmation séquentielle, chaque instruction ou ensemble d'instructions dépend des instructions ou des séquences d'instructions précédentes. En d'autres termes, les modifications apportées par une séquence d'instructions au contexte influencent les résultats des séquences d'instructions qui suivent.

Il est possible d'isoler certaines séquences d'instructions au sein d'un même programme de telle sorte que, si on exécute simultanément ces séquences à partir d'un contexte de départ commun, les résultats ne seront pas différents de ceux qui résultent de l'exécution séquentielle. On dira donc que ces séquences sont des séquences parallèles.

2.B.1.1. Parallélisme disjoint et parallélisme conjoint

Ce qui nous intéresse donc dans notre réalisation est le parallélisme interne aux programmes. Au sein d'un programme, on distingue deux types essentiels de séquences parallèles:

1. Des séquences dans lesquelles on exécute des actions distinctes sur des ensembles de données distinctes. Le résultat de l'exécution de l'une de ces séquences n'influence en aucun cas les résultats de l'exécution des autres séquences (*Parallélisme Disjoint*).

2. Une séquence d'instructions qui s'exécute d'une façon répétitive sur un ensemble de données un certain nombre de fois: c'est le cas dans une boucle (*Parallélisme Conjoint*).

   L'index de la boucle est lu et incrémenté au démarrage de chaque exécution de cette séquence. Dans le corps de la boucle l'accès à cet index n'est autorisé qu'en lecture (voir chapitre 3.2.).

On peut ajouter que, dans le cas des machines dont l'architecture est du type S.I.M.D. les séquences parallèles sont essentiellement des séquences d'instructions communes qui agissent sur les ensembles de données privées des processeurs élémentaires.

2.B.1.2. Parallélisme ET et parallélisme OU

Un type particulier de parallélisme est utilisé dans les architectures multiprocesseurs essentiellement pour une exécution plus rapide des programmes (ainsi que pour la fiabilité). Il est possible de lancer plusieurs processeurs sur des séquences différentes qui réalisent le même travail; la séquence qui se termine plus rapidement est validée, le processeur qui l'a exécutée continuera l'exécution du programme, les autres seront avortées. C'est ce qu'on définit par le parallélisme "OU". (Ce type de parallélisme est aussi utilisé par les machines "pipe-line" surtout dans les expressions conditionnelles. Dans ce cas, on lance plusieurs processeurs, chacun sur une branche, pendant qu'un des processeurs évalue la condition. Une fois que l'expression conditionnelle est évaluée, la branche qui correspond au résultat de la condition continuera et les autres seront arrêtées.)

Dans les systèmes à architecture M.I.M.D. le parallélisme "OU" peut être utilisé dans le cas où l'on a un grand nombre de processeurs (qui sont considérés comme une ressource qui n'est pas chère). Dans le chapitre 3 du présent document, on évaluera ce type de parallélisme dans un programme qui fait la recherche d'un élément dans un tableau.

Par opposition il est possible d'éclater certaines séquences d'instructions d'un programme en plusieurs branches qui peuvent être exécutées simultanément par plusieurs processeurs. Ces séquences ne peuvent se terminer que si toutes les branches issues de cet éclatement sont terminées. Ce type de parallélisme est le plus fréquent dans les programmes parallèles. On le définit comme étant le parallélisme "ET".

2.B.1.3. Détection du parallélisme

Dans les systèmes M.I.M.D. la détection du parallélisme interne aux programmes peut être réalisée par l'utilisateur. Une détection automatique de certains types de parallélisme est aussi possible.

- Dans le cas de la détection du parallélisme par l'utilisateur, celui-ci doit définir les séquences qui seront exécutées en parallèle: un travail qui consiste à éliminer les dépendances entre les séquences d'instructions, à définir les débuts et les fins de ces séquences et enfin à définir le début de la séquence qui suit les processus parallèles. Le programmeur doit être formé à une méthode particulière d'analyse et de programmation. Il devra se servir des différents algorithmes parallèles plutôt que de se contenter de traduire certaines séquences en séquences parallèles. Un programme parallèle peut paraître moins optimisé qu'un programme séquentiel.

- Dans le cas d'utilisation d'un langage de haut niveau d'expression du parallélisme, une autodétection de certains types de parallélisme peut être implémentée au niveau du compilateur de ce langage. Des langages qui expriment le parallélisme, on cite le langage Pastoral, le Pascal concurrent, le langage LTR et le langage ADA. On verra dans la suite des idées sur la traduction d'une boucle de traitement n fois en séquences exécutables en parallèles.

2.B.2. *LOGICIEL D'EXPLOITATION DU PARALLELISME*

Le logiciel réalisé sous le titre de système d'exploitation parallèle ne constitue pas un système d'exploitation complet d'une machine, car il permet uniquement à l'utilisateur d'exploiter le parallélisme interne à ses programmes par l'intermédiaire d'un certain nombre de primitives d'expression du parallélisme. Il permet aussi l'initialisation des processeurs, le chargement des programmes et l'arrêt des processeurs. Ce système d'exploitation est inspiré en grande partie du "MINI-OS" du projet MCS. Les

divergences majeures entre notre système et le Mini-Os sont en général dans la manière d'alerter les différents processeurs d'instructions pour s'attribuer des travaux et dans les primitives d'arrêt des processus frères et descendants d'un processus donné. C'est pour exploiter les possibilités offertes par les mécanismes d'interruptions et les registres internes des microprocesseurs que ces modifications ont été apportées au système d'exploitation parallèle.

Les termes FORK et JOIN ont été utilisés par M. CONWAY en 1963 pour exprimer les opérations d'éclatement d'un processus pour une exécution parallèle à mener sur un autre processeur. Le terme JOIN correspondait à la synchronisation en fin d'exécution de ces branches parallèles (miniprocessus).

Les différentes primitives d'éclatement de processus (FORK), de synchronisation en fin de miniprocessus (JOIN), d'éclatement d'une itération en séquences parallèles (NFORK), de synchronisation en fin des processus d'itération parallèle (NJOIN) et d'arrêt forcé des processus (FSAIS) ont été réalisées et l'overhead dû à leur traitement a été évalué sur la machine multimicroprocesseur décrite dans la première partie de ce chapitre.

Avant d'exposer les fonctions des différentes primitives du S.E.P., on expliquera, dans ce qui suit, les moyens de communication entre les processeurs.

2.B.2.1. Communications interprocesseurs

Une zone de blocs de contrôle située en mémoire centrale permet la communication entre les microprocesseurs. Comme les processeurs sont entièrement banalisés, les blocs de contrôle deviennent des boîtes aux lettres communes à tous les processeurs.

Chaque bloc de contrôle utilisé correspond, en général, à un éclatement de processus, en 'n' séquences parallèles. Les informations qu'il contient reflètent l'état de ce processus ou plutôt l'état des branches générées par son éclatement.

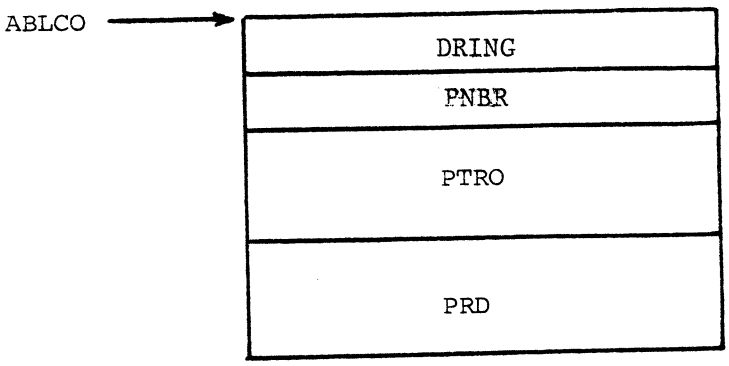
Les blocs de contrôle contiennent aussi une sauvegarde du contexte à chaque éclatement ainsi que des indicateurs sur le type de parallélisme et les différents chaînages entre ces blocs. Une vérification du contenu de ces blocs nous permet d'avoir une trace d'exécution des différents processeurs dans le système.

43

I. Description des blocs de contrôle

Les blocs de contrôle ont une taille fixe qui est définie à l'initialisation du système. Une taille minimum de 11 octets est nécessaire, elle correspond aux informations nécessaires aux parallélismes. A celle-ci s'ajoute une zone de sauvegarde du contexte ayant, par défaut, la taille de 14 octets. Donc, un bloc de contrôle occupe, par défaut, 25 octets mémoire, sauf avis contraire de l'utilisateur à l'initialisation.

a. *BLOC 0:* Seul le premier bloc est différent. Il est de 6 octets de taille. Il contient les ancrages des 2 types de chaînage des blocs de contrôle, un compteur de processeurs en ligne et la sonnette (indicateur de travail à démarrer). Ce bloc 0 se trouve à l'entête de la zone réservée pour les blocs de contrôle.



Structure du bloc 0

Schéma 2.5.

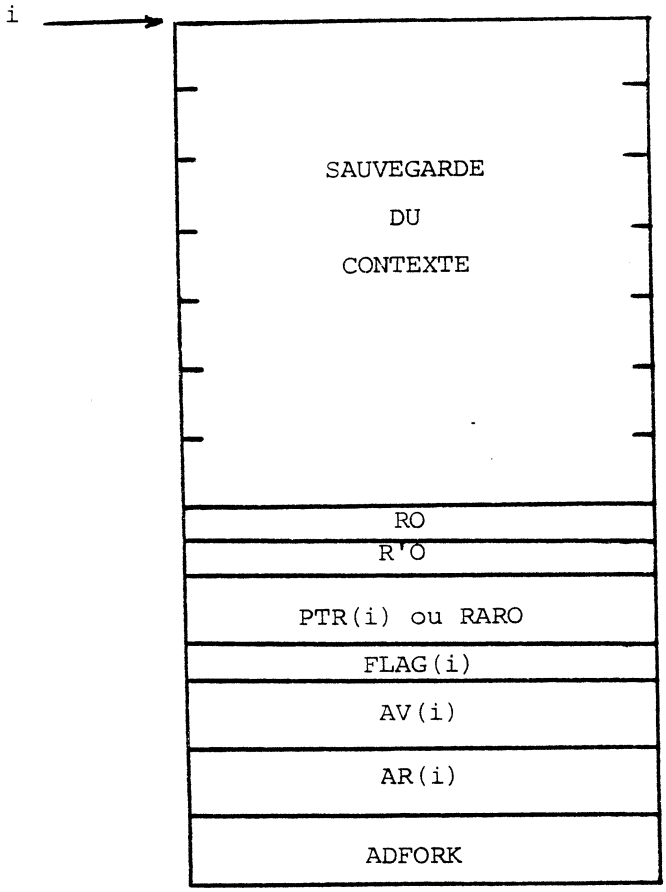
DRING: (*Un octet*) Sonnette ou indicateur d'éclatement de processus

PNBR: (*Un octet*) Compteur des processeurs en ligne

PTRO: (*Deux octets*) Tête de liste pour les blocs libres, pointeur vers le dernier bloc libéré

PRD: (*Deux octets*) Pointeur vers le bloc du dernier FORK où il y a encore des branches à démarrer

a. *BLOC i:*



Structure d'un bloc i

Schéma 2.6.

CONTEXTE: A part les registres internes du microprocesseurs, les variables locales sont sauvegardées dans cette zone

R'O : Compteur des branches qui n'ont pas encore démarré

RO : Compteur des branches qui ne sont pas terminés

PTR (i) : Pointeur pour le chaînage des blocs libres

AV (i) : Chaînage avant de la liste des FORKS où il reste des branches

AR (i) : Chaînage arrière de la liste des FORKS où il reste des branches à démarrer

FLAG (i) : Drapeaux indiquant l'état et la nature du processus éclaté. Le bit du poids plus fort de ce flag indique, s'il est à 1, que c'est le bloc de fin de chaîne dans le chaînage des blocs qui contiennent des processus où il reste des branches à démarrer. Les 2 bits poids faible de l'octet du poids fort de ce mot indiquent le type d'éclatement de processus. La valeur "00" représente un FORK et "10" représente un NFORK. Dans ce flag, on trouve aussi un indicateur (bit 3 de l'octet poids faible) que le processus est à avorter par la primitive FSAIS.

ADFORK : Adresse du descripteur qui contient les informations relatives au processus à éclater dans le programme utilisateur. Ce descripteur est accédé uniquement en lecture.

II. Chaînage des blocs libres

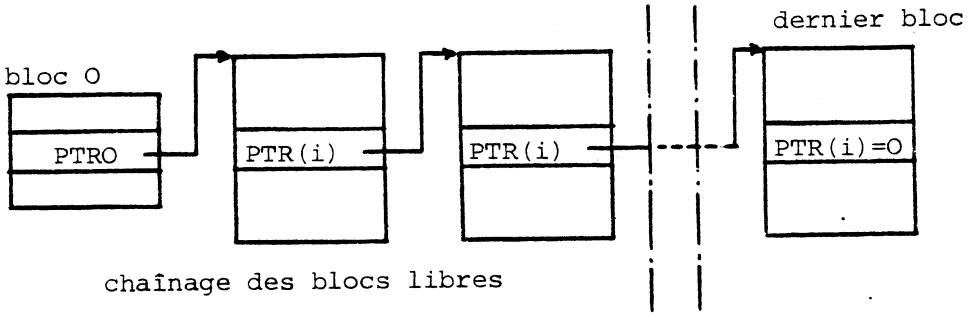


Schéma 2.7.

Ces blocs sont alloués et restitués dynamiquement. La valeur 0 du pointeur vers le bloc libre suivant indique que c'est le dernier bloc dans la liste. Le bloc n'est restitué que si tous les mini-processus issus du processus éclaté sont terminés.

a. *Allocation d'un bloc:* L'allocation d'un bloc (i) est donnée par l'algorithme suivant:

Début

i ← PTRO

/Si i = 0 *Alors* (il n'y a plus de bloc libre)
Exécution séquentielle

/Finsi.

PTRO ← PTR (i)

Fin.

b. *Restitution d'un bloc:* La restitution d'un bloc (i) se fait en début de la liste selon l'algorithme suivant:

Début

PTR (i) ← PTRO

PTRO ← i

Fin

III. Chaînage des blocs de la liste des FORKS ou il reste des branches à démarrer

Les blocs contenant des informations relatives aux processus éclatés et qui contiennent des miniprocessus qui n'ont pas encore commencé sont liés par un chaînage double. A l'affectation du dernier de ces miniprocessus à un processeur, celui-ci supprime le bloc du chaînage mais le bloc n'est pas restitué à la liste des blocs libres, donc il reste alloué au processus éclaté.

L'insertion et la suppression d'un bloc du chaînage sont décrites par les algorithmes suivants:

a. *Queue (i):* Insertion d'un bloc dans la liste des FORKS où il reste des branches à démarrer. On dispose d'un double chaînage. La tête de liste est sauvegardée dans le bloc de contrôle i = 0 dans PRD et elle est mise à jour à chaque insertion. Si la liste est vide, PRD contient 0.

Le pointeur AR de la tête de liste pointe sur lui-même et il en est de même pour le pointeur AV de fin de liste.

QUEUE (i)

```
Début
      DRING = 1
/Si   PRD NE 0      Alors  AV (i) ← PRD
                           AR (i) ← i
                           AR(PRD) ← i
                           EOF (i) ← 0

                    Sinon  AV (i) ← i
                           AR (i) ← i
                           EOF (i) ← 1

/Finsi
PRD ← i
Fin QUEUE.
```

b. *DEQUEUE (i):* Suppression d'un bloc de la liste des FORKS où il reste des branches à démarrer.

DEQUEUE (i)

```
Début

/Si   PRD = i    Alors
                 /Si  EOF (i) = 1    Alors  PRD ← 0
                                            DRING ← 0

                                     Sinon  PRD ← AV (i)
                                            AR (AV (i) ← AV (i)
                 /Fsi
                 Sinon
                 /Si  EOF (i) = 1    Alors  AV (AR (i) ← AR (i)
                                            EOF (AR (i) ← 1

                                     Sinon  AV (AR (i) ← AV (i)

                 /Fsi

/Fsi

Fin DEQUEUE
```

IV. Sonnette (DRING)

L'objectif de la sonnette est d'informer les microprocesseurs d'instructions de l'existence de travaux à exécuter. La sonnette est réalisée par un moyen matériel qui fait sortir les processeurs de leurs boucles d'attente et un drapeau qui leur signale qu'il reste encore du travail à faire. Mais pour exécuter un tel travail, il faut leur communiquer les informations nécessaires pour accomplir leur mission.

Les processeurs peuvent être dans deux états:

1. Libres dans une boucle d'attente

2. Occupés par du travail

Les processeurs qui sont libres seront alertés par une interruption de type IRQ (masquable). Les processeurs qui ont déjà du travail ont leur interruption masquée et ne peuvent répondre à cette alerte.

Les processeurs ainsi interrompus demanderont, par le moyen du Test & SET réalisé, un accès exclusif à ce drapeau. S'ils trouvent ce drapeau où cette sonnette positionnée, ils iront exécuter une séquence dans le S.E.P. qui leur permette de s'attribuer du travail. Une fois que les branches sont épuisées, le processeur qui s'alloue la dernière branche disponible supprime le bloc de la liste des FORKS où il reste des branches à démarrer et il remet la sonnette à zéro évitant ainsi aux autres processeurs de faire des démarches inutiles. Ceux-ci retourneront à leurs boucles dans l'attente d'une nouvelle alerte.

V. NMI (Interruptions Non Masquables)

Ce signal d'interruption n'est utilisé que dans le cas d'une demande d'arrêt forcé des processus frères et descendants d'un processus donné. On verra dans l'algorithme de cette primitive comment on utilise ce signal d'interruption pour réaliser le parallélisme "OU".

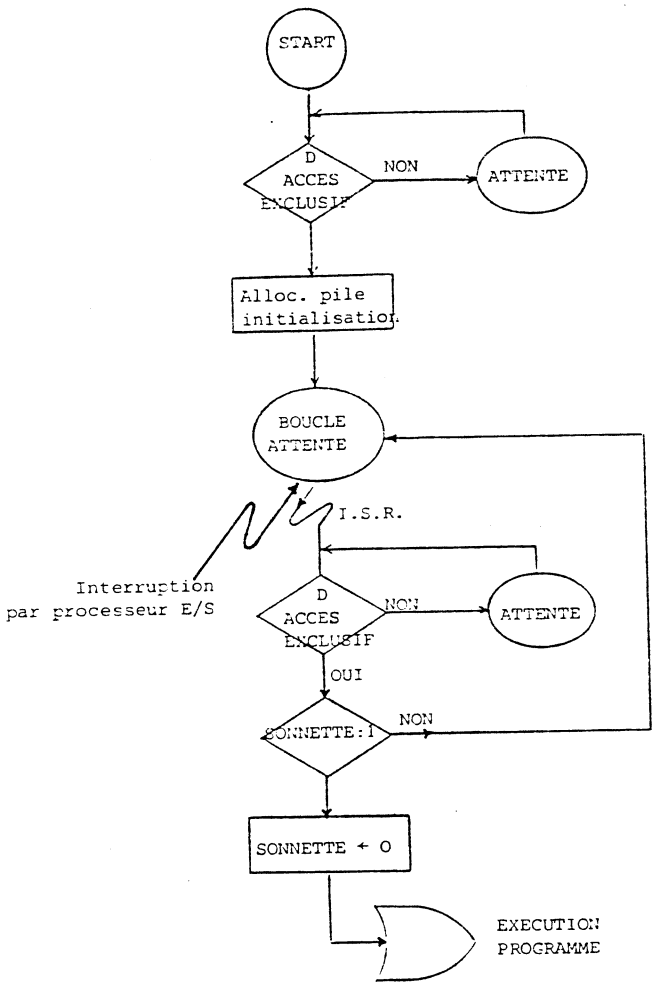
2.B.2.2. Initialisation du système multimicroprocesseur

Avant de lancer le téléchargement du S.E.P et des programmes utilisateurs par le processeur d'E/S, celui-ci, en réponse à la commande "HALT" nous permet d'arrêter tous les processeurs d'instructions. Une

49

fois que les programmes sont chargés, on demandera au processeur d'E/S par la commande "RUN" de lancer les processeurs d'instructions par le signal "RESET".

Dans le cas d'absence des mémoires locales, les processeurs d'instructions vont s'initialiser et s'allouer une zone de la mémoire centrale pour leur pile utilisateur, ensuite, ils se mettront en attente d'une alerte au travail selon l'organigramme suivant:



Séquence initialisation et démarrage d'un programme utilisateur

Schéma 2.8.

Après l'initialisation des processeurs d'instructions, et avant l'exécution des programmes utilisateurs, il faut lancer le programme d'initialisation des blocs de contrôle. Ce programme sera exécuté comme un programme utilisateur. Le processeur d'E/S chargera l'adresse de démarrage de ce programme dans la zone STRTAD qui se situe dans l'I.S.R. (Interruption Service Routine). Il positionnera la sonnette à 1 déclarant l'existence d'un programme utilisateur à exécuter. C'est seulement dans ce cas (chargement d'un programme) que le processeur d'E/S envoie une interruption générale aux processeurs d'instructions. Les processeurs d'instructions interrompus de leurs boucles d'attente vont se précipiter pour exécuter la routine qui dessert l'interruption à l'adresse 38h. Ils demanderont un accès exclusif à cette routine. Seul le processeur le plus rapide se lancera à l'exécution du programme après avoir remis à zéro la sonnette. Et ainsi les autres processeurs retourneront à leurs boucles d'attente.

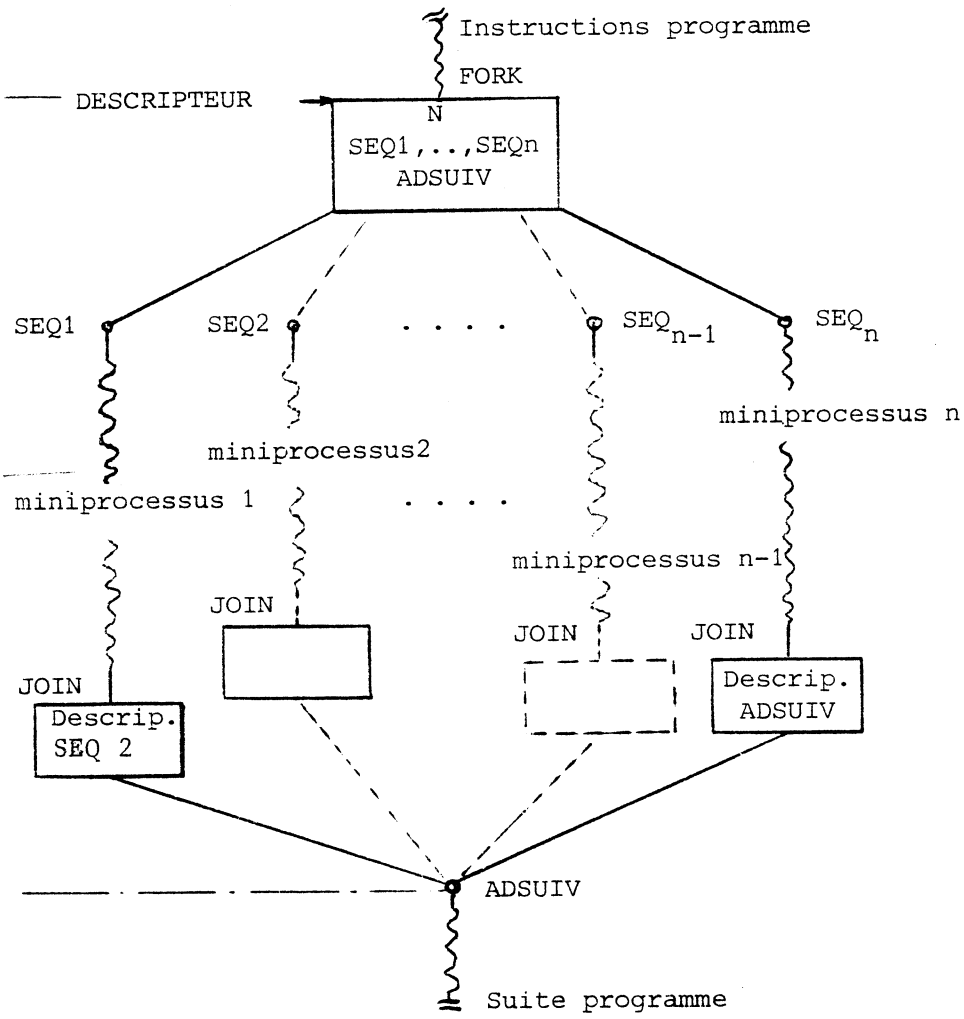
2.B.3. *PRIMITIVES D'EXPRESSION DU PARALLELISME DANS LE S.E.P*

En ce qui suit nous donnons le fonctionnement des différentes primitives d'expression du parallélisme.

2.B.3.1. Primitives FORK et JOIN

Ces primitives permettent à l'utilisateur de découper une partie de son programme en un certain nombre de séquences qui sont susceptibles d'être exécutées en parallèle. Il fera appel en début du processus à la primitive FORK en lui fournissant comme paramètres le nombre de branches parallèles, les adresses de début de ces branches et l'adresse de début de la séquence qu'il désire exécuter à la terminaison de ces séquences. Ces paramètres seront regroupés dans une zone désignée par le descripteur du FORK. A la fin de chacune de ces séquences, il devra faire appel à la primitive de synchronisation des fins de branches "JOIN" et en lui fournissant deux paramètres l'adresse du descripteur du FORK correspondant et l'adresse de la séquence suivante qui sera utilisée dans le cas d'une exécution séquentielle de ces branches résultant de la saturation des blocs de contrôle. En ce qui suit nous donnons un exemple d'utilisation de ces primitives.

✱ On appellera "miniprocessus" l'exécution d'une des branches parallèles résultant de l'éclatement d'un processus.



Parallélisme disjoint utilisant FORK-JOIN

Schéma 2.9.

```
     PROGRAMME ....

     ....      ....

     ....      ....

     ....      ....

     CALL      FORK  (Demande d'exécution paral. de n séq.)
DESCRIPTEUR

     BYTE      N     (Nombre de branches)

     WORD      SEQ1, SEQ2,... SEQn

     WORD      ADSUI (Adresse de la séq. à la fin du proc.)
SEQ1   Début de la première séquence

     ....      ....

     ....      ....

        Fin de la première séquence

     CALL      JOIN

     WORD      DESCRIPTEUR

     WORD      SEQ2
SEQ2   Début de la deuxième séquence

     ....      ....

     ....      ....

     ....      ....

     CALL      JOIN

     WORD      DESCRIPTEUR

     WORD      ADSUI
ADSUI  Séquence qui sera exécutée à la fin de toutes les branches
FIN
```

A l'entrée de la primitive FORK le processeur qui l'exécute demande un accès exclusif en mémoire pour pouvoir modifier les informations communes dans les blocs de contrôle. Il s'alloue un bloc de contrôle (s'il n'y a plus des blocs libres, il exécutera les branches l'une après l'autre; pour cela il utilise son registre interne B' comme un compteur de ces FORKS qu'on appellera "les FORKS ratés"). Il garnit les blocs de contrôle par les informations relatives aux exécutions parallèles (contexte, nombre de branches restantes, type de parallélisme...) et insère ce bloc dans la liste des blocs où il reste des branches à démarrer. Ensuite, il positionne la sonnette et envoie une interruption générale à tous les processeurs d'instructions.

Les processeurs d'instructions qui exécuteront le reste des branches vont mettre à jour ces informations. Leur accès est réglementé par le Test & Set.

Durant l'exécution de la primitive JOIN à la fin d'une branche parallèle si le processeur ne trouve plus de branches dans le processus père il cherchera du travail dans un autre processus parallèle. Il ne retournera dans sa boucle d'attente que s'il n'y a plus de branche dans aucun processus qui attend d'être traitée. Par conséquent, la primitive JOIN ne consiste pas seulement à terminer un miniprocessus, mais aussi à démarrer éventuellement un autre miniprocessus issu du même FORK.

2.B.3. 2. Primitives NFORK et NJOIN

Ces primitives permettent à l'utilisateur d'exprimer l'exécution parallèle des séquences d'une boucle. L'index de la branche exécutée est disponible à l'utilisateur dans le registre C' du microprocesseur Z. 80 (ou dans la mémoire locale si elle existe). Il faut cependant faire attention à l'utilisation de cet index pour indicer une variable. Supposons une boucle qui exploite une image et crée à partir de cette image une autre image, si l'image résultante remplace l'image d'origine (dans la même zone mémoire) le résultat est imprévisible.

Pour utiliser ces possibilités le programmeur fera l'appel en début de la boucle à la primitive NFORK en lui fournissant 3 paramètres nécessaires à l'exécution parallèle: le nombre d'exécution demandées de la boucle, l'adresse du début de la boucle et l'adresse début de la séquence qu'il désire après la boucle.

A la fin de la boucle il devra aussi faire appel à la primitive NJOIN en lui précisant le descripteur du NFORK correspondant et l'adresse de la séquence qui suivra la boucle.

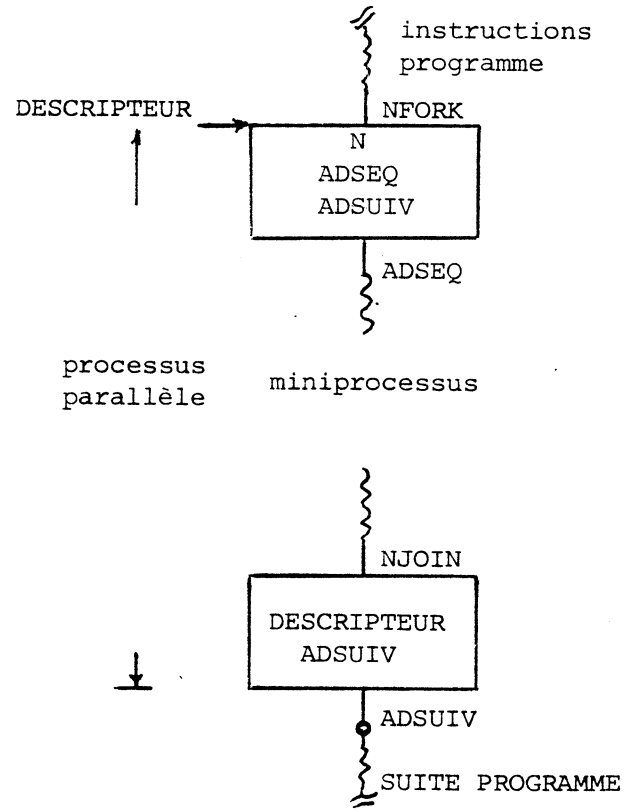
2.B.3.3. Primitive FSAIS

Cette primitive permettra d'arrêter les miniprocessus frères, c'est-à-dire issus du dernier FORK exécuté, réalisant ainsi le parallélisme "OU".

Le processeur exécutant cette primitive demandera un accès exclusif aux blocs de contrôle pour vérifier s'il y a des miniprocessus à tuer. Il positionnera l'indicateur dans son bloc de contrôle que ce miniprocessus est à avorter, il enverra une interruption non masquable aux processeurs d'instructions et commencera l'exécution de la branche qui suit ce processus comme si toutes les branches étaient terminées.

Les autres processeurs interrompus iront par un accès exclusif consulter leurs blocs de contrôle et ceux de leurs pères pour vérifier s'ils sont concernés. S'ils ne sont pas concernés, ils récupéreront leurs contextes et continueront leurs travaux. Les autres avorteront leurs branches, libéreront leurs blocs si c'est nécessaire et retourneront à leurs boucles d'attente.

Les primitives de S.E.P. peuvent être imbriquées.



Parallélisme conjoint utilisant NFORK-NJOIN

Schéma 2.10.

CHAPITRE 3

EVALUATION DU SYSTEME MULTIMICROPROCESSEUR DANS LE TRAITEMENT DES ALGORITHMES PARALLELES

Le but de ce travail et de la réalisation de cette maquette était d'évaluer l'efficacité de l'architecture et des mécanismes de gestion du parallélisme proposés.

Dans ce chapitre on présente l'évaluation des performances du système réalisé dans le traitement d'un certain nombre d'algorithmes parallèles. Ces programmes ont été choisis pour réaliser les objectifs suivants:

- Couvrir le plus possible les différents types de parallélisme: parallélisme disjoint et parallélisme conjoint
- Evaluer les performances du système dans le cas des processus parallèles imbriqués et ressortir des résultats obtenus les problèmes de saturation des blocs de contrôle
- Evaluation de l'exécution des boucles DO du fortran en parallèle selon certaines techniques
- Calculer le gain en temps dans l'exécution des programmes utilisant le parallélisme "OU"

Ces tests permettent aussi de calculer l'overhead dû à l'exécution des primitives du S.E.P. ainsi que la pénalisation de l'exécution résultante de l'arbitrage du bus.

Tous les programmes exécutés sur la machine sont écrits en Assembleur Z 80. La durée d'exécution globale d'un programme sur le système de développement (en mode monoprocesseur) est évaluée au moyen d'un analyseur temps réel (Real Time Trace Analyser Tecktronix LAB 8002). C'est avec cet analyseur que l'on a évalué le temps écoulé pour l'exécution de chacune des primitives du parallélisme.

Pour chaque type de parallélisme, le programme d'évaluation correspondant est réalisé en deux versions: une version séquentielle à exécuter sans faire appel aux primitives du S.E.P sur monoprocesseur évidemment. Cette version nous sert de référence pour évaluer l'intérêt de l'exécution parallèle. Dans l'autre version, on fait appel aux primitives correspondantes aux types de parallélisme utilisés.

L'évaluation des durées d'exécution des différents programmes parallèles sur le prototype est réalisée selon une méthode simple utilisant le processeur d'E/S, le système d'interruption et de l'analyseur temps réel que nous allons développer ci-dessous:

Après la phase d'initialisation du système, du téléchargement des programmes et d'initialisation du chaînage des blocs de contrôle (décrite dans le chapitre précédent) le processeur d'E/S lance les processeurs d'instructions à l'exécution du programme parallèle. Il devient ensuite disponible pour exécuter une boucle de comptage (logiciel). Le processeur d'instruction qui termine le programme envoie une interruption non masquable (NMI) au processeur d'E/S et arrête ainsi le processus de comptage. La durée d'exécution du programme est déduite de la lecture du contenu de ces compteurs qui se trouvent dans le RAM privée du processeur d'E/S.

La valeur d'une unité de comptage est donnée par l'analyseur temps réel en exécutant la boucle sur le système de développement. Les durées sont données en micro ou en millisecondes. Ces tests ont été réalisés sur la maquette présentée au chapitre 2 avec 4 cartes processeurs d'instructions et sans mémoire locale.

Il est à noter que les évaluations étroitement liées aux performances du système multimicroprocesseur du point de vue matériel:

- Temps de réponse du système d'interruption utilisé dans la sonnette et l'arrêt des processus.
- Performance du Test e Set
- Arbitrage du bus en multimicro

nécessitent un moyen de mesure plus précis que l'analyseur temps réel du système de développement.

57

Pour répondre à ce besoin, on a mis en œuvre une carte d'évaluation spécialisée. Elle est constituée d'un processeur Z 80, des compteurs CTC 80 et d'une liaison série RS232C pour le dialogue de ce processeur avec l'utilisateur.

Cette carte sera utilisée dans le suivi de ce projet et les résultats des évaluations des performances matérielles seront présentés dans un rapport ultérieur.

3.1. *EVALUATION DU PARALLELISME DISJOINT*

Nous avons pris pour évaluer ce parallélisme l'exemple d'un programme couramment utilisé en traitement d'image.

Ce programme consiste à superposer deux images de 64 x 64 points. Chaque point a la valeur 1 s'il appartient à une figure de l'image, sinon sa valeur est de 0. La superposition des deux images consiste à additionner les deux matrices correspondantes aux données des images. L'image résultante sera donc constituée de points ayant pour valeurs 0, 1 ou 2. La valeur 0 indique que le point n'appartient à aucune figure de l'image, 1 correspondra à l'appartenance à une seule figure et 2 correspondra à l'appartenance à 2 figures de l'image résultante (Figures1, 2 et 3).

Cette opération est réalisée en parallèle: l'image est découpée en 8 segments de 512 points chacun, le traitement des segments est réalisé en parallèle par les différents microprocesseurs de base.

Le premier processeur qui exécute le programme segmentera donc les images et fera appel à la primitive FORK. Il garnira le bloc de contrôle et se lancera au traitement des premiers segments après avoir interrompu les autres processeurs. Les autres processeurs exécuteront chacun un segment de chaque image et quand ils termineront leurs traitements, ils s'attaqueront aux segments restants. Le processeur qui terminera le dernier miniprocessus affichera une valeur dans les diodes prévues dans les cartes d'instructions pour contrôler l'activité des processeurs. Les autres processeurs retourneront à leurs boucles d'attente.

Il est important de signaler que les données des images à l'intérieur d'un segment ne sont utilisées que par un seul processeur dans chaque branche parallèle. C'est ce qui nous a permis d'écrire dans une des images d'origine. Il est évident que si un processeur devait écrire dans un segment qui peut être lu par un autre processeur, le résultat serait imprévisible.

Primitives utilisées: FORK et JOIN

```
      PROGRAMME      SUPIMG

Début
      tableau entier SEG11(512), (SEG12(512),... ,SEG18(512)
                     SEG21(512), (SEG22(512),... ,SEG28(512);

+++   FORK     (8, SEQ1, SEQ2,...,SEQ8,AFFICH)

SEQ1 :         pour i:=1 jusqu'à 512 faire
                    SEG21(i) = SEG11(i) + SEG21(i) ;

+++   JOIN

SEQ2 :         pour i:=1 jusqu'à 512 faire
                    SEG22(i) = SEG12(i) + SEG22(i) ;

+++   JOIN

SEQ8 :         pour i:=1 jusqu'à 512 faire
                    SEG28(i) = SEG18(i) + SEG28(i) ;

+++   JOIN

AFFICH :       (AFFICHAGE PERMANENT)

Fin
```

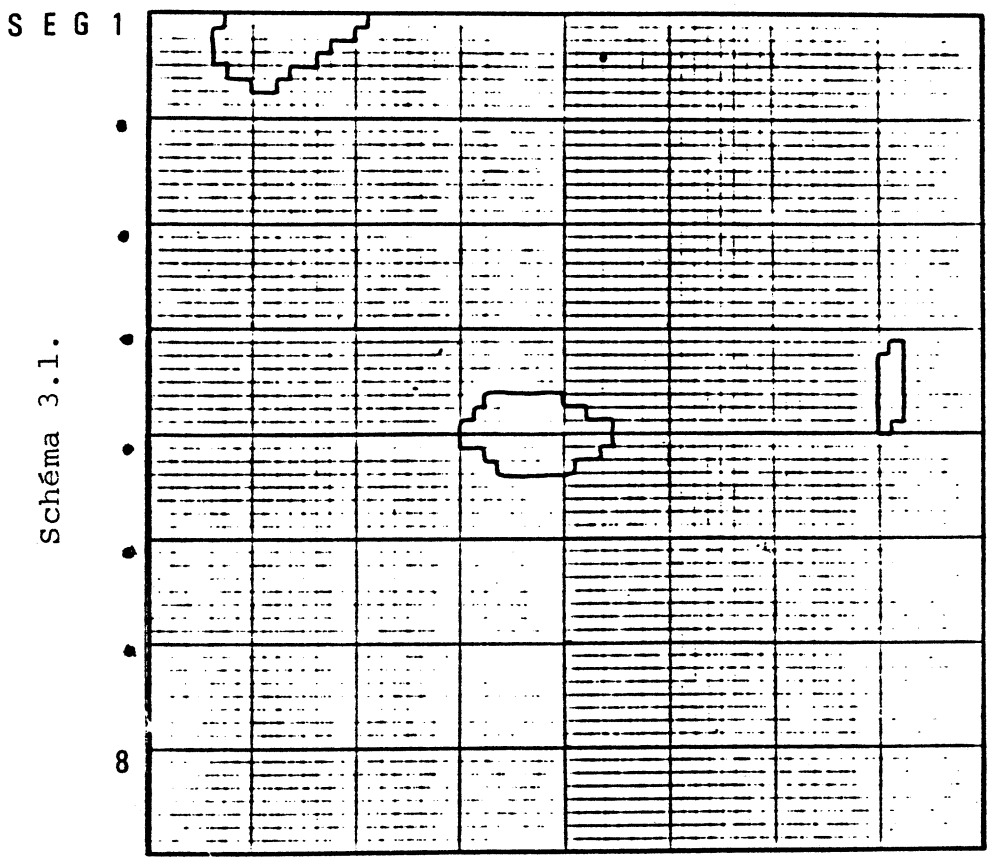


Schéma 3.1.

IMAGE 1

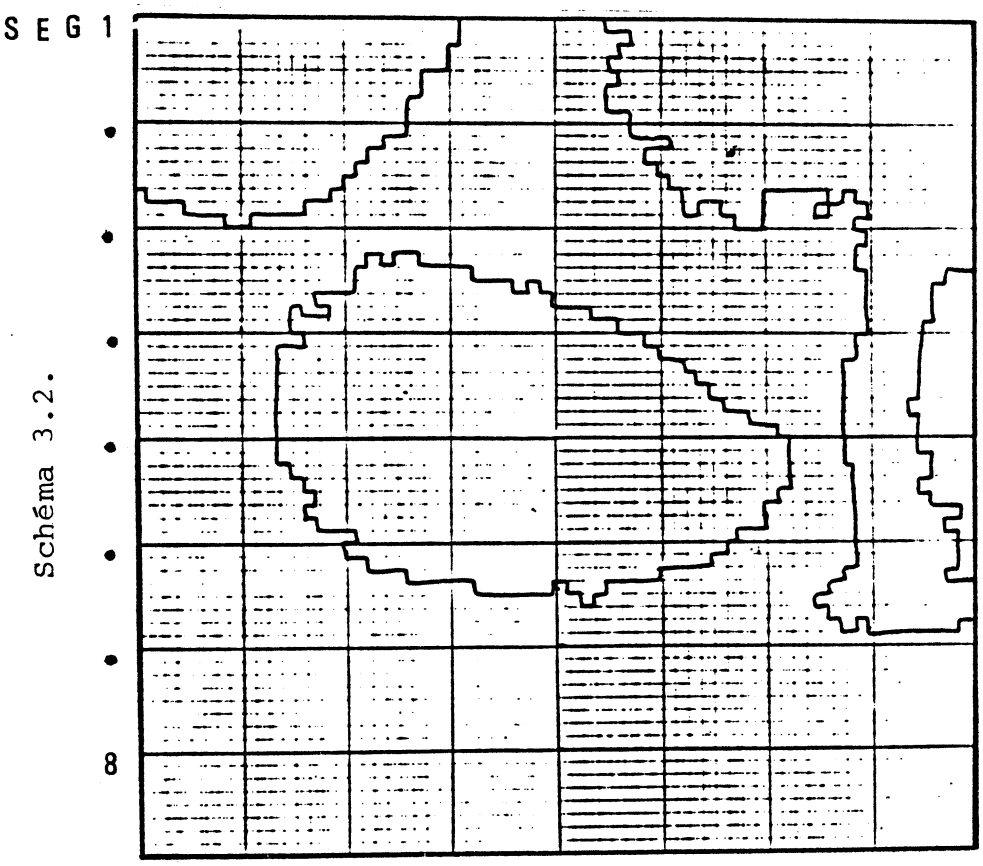


Schéma 3.2.

IMAGE 2

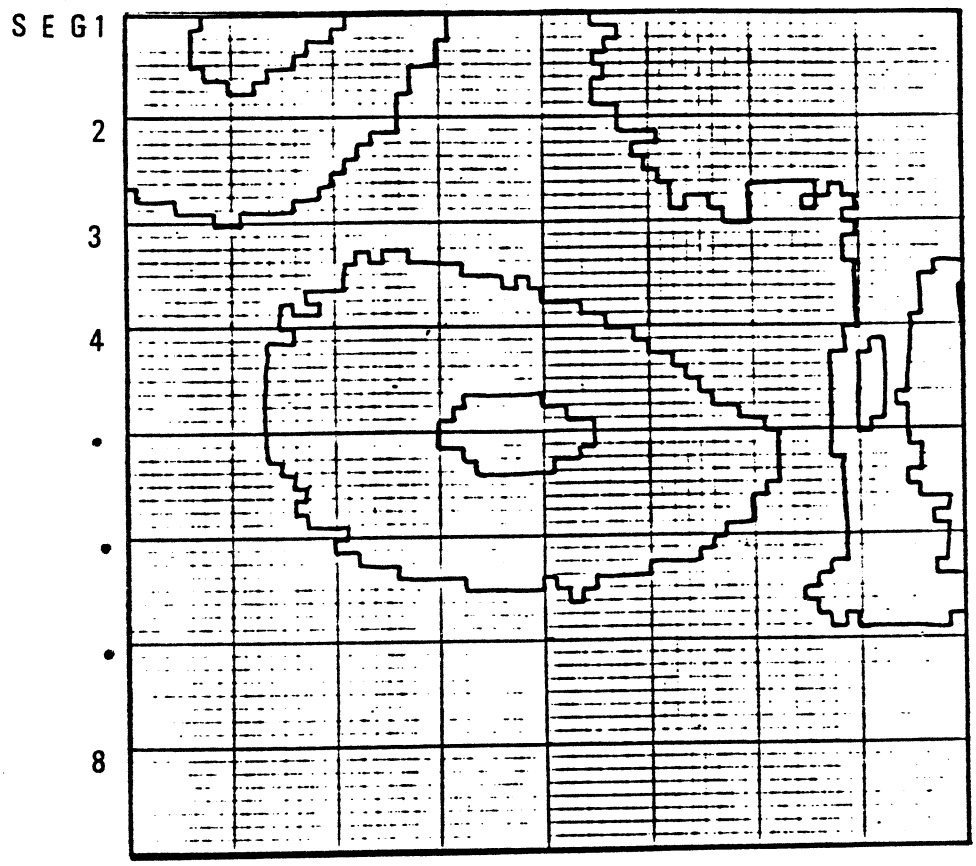


IMAGE RESULTANT DE LA SUPERPOSITION
DE DEUX IMAGES

Schéma 3.3.

METHODES D'EVALUATION

1. Ce programme est exécuté sur le système de développement avec le même type de microprocesseur en mode monoprocesseur. Les résultats sont comparés avec l'exécution sur la maquette avec un seul processeur.

2. Le programme est exécuté ensuite sur le système multimicroprocesseur avec 1, 2, 3 et 4 microprocesseurs d'instructions. On aura ainsi le temps global d'exécution du programme dans chaque cas. Ce test est suffisant pour nous permettre de déduire la relation entre le temps d'exécution et le nombre de processeurs exécutant le programme.

3. Le troisième test consiste à exécuter le programme sur le système multimicroprocesseur avec un seul processeur sans parallélisme. Le programme sera modifié pour qu'un seul processeur superpose les deux images sans segmentation. On pourra évaluer ainsi l'overhead dû à l'exécution des primitives du système d'exploitation parallèle.

RESULTATS

- Durée d'exécution de la primitive FØRK = 469 µsec

- Durée d'exécution (moyenne) de la primitive JOIN..... = 330 µsec

Nombre de Processeurs	Durée d'Exécution (ms)	Remarques
1 uP	132	Sur prototype
2 uP	78	"
3 uP	61	"
4 uP	48	"
1 uP	132	Système de Développement
1 uP	129	Séqentiellemment

Taille mémoire nécessaire pour le programme parallèle: 316 octets

Taille mémoire nécessaire pour le programme séquentiel: 238 octets

CONSTATATIONS ET COMMENTAIRES

1. Le temps d'exécution du programme parrallèle sur le système de développement (en monoprocesseur) est le même que celui d'un seul processeur sur la maquette. La méthode asynchrone étant utilisée, l'effet de l'arbitrage du bus paraîtra dès l'utilisation du deuxième processeur. Dans le cas où le nombre de processeurs ne dépasse pas le rapport cycle instruction/accès mémoire (3:1) et ils exécutent exactement le même code l'overhead dû à l'arbitrage du bus sera quasiment nulle: ses processeurs s'autosynchronisent et le décalage de phase créé au démarrage sera conservé comme si on était en mode synchrone.

2. On constate une réduction importante de la durée d'exécution du programme en mode multiprocesseur. Le rythme de cette réduction est ralenti avec l'augmentation du nombre des processeurs élémentaires. Ce ralentissement est dû à l'overhead de l'exécution des primitives du parallélisme. Ces primitives qui manipulent des variables dans la zone de communication interprocesseur , s'exécutent en section critique. L'attente d'entrée dans cette section augmente avec le nombre de processeurs utilisés et constitue ainsi un des facteurs importants qui provoquent ce ralentissement. Pourtant, on constate qu'en utilisant quatre processeurs, la durée d'exécution du programme est réduite à 36 % de la durée globale d'exécution du programme en monoprocesseur. Ceci est représenté par le schéma 3.4.

3. Les résultats de l'expérimentation montrent que l'overhead dû uniquement à l'exécution des primitives FØRK-JOIN du S.E.P est de 2,3 % de la durée globale de l'expérience en monoprocesseur (schéma 3.4.).

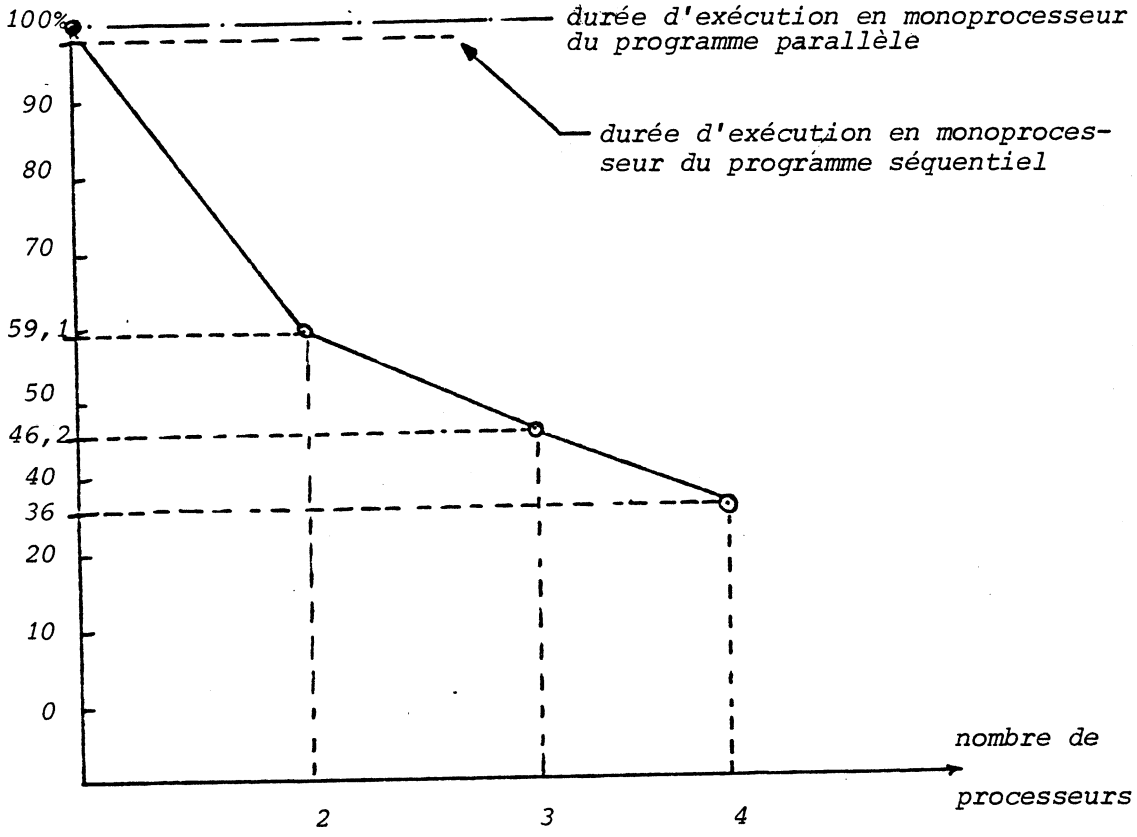
4. L'overhead résultant de l'exécution d'un programme parallèle par plusieurs processeurs est attribué aux effets suivants:

   a. Overhead dû à l'arbitrage du bus

   b. Overhead dû à l'exécution des primitives du S.E.P.

   c. Overhead dû à l'attente d'entrée en section critique (Test et Set)

Dans un programme entièrement parallèle comme celui-ci on pourrait évaluer la durée théorique de l'exécution en multiprocesseur . Par suite on pourrait calculer l'overhead global. Il restera donc à détailler cet overhead selon les 3 rubriques ci-dessus. L'image étant découpée en 8 segments, on notera la durée Tde traitement d'un segment de l'image par 1 processeur d'instruction.



RESULTATS EN POURCENTAGE DE LA DUREE D'EXECUTION DU PROGRAMME PARALLELE EN FONCTION DU NOMBRE DE PROCESSEURS

Schéma 3.4.

La durée du traitement des 8 segments par un seul processeur a été mesurée, et vaut 129 ms. $T = \frac{129}{8} = 16{,}125$ millisecondes.

Le tableau suivant donne l'overhead global de l'exécution du programme parralèle avec 1, 2, 3 et 4 processeurs.

Nombre de processeurs	Durée d'exécution théorique (sans overhead)	Durée d'exécution mesurée	Overhead global	% overhead sur la durée d'exécution
1	8 T = 129 ms	132	3 ms.	2,3 %
2	4 T = 64.5 ms.	78	13,5	17,3 %
3	3 T = 48.75	61	12,25	20,1 %
4	2 T = 32.25	48	15,75	32,8 %

Puisqu'on connaît les durées d'exécution des différentes primitives du S.E.P on pourrait calculer l'overhead dû à l'exécution de ces primitives en reconstituant les traces d'exécution des différents processeurs comme suit:

Nombre de processeurs	Primitives exécutées par 1 processeur en //	Durée d'exécution (Max) des primitives du S.E.P
1	1 FØRK + 8 JOIN	0.469 + 8 x 0.330 = 3.11 ms.
2	1 FORK + 4 JOIN	0.469 + 4 x 0.330 = 1.79 ms.
3	1 FORK + 3 JOIN	0.469 + 3 x 0.330 = 1.46 ms.
4	1 FORK + 2 JOIN	0.469 + 2 x 0.330 = 1.13 ms.

On remarque de ces valeurs que l'overhead dû uniquement à l'exécution des primitives du S.E.P est très faible même en multiprocesseur .

La carte d'évaluation spécialisée nous permettra de mesurer avec plus de précision l'overhead de l'arbitrage du bus et celui du Test & Set. Il aurait été possible d'employer une méthode pour essayer d'évaluer l'overhead dû à l'attente d'entrée en section critique. Cette méthode consiste à incrémenter un compteur logiciel à chaque demande d'accès exclusif non satisfait. A la fin de l'exécution du programme la valeur du compteur peut être convertie en durée d'attente comme indiqué en début de ce chapitre. Cela n'a pas été fait, parce que cette méthode a l'inconvénient de fausser la période de surveillance de la section critique en rajoutant les instructions d'incrémentation du compteur, et donc de fournir des résultats inexacts impossibles à interpréter de façon rigoureuse.

3.2. *EVALUATION DU PARALLELISME CONJOINT*

Le programme précédent de superposition de deux images est réalisé par une boucle qui sera exécutée en parallèle.

Chaque séquence de la boucle traitera un segment de chaque image. Le corps de cette boucle sera exécuté 8 fois. On fera donc appel à la primitive NFORK. On utilisera à l'intérieur de la boucle l'index de la branche pour se pointer en début du segment à traiter de chaque image. Le processeur qui terminera la dernière branche de la boucle affichera comme dans le programme précédent une valeur dans ses diodes de contrôle des activités des processeurs. Les autres retourneront à leur boucle d'attente.

Avec cette version du programme de superposition d'images, on évaluera ainsi les programmes utilisant le parallélisme conjoint.

Primitives utilisées: NFORK et NJOIN

```
        PROGRAMME   BCLIMG
Début
        tableau entier IMG1 (4096), IMG2 (4096);
        Entier  Index, I;
+++     NFORK    (8, BCLSQ, AFFICH)
BCLSQ:  I : = (Index-1) x 512;
        pour J : = 1 jusqu'à 512 faire
             IMG2 (I + J) : = IMG1 (I + J) + IMG2 (I + J);
+++     NJOIN
AFFICH: (AFFICHAGE PERMANENT)
Fin
```

METHODES D'EVALUATION

1. Comme dans le cas précédent, ce programme sera exécuté sur le système de développement en monoprocesseur et sur la maquette avec un seul processeur.

2. De même, le programme sera exécuté sur la maquette avec 1,2,3 et 4 microprocesseurs d'instructions. On déduira ainsi la relation entre le temps d'exécution et le nombre de processeurs.

3. On évaluera l'overhead dû à l'exécution des primitives du parallélisme conjoint NFORK et NJOIN. Le programme donc sera traduit en une boucle normale sans parallélisme, il sera exécuté sur un seul processeur de la maquette.

4. On fera ensuite la comparaison des résultats d'exécution du programme de superposition des deux images dans les deux cas de parallélisme disjoint et conjoint.

RESULTATS

- Durée d'exécution de la primitive NFORK = 366 µ sec

- Durée d'exécution (moyenne) de la primitive NJOIN..... = 200 µ sec

Nombre de Processeurs	Durée d'Exécution (ms)	Remarques
1 uP	132	Sur Prototype
2 uP	79	"
3 uP	60	"
4 uP	46	"
1 uP	132	Système de Développement
1 uP	130	Séqentiellemment

Taille mémoire nécessaire pour le programme parallèle: 67 octets

Taille mémoire nécessaire pour le programme séquentiel: 54 octets

CONSTATATIONS ET COMMENTAIRES

1. On remarque que l'overhead dû aux primitives NFØRK-NJOIN est très légèrement moins important que celui du FORK & JOIN. Cette différence résulte du fait que dans le parallélisme conjoint on exécute la même séquence; par contre, dans le parallélisme disjoint, on décide selon les compteurs de parallélisme quelle séquence il faut exécuter. Il est à rappeler aussi que la précision des moyens de mesure utilisés (analyseur temps réel & boucle de comptage) est de l'ordre de $\pm$ 1 MSEC pour les durées globales d'exécution.

2. Le seul intérêt significatif de l'utilisation du parallélisme conjoint (si les séquences parallèles sont semblables) est que l'écriture des programmes est plus compacte et par conséquent occupe moins de place en mémoire utilisateur.

3. Le reste des résultats obtenus confirme ceux de l'expérimentation précédente.

4. Il est à noter que, dans le cas d'absence des mémoires locales (le cas de notre expérimentation), il faut manipuler le registre C' qui contient l'index de la branche parallèle avec prudence. Durant l'accès à ces registres par l'utilisateur, il faut éviter toute interruption. Pour cela, on propose deux solutions possibles:

   a. A l'attribution d'une branche à un processeur, on lui transmet, par les primitives du S.E.P, l'index dans un registre interne normal en vue de l'utiliser à l'intérieur de la branche.

   b. Soit l'utiliser en section critique par demande d'accès exclusif de l'utilisateur (cas retenu dans ce programme).

3.3. *EVALUATION DES PROGRAMMES UTILISANT DES PRIMITIVES DE PARALLELISME IMBRIQUEES.*

L'objectif de ce groupe de test est d'évaluer les performances du système dans le cas des processus parallèles imbriqués avec tous les problèmes d'allocations et de restitutions des blocs de contrôle. On expliquera le comportement du système dans le cas d'exécutions de processus sous forme arborescentes en se servant des traces d'informations contenues dans les blocs de contrôle avant et après l'exécution des programmes.

Soit le programme de tri en utilisant l'algorithme de Quicksort. La méthode consiste à découper un tableau TAB (X) § x = 1:n § en deux sous-tableaux. L'un contient les valeurs inférieures à un repère INDEX, l'autre contient les valeurs supérieures à ce même repère. L'algorithme en langage de haut niveau parallèle (langage pastoral) est donné à la page suivante.

La procédure SEGMENT rend la valeur INDEX comme résultat.

Chaque sous-tableau sera traité par un processeur élémentaire. Ceci est réalisé à l'aide des primitives FØRK-JOIN. Les deux séquences (séquence 1 et séquence 2) de la procédure TRI seront exécutées en parallèle (MAS 78).

Ce programme étant récursif, il était beaucoup plus facile de l'implémenter en langage de haut niveau qui traduit la récursivité tel que le Pascal (le Pascal séquentiel pourrait être utilisé dans notre système évidemment en utilisant les primitives du S.E.P comme appel aux sous-programmes assembleurs). Ce programme a été écrit en assembleur Z 80 (8002 de TEKTRONIX); la pile a été utilisée pour résoudre les problèmes dûs à la récursivité des appels de la procédure TRI.

Primitives utilisées: FORK & JOIN (imbriquées)

```
vec ( 1:n) ent T :
procédure SEGMENT ( val ent INF, SUP ; ent INDEX) ;
|  ent I, J, TEMOIN :
|  procédure ECHANGE ( val ent X,Y) ;
|  |  ent Z ;
|  |  Z := T( X) ; T( X) := T( Y) ; T( Y) := Z ;
|  finst :
|  I := INF ; J := SUP-1 ; TEMOIN := T( SUP) ;
|  tant que I≠J faire
|  ( si T( I) <= TEMOIN alors I := I + 1
|  | sinon
|  |    si T( J) > TEMOIN alors J := J - 1
|  |    sinon ( ECHANGE ( I, J) ; I := I + 1)
|  ) ;
|  si T( I) <= TEMOIN alors INDEX := I + 1 sinon INDEX := I ;
|  ECHANGE ( INDEX, SUP) ;
finst :
procédure TRI ( val ent INF, SUP) ;
|  ent ASAPLACE ;
|  si INF < SUP alors
|  ( SEGMENT ( INF, SUP, ASAPLACE) ;
|  |  P. ( TRI ( INF, ASAPLACE - 1) ;  %seg 1%
|  |  |     TRI ( ASAPLACE + 1, SUP)    %seg 2%
|  |  |     )
|  ) fsi
finst ;
TRI ( 1, N) ;
```

METHODES D'EVALUATION

Le premier processeur qui exécute ce programme sauvegarde dans son contexte la séquence à exécuter à la fin de la procédure de TRI récursive. Cette séquence qui consiste à afficher une valeur sur les diodes de contrôle d'activité des processeurs sera exécutée par le processeur qui termine le tri.

Le programme est exécuté sur un ensemble de données de 7, 21 et 32 éléments.

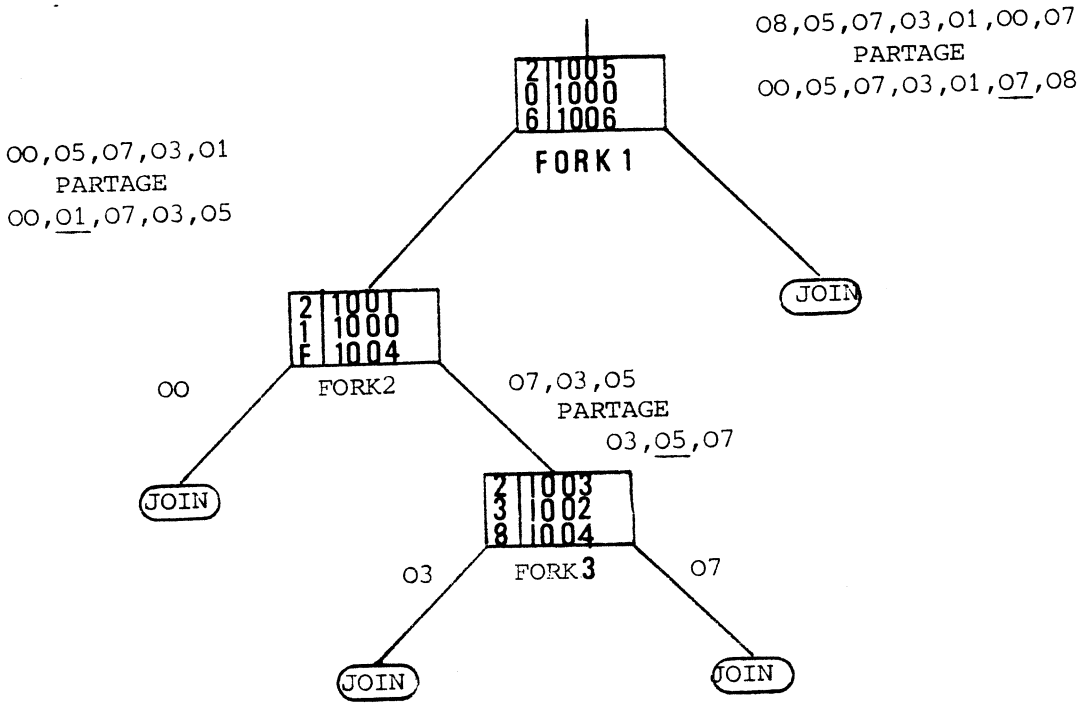
En ce qui suit on donne les résultats d'exécutions et les schémas décrivant l'éclatement des processus grâce aux traces d'informations dans les blocs de contrôle dans 2 cas.

RESULTATS

Test n° 1 : Tri de 7 éléments

Nombre de Processeurs	Durée d'Exécution (ms)	Remarques
1 uP	5	Les mesures de temps sont effectuées en
2 uP	5	nombre entier de tour (boucle de comptage);
3 uP	5	un tour vaut environ 1 ms.
4 uP	7	

71



TRI DE 7 ELEMENTS

Schéma 3.6.

Test n° 2: Tri de 21 éléments

Nombre de Processeurs	Durée d'Exécution (ms)	Remarques
1 uP	24	
2 uP	20	
3 uP	18	
4 uP	18	

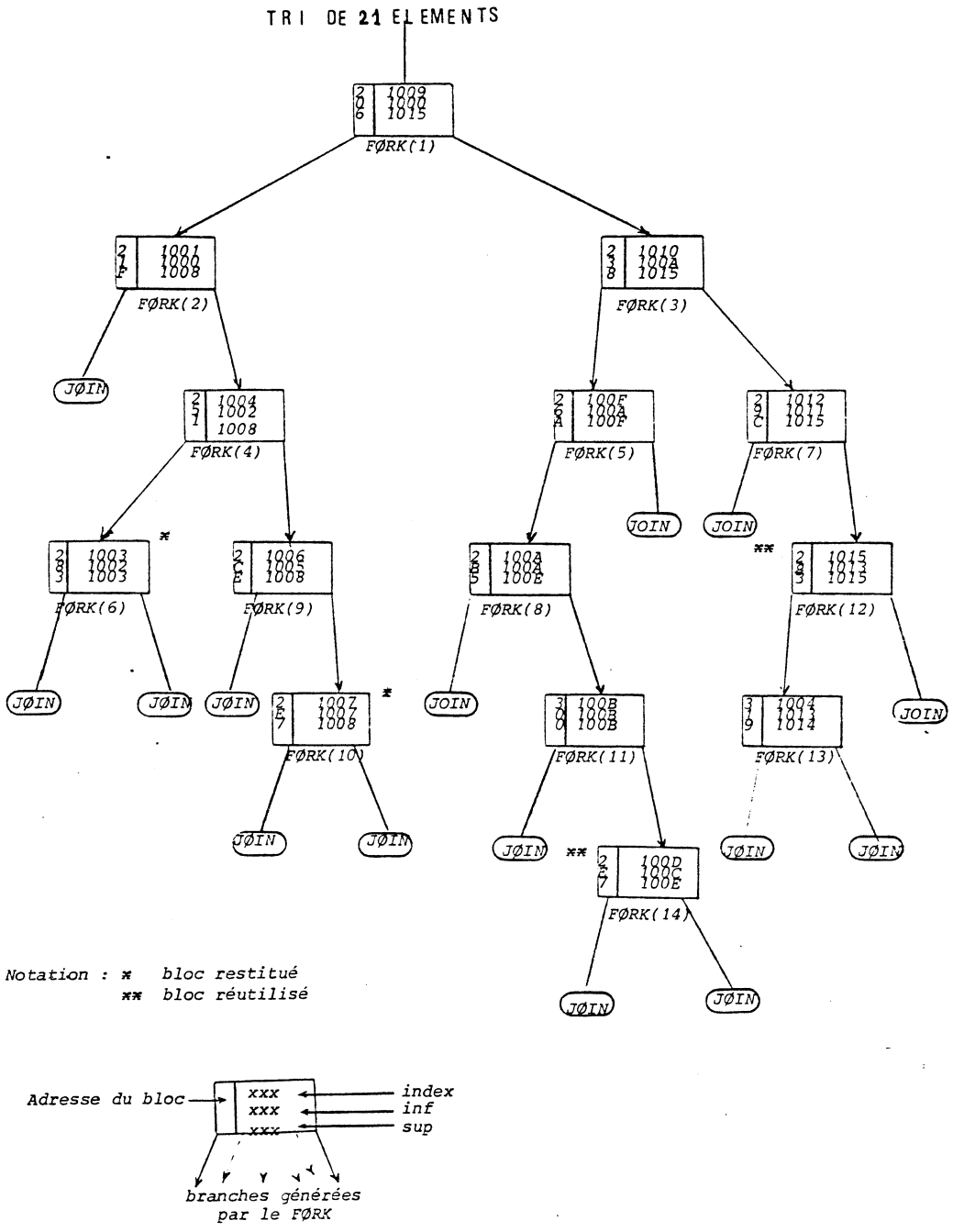

TRI DE 21 ELEMENTS
FØRK(1)
FØRK(2)
FØRK(3)
FØRK(4)
FØRK(5)
FØRK(7)
FØRK(6)
FØRK(9)
FØRK(8)
FØRK(12)
FØRK(10)
FØRK(11)
FØRK(13)
FØRK(14)
JØIN
JOIN
Notation : * bloc restitué
** bloc réutilisé
Adresse du bloc
xxx
xxx
xxx
index
inf
sup
branches générées
par le FØRK

Schéma 3.7.

Test n° 3: Tri de 32 éléments

Nombre de Processeurs	Durée d'Exécution (ms)	Remarques
1 uP	42	
2 uP	29	
3 uP	27	
4 uP	27	

CONSTATATIONS ET COMMENTAIRES

Le temps d'exécution de ce programme pourrait être accéléré si on limite l'éclatement du processus à une taille minimum de sous-tableau à trier. Ceci évite l'overhead dû au parallélisme pour trier des sous-tableaux ne dépassant pas trois éléments:

```
Procédure TRI  (val ent INF, SUP, TMIN);
|
|   ent  ASAPLACE;
|
|   Si      SUP - INF ≥   TMIN
|
|       Alors
|       (Segment (INF, SUP, ASAPLACE);
|           |
|           P. (TRI (INF, ASAPLACE-1, TMIN);
|               |
|               TRI (ASAPLACE + 1, SUP, TMIN)
|               )
|       Sinon
|       (tant que INF < SUP  Faire
|           |
|           (pp := INF ;
|               |
|               pour i := INF+ 1 jusqu'à SUP Faire
|                   |
|                   (Si  T (pp) ≤  T (i)  Alors  pp := I;
|                   )
|                   T (INF) := T (pp);
|                   |
|                   INF := INF + 1 ;
|               )
|           )
|   Fsi
|
Finst

TRI (1, N, TMIN)
```

D'après les résultats de notre première expérience avec 7 éléments on constate le phénomène suivant:

Il y a très peu d'éléments à trier et beaucoup plus de processeurs qu'il n'en faut pour exécuter très peu de branches paralléles; celles-ci sont très courtes par rapport à l'overhead de recherche de travail pour les processeurs libres. Cette recherche s'exécute en section critique et bloque ainsi pour un certain temps les processeurs qui ont déjà du travail et veulent faire des JOIN ou des nouveaux FORK.

Un ralentissement peut donc apparaître quand les branches parallèles sont trop courtes. De ce phénomène on déduit que le parallélisme n'est pas avantageux quand les branches parallèles sont trop courtes.

Le schéma 3.9. confirme cette observation car on remarque qu'en augmentant le nombre de processeurs d'instructions les performances s'améliorent lorsque la taille du tableau à trier est assez importante. Par contre, on aurait pu réduire cet overhead en rendant l'accès des processeurs libres à la section critique moins prioritaire que celui des processeurs qui ont déjà du travail.

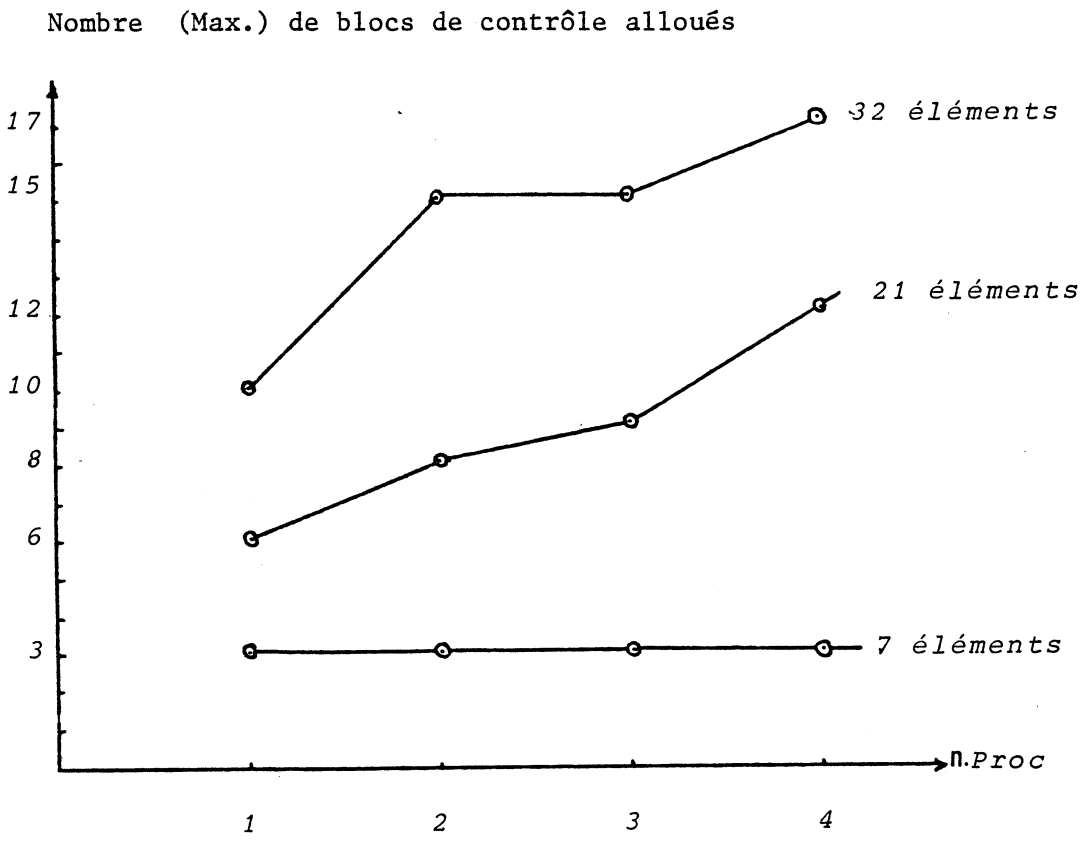
Nombre de blocs de contrôle utilisés:

Le nombre de blocs de contrôle alloués aux processus parallèles dépend du nombre de processeurs et de l'organisation du programme.

Malgré le fait que des blocs restitués par certains processeurs sont alloués de nouveau à d'autres processeurs (schéma 3.7.), le nombre des blocs utilisés augmente avec le nombre de processeurs.

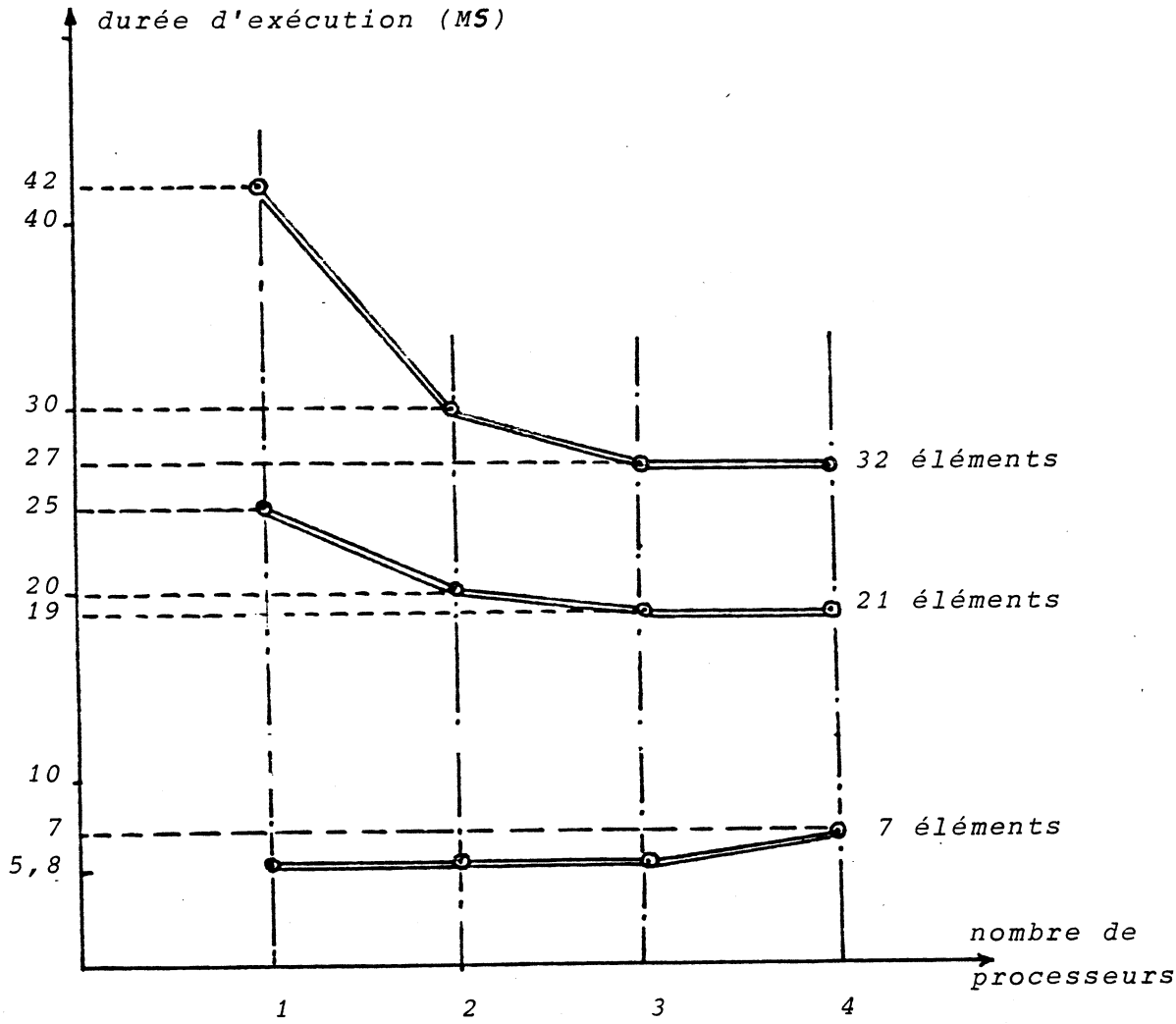
Il est à noter que le nombre des blocs utilisés dans l'exécution d'un même programme n'est pas obligatoirement le même dans deux expérimentations distinctes.

De même les programmes récursifs comme ce programme de tri "Quick sort" utilisent un grand nombre de blocs de contrôle. Le schéma 3.8. donne le nombre des blocs de contrôle utilisés en fonction du degré de parallélisme et du nombre de processeurs utilisés.



BLOCS DE CONTROLE
ALLOUES (MAX)

Schéma 3.8.



DUREE D'EXECUTION DU PROGRAMME
DE TRI QUICKSORT

Schéma 3.9.

3.4. *EVALUATION DE L'EXECUTION PARALLELE DES INSTRUCTIONS "DO" DU FORTRAN*

Un autre domaine où le traitement parallèle peut être particulièrement avantageux par rapport au traitement séquentiel classique est ce domaine de traitement de l'image.

Pour être plus proche des problèmes soulevés dans l'application des algorithmes parallèles dans le traitement de l'image, on a repris certaines séquences des programmes développés au sein de l'équipe travaillant dans le cadre du projet SAMBA (Ste THOMSON-TITN & Laboratoire de Microscopie Quantitave à l'U.S.M.G.), ces séquences écrites en langage Fortran (Séquentiel) ont été réecrites en Assembleur Z 80 pour être exécutés sur notre machine.

Comme on l'a déjà indiqué dans le chapitre 2, la traduction automatique des boucles telles que les boucles "DO... END DO" dans certains langages de haut niveau, ont fait l'objet de plusieurs travaux de recherches. Puisque, à l'origine, ce programme était écrit en Fortran, on développera, dans ce qui suit, quelques idées concernant l'exécution en parallèle des boucles DO du langage FORTRAN.

Considérons la séquence des boucles DO imbriquées comme suit:

```
DO  L1  I1 = 1, Q1
DO  L1  I2 = 1,Q2
.................
DO  L1  Ij = 1,Qj

Xm (Ī) = F (Ī)
X2 (Ī) = F (Ī)
.................
L1 Xm (Ī) = F (Ī)
```

où $\bar{I}$ est le vesteur représentant l'ensemble des indices (I1, I2..., Ij) des boucles.

Cette boucle est constituée d'une série d'instructions d'affectations. Par instruction d'affectation, on désigne une instruction ayant la forme (X = E) où "E" est une expression arithmétique ou logique.

Dans ce type d'instructions on définit "X" comme une variable de Sortie et toutes les variables de "E" comme variables d'Entrées. La boucle est dite linéaire si toutes les instructions qui la composent sont linéaires c'est-à-dire ayant la forme:

C + A1.X1 + A2.X2 + ... + AK.XK
(A,C étant des constantes et X1, X2, ...,XK sont des variables).

SHYH-CHING et KUCK ont formulé certaines méthodes pour exécuter en parallèle les instructions à l'intérieur de ce type de boucles. Leur méthode consistait à déterminer les dépendances entre ces instructions, puis à éliminer ces dépendances afin de pouvoir réussir leur exécution en parallèle. (KU 72-79)

Dépendances:

Considérons deux instructions X1, Xm de la boucle citée dans le paragraphe précédent.

On peut dire que X1 dépend de Xm (notée Xm δ X1) s'il existe deux valeurs des vecteurs $\bar{i}$ et $\bar{j}$ élément du vecteur $\bar{I}$ tels que:

L'instruction Xm ($\bar{i}$) est située avant l'instruction X1 ($\bar{j}$) dans l'exécution en série de la boucle et en plus on vérifie une des conditions suivantes:

a. La variable de Sortie de Xm ($\bar{i}$) est la même que celle de X1 ($\bar{j}$).

ou

b. La variable de Sortie de Xm ($\bar{i}$) est une des variables d'Entrée de X1 ($\bar{j}$).

ou

c. La variable de Sortie de X1 ($\bar{j}$) est une des variables d'Entrée de Xm ($\bar{i}$).

Selon la deuxième condition on définit ainsi trous types de dépendances:

a. Dépendance tout court de X1 sur Xm, notée Xm δ X1.

b. Dépendance en Sortie de X1 et de Xm, notée Xm δ^{o} X1.

c. Antidépendance de X1 sur Xm, notée Xm $\bar{\delta}$ X1.

Les méthodes proposées par (KU 79) sont utilisables pour éliminer les dépendances des instructions. Ces instructions peuvent être alors exécutées en parallèle sur plusieurs processeurs. L'exemple suivant illustre cette approche:

```
          DO I2   i = 0,n

I1        A(i) = B(i) x C(i)
I2        D(i) = A(i) + A(i) X A(i-1)
```

En éliminant la dépendance de I2 sur I1 (I1 δ I2) on obtient deux instructions exécutables simultanément:

```
          DO I2   i = 0,n
I1        A(i) = B(i) x C(i)
I2        D(i) = B(i) x C(i) + B(i) x C(i) x B(i-1) x C(i-1)
```

Sérialisation des boucles "DO" du Fortran

Considérons ce cas très fréquent dans les applications bidimensionnelles et plus précisément dans le traitement de l'image. En général, une image est composée par un ensemble de points. Un point est défini par 2 indices: un index qui désigne la ligne et un autre qui désigne la colonne. L'ensemble de ces deux index situe le point au sein d'une image. Ces déplacements sont pris par rapport à un point de référence.

Dans la plupart des cas les différents points de l'image sont consultés et à partir d'un ensemble de conditions définies par les utilisateurs on génère une autre image.

Pour illustrer le mécanisme décrit ci-dessus, considérons la séquence des boucles DO suivante qui construit une image A (i,j) à partir de deux vecteurs de données C (i) et B (j) (on considère que C (i) = A (i,o)) .

```
          DO   IX    i = 1,10
          DO   IX    j = 1,10

Ix        A(i,j) = A(i,j-1) V B(j)

          CONTINUE
```

Pour traduire cette boucle en séquences exécutables simultanément par plusieurs processeurs on procèdera selon le degré de parallélisme voulu comme suit:

A. En sérialisant cette boucle en 100 instructions parallèlement exécutables. Si l'on dispose de 100 processeurs, chaque processeur calculera la valeur d'un point dans la nouvelle image selon les instructions suivantes:

```
Processeur 1 .... A(1,1) = A(1,0)  V  B(1)
Processeur 2 .... A(1,2) = A(1,0)  V  B(1)  V  B (2)
.................................................
Processeur 100. A(10,10) = A(10,0) V.. V B(10)
```

Cette méthode transforme la boucle en instructions entièrement indépendantes mais comme on peut le remarquer elle multiplie le nombre des opérations effectué par l'ensemble des processeurs.

B. En affectant à chaque processeur le calcul des valeurs des points d'une ligne de l'image selon le découpage suivant:

```
Processeur 1 .... A(1,1) = A(1,0)  V  B(1)
                  A(1,2) = A(1,1)  V  B(2)
                  ........................
                  A(1,10) = A(1,9) V  B(10)
Processeur 2 .... A(2,1)  = A(2,0) V  B(1)
                  A(2,2)  = A(2,1) V  B(2)
                  ........................
                  A(2,10) = A(2,9) V  B(9)
.........................................
Processeur 10... A(10,1) = A(10,0) V B(1)
                  ........................
                  A(10,10) = A(10,9) V B(9)
```

Revenant à notre évaluation, il s'agit dans ce programme de délimiter les frontières d'une image. La méthode utilisée consiste à contracter l'image puis la dilater:

- Contracter une image (Sous-programme SHRINK) se traduit par une mise à zéro de tous les points qui ont un de leurs voisins les plus proches qui est égal à zéro.

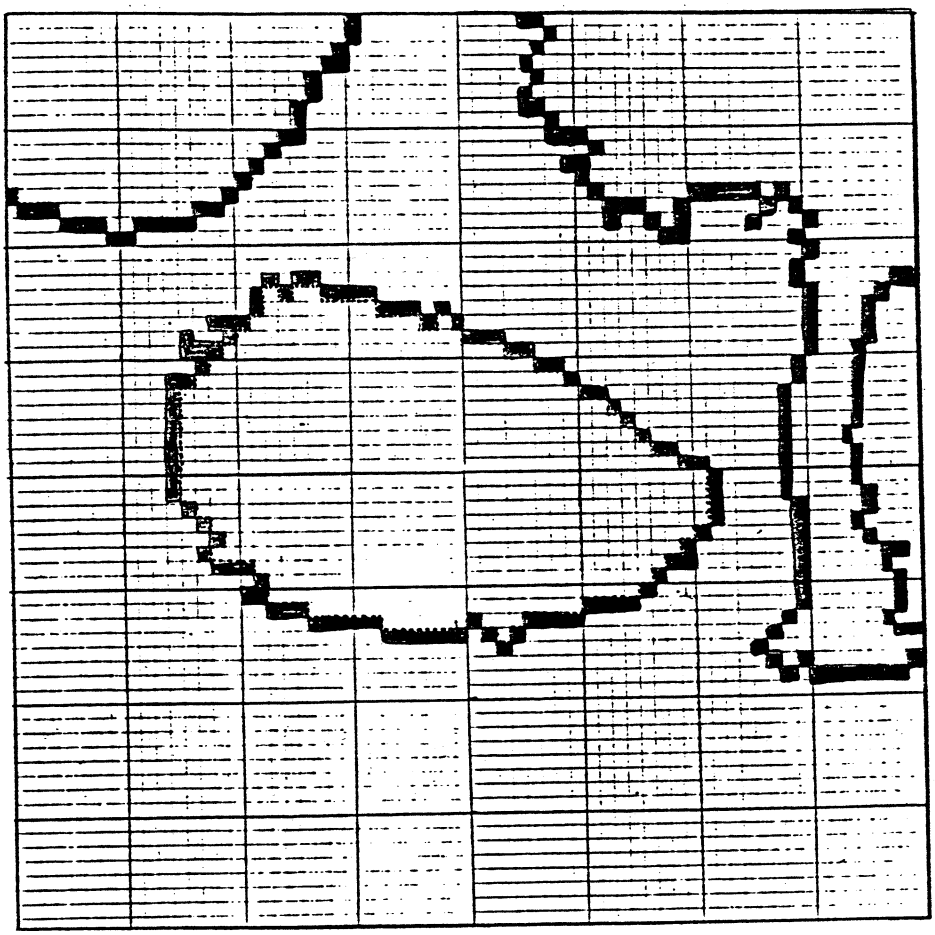
- Dilater une image (Sous-programme DILATE) est en effet l'inverse de SHRINK où l'on met à 1 les points qui ont un de leurs voisins les plus proches à 1.

Le parallélisme exploité est de la même nature que celui décrit ci-dessus. Les dépendances ont été tout naturellement éliminées par l'utilisation d'un tableau "Cible" IMAGE 1 différent du tableau d'origine IMAGE 2. (On remarquera une fois encore que l'exploitation du parallélisme nécessite de multiplier les mémoires utilisées.)

L'algorithme de la procédure SHRINK est donnée en pseudo. PASCAL, en employant l'instruction *avec* du langage PASTORAL pour désigner un "Pour parallèle".

```
PROCEDURE SHRINK ( IMG1, IMG2)

tableau IMAG1 ( 4096), IMAG2 ( 4093) entier ; (*image résultat, origine)

entier   TS, NC, NL, DS, DL, DP (*taille segment, nombre de colonnes,
                                  nombre de lignes, déplacement seg.,
                                  déplacement lignes, dépl. point )

début
|   TS := 512 ; NC:= 64 ; NL:= 64 ;
BCLP :  avec  i:= 1 jusqu'à 8 faire     (*boucle parallèle de 8 branches)
|       début
|       |   DS := (i - 1) *TS ;
|       |   pour  j:= 1 jusqu'à 8  faire
|       |   début
|       |   |   DL :=  (j - 1) * NC ;
|       |   |   pour k:= 2  jusqu'à NL - 1 faire
|       |   |   début
|       |   |   |   DP := DS + DL + k ;
|       |   |   |   IMAG1 (DP) := IMAG2 (DP + 1) et IMAG2 (DP - 1)
|       |   |   |                 et IMAG2 (DP + 64) et IMAG2 (DP - 64)
|       |   |   |                 et IMAG2 (DP) ;
|       |   |   fin ;
|       |   fin ;
|       fin ;
|
|
```

```
   (* exécution séquentielle des bords de l'image, le traitement sera
      séquentiel)
BCLS : pour i:= 2 jusqu'à 63 faire
       début
           DP := 1 + (i - 1) * NC ;
       IMAG1 (DP):= IMAG2 (DP + 1) et IMAG2 (DP + 64)
                    et IMAG2 (DP - 64) et IMAG2 (DP) ;
           DP := i *NL ;
       IMAG1 (DP):= IMAG2 (DP + 64) et IMAG2 (DP - 64)
                    et IMAG2 (DP - 1) et IMAG2 (DP) ;
           DP := i ;
       IMAG1 (DP):= IMAG2 (DP - 1) et IMAG2 (DP + 1)
                    et IMAG2 (DP + 64) et IMAG2 (DP);
           DP :=  i + NC (NL - 1) ;
       IMAG1 (DP):= IMAG2 (DP - 1) et IMAG2 (DP + 1)
                    et IMAG2 (DP - 64) et IMAG2 (DP) ;

       fin;
       DP := 1 ;
       IMAG1 (DP):= IMAG2 (DP) et IMAG2 (DP + 1) et IMAG2 (DP + 64) ;
       DP := NC ;
       IMAG1 (DP):= IMAG2 (DP) et IMAG2 (DP - 1) et IMAG2 (DP + 64) ;
       DP := NC *(NL - 1) + 1 ;
       IMAG1 (DP):= IMAG2 (DP) et IMAG2 (DP - 64) et IMAG2 (DP + 1) ;
       DP := NC *NL ;
       IMAG1 (DP):= IMAG2 (DP) et IMAG2 (DP - 1) et IMAG2 (DP - 64) ;

fin.
```



CONTRACTION DE L'IMAGE

Schéma 3.10.

METHODES D'EVALUATION

Le sous-programme SHRINK est traduit par une boucle de parallélisme conjoint utilisant les primitives NFORK-NJOIN. Les points à l'intérieur de l'image sont traités par cette boucle. L'image est découpée en 8 segments horizontaux. L'index est utilisé à l'entrée dans les mini-processus pour pointer sur le segment à traiter. Le processeur qui terminera le dernier segment se chargera du traitement des points sur les côtés de l'image et enfin il affichera une valeur sur ses diodes de contrôle des activités.

En ce qui suit, on donne les résultats de l'expérimentation.

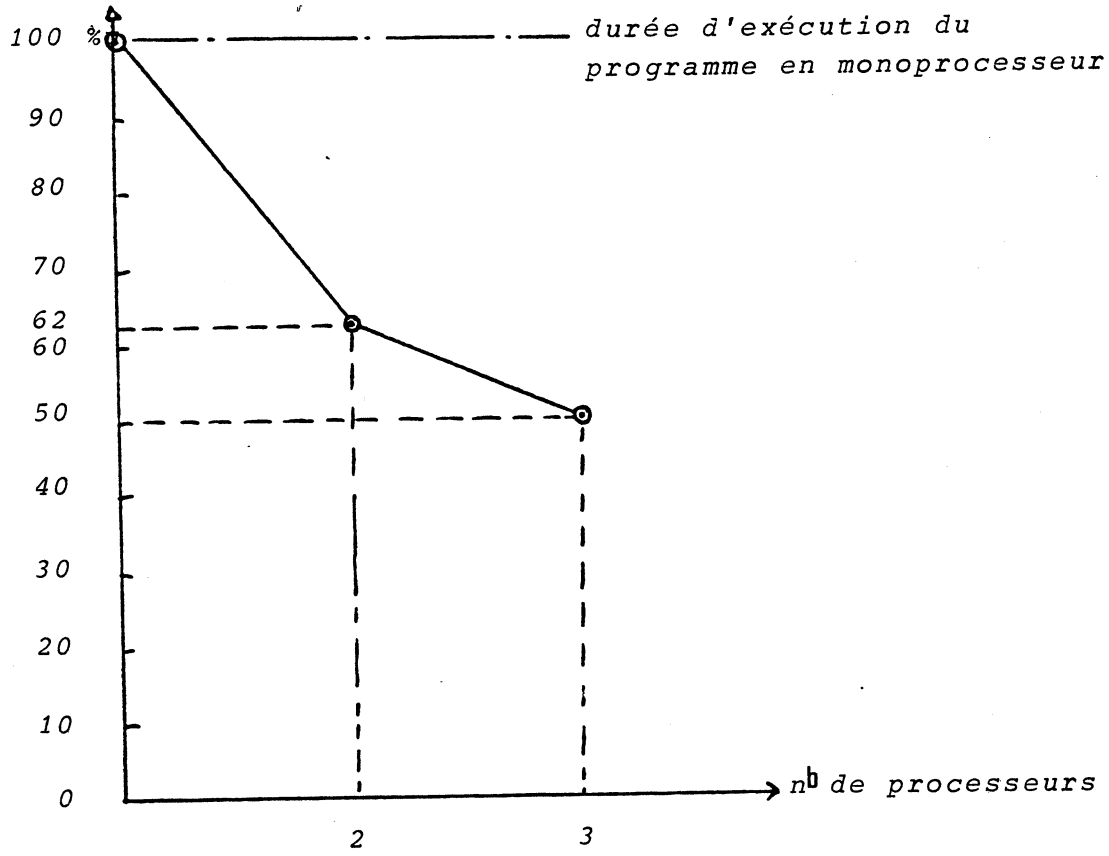
RESULTATS

Nombre de Processeurs	Durée d'Exécution (ms)	Remarques
1 uP	221	
2 uP	137	
3 uP	110	

CONSTATATIONS ET COMMENTAIRES

La durée d'exécution du programme, en utilisant deux processeurs, est réduite à 62 % de la durée globale du programme en monoprocesseur.

Elle passe à 50 % avec trois processeurs (schéma 3.11.). On constate que le taux de réduction est plus faible que celui des tests précédents. Ceci résulte du fait que le traitement des bords de l'image est séquentiel. Notre évaluation comportait tout le programme y compris cette séquence.



DUREE D'EXECUTION DU PROGRAMME SHRINK

Schéma 3.11.

3.5. *EVALUATION DU PARALLELISME "OU"*

Un des cas les plus simples d'utilisation du parallélisme "OU" est celui de la recherche d'un élément dans un tableau.

Soit un tableau T (1: n) éléments. Ce tableau sera découpé en N sous-tableaux de taille n/N chacun. Le processus de recherche d'un élément dans un tableau sera donc éclatéenNminiprocessus de recherche en parallèle dans les sous-tableaux. Dès qu'un processeur réussit à trouver l'élément, il demandera par l'appel à la primitive FSAIS du système d'exploitation parallèle, d'arrêter les autres miniprocessus et il affichera sur ses diodes de contrôle des activités des processeurs une valeur prédéfinie. Les autres processeurs, comme dans les cas précédents, retourneront à leur boucle d'attente.

Primitives utilisées: FORK, JOIN et FSAIS

METHODES D'EVALUATION

Le tableau est de 8 K octets de taille. Il est découpé en 4 sous-tableaux ou segments. L'emplacement de l'élément recherché sera choisi de telle sorte que les différents cas puissent se présenter.

3.5.1 1re évaluation

La valeur recherchée est à l'adresse 2 000 H (en début du 3e segment). Cette première évaluation très simple permet déjà un certain nombre de constatations. On trouvera plus loin une évaluation plus complète, c'est-à-dire donnant des résultats indépendants de l'emplacement de l'élément dans le tableau.

RESULTATS

Nombre de Processeurs	Durée d'Exécution (ms)	Remarques
1 uP	73	
2 uP	40	
3 uP	2	CAS TRES FAVORABLE

CONSTATATIONS ET COMMENTAIRES

1. L'élément recherché se trouve en début du 3e segment. Avec un seul processeur on doit parcourir entièrement les deux premiers segments pour trouver l'élément.

   Dans le cas de deux processeurs, on parcourt en parallèle les deux premiers segments en même temps. Ensuite on trouve l'élément en début du troisième segment:

   $$\text{Durée d'exécution du programme} \simeq \frac{\text{durée avec 1 processeur}}{2} + T$$

   T étant l'overhead dû à l'exécution des primitives du S.E.P et à l'attente d'entrée en section critique pour exécuter le "FSAIS".

   En utilisant trois processeurs, chaque processeur se chargera de la recherche de l'élément dans un segment. Le troisième processeur le trouve en début de son segment. Il positionne l'indicateur correspondant, demande l'arrêt des autres processeurs et termine tout seul le programme sans attendre la libération des autres processeurs. Ce qui explique le temps très court d'exécution du programme dans ce cas très favorable.

2. Dans certains cas on a observé une pénalisation trop importante dans l'exécution du programme en multiprocesseur. Cette pénalisation vient du fait de l'exécution de la primitive "FSAIS" en section critique par le mécanisme du Test & Set câblé. Ce mécanisme permet l'accès exclusif à cette zone, mais d'une manière aléatoire.

   Considérons le cas d'exécution du programme par trois processeurs. D'après le schéma 3.12 on constate une situation classique de "privation". Pour remédier à celà, plusieurs solutions sont possibles:

   a. Modifier le circuit du Test & Set pour réaliser une allocation à priorité circulaire de la section critique.

   b. Réaliser un programme d'allocation du verrou prioritaire qui inhibe le Test & Set normal et qui soit réservé uniquement aux processeurs désirant exécuter des FSAIS.

   c. Avoir un grand nombre de verrous, chacun protégeant une seule ressource. Faire des Test & Set sur ces différents verrous. La probabilité de conflit (et donc de "privation") est alors très faible, mais il y a un risque de blocage ("dead-lock") qu'il faut éliminer.

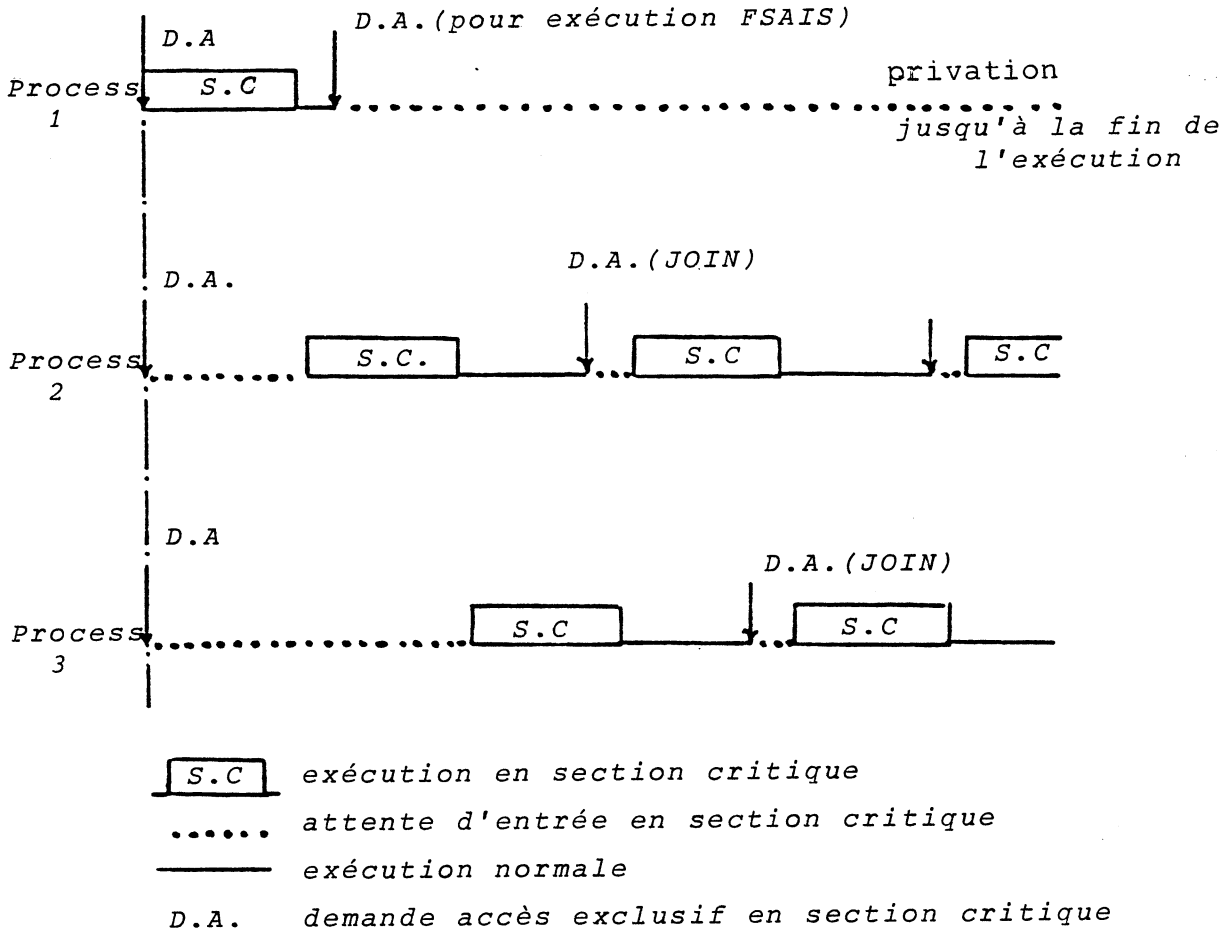


Schéma 3.12.

PRIVATION DANS L'EXECUTION DU FSAIS

La première solution suffit pour éliminer la privation, mais elle nécessite une retouche du matériel; nous ne l'avons pas implémentée.

La seconde solution permet de régler le problème du FSAIS avec une bonne efficacité puisqu'il sera toujours servi en priorité. Elle présente aussi d'autres avantages: on a vu à la page 75 un intérêt de ce Test & Set prioritaire. On la présente dans le paragraphe suivant.

3. Réalisation de l'accès prioritaire à la section critique. La méthode consiste à réaliser un mécanisme qui permet de signaler aux processeurs désirant accéder à la section critique, les demandes d'accès prioritaires d'autres processeurs pour l'exécution des FSAIS. Il suffit donc d'ajouter un indicateur d'accès prioritaire utilisé selon les séquences suivantes:

```
%   Demande d'accès normal en section critique  %
X:  Si indicateur positionné
        Alors aller à X
        Sinon demande d'accès exclusif (Test & Set)
            Si  ECHEC
                Alors aller à X
            Fsi
    Fsi
%   Demande d'accès prioritaire  %
B:  positionner l'indicateur
    demande d'accès exclusif (Test & Set)
    Si  ECHEC   Alors   aller à B
                Sinon   remise à zéro de l'indicateur
   Fsi
```

Il est à noter que ce mécanisme ne résoud pas forcément le problème de la privation entre des FSAIS; mais il faut remarquer que l'exécution d'une FSAIS est rare puisqu'elle correspond à l'extinction complète d'un arbre de mini-processus parallèles menant à la désactivation des processus: un cas de privation entre plusieurs FSAIS ne saurait donc se prolonger.

3.5.2. 2e évaluation

La position de l'élément recherché varie pour chaque exécution de manière qu'on parcoure les 4 segements du tableau. 32 mesures ont été prises, chacune avec 1, 2, 3 et 4 processeurs (schéma 3.13.).

Chaque unité de mesure correspond à 1,17 ms. (boucle de comptage) et les résultats sont donnés à une unité près.

Cette expérience nous permet d'évaluer le parallélisme "OU" dans la recherche d'un élément dans un tableau quelque soit son emplacement.

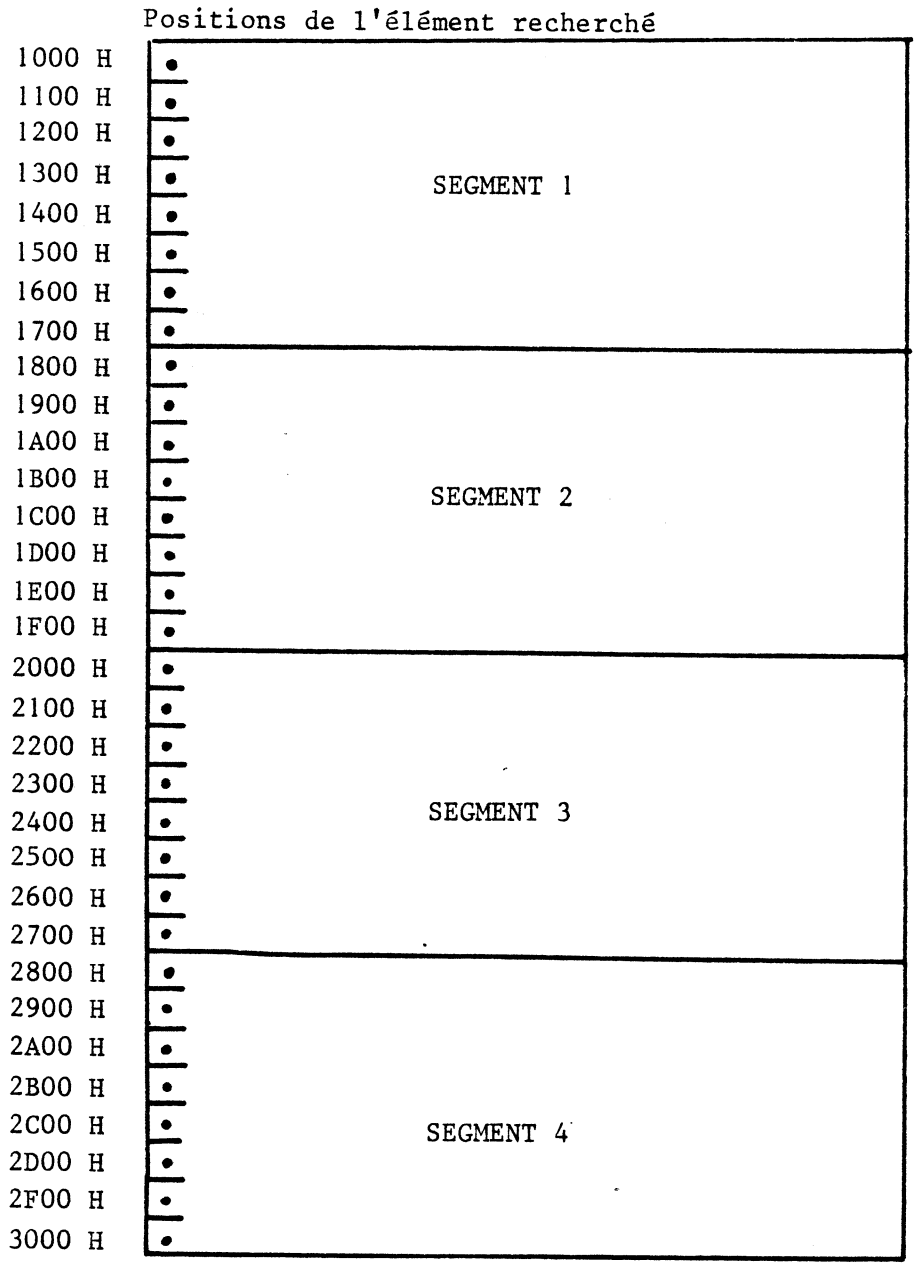
CONSTATATIONS ET COMMENTAIRES

D'après les résultats obtenus et représentés par le schéma 3.14. on constate que:

1. La durée d'exécution moyenne (quelque soit la position de l'élément) du programme avec 1, 2, 3 et 4 processeurs est réduite à 55 %, 45 % et 35 % respectivement de la durée du programme en monoprocesseur.

2. Ces résultats confirment ceux obtenus pour les autres types de parallélisme et peuvent être généralisés.

Il est à noter que les performances du système sont mesurées dans les cas les plus défavorables car tous les programmes paralléles utilisés nécessitent des accès fréquents à la mémoire centrale.



SEGMENTATION DU TABLEAU

Schéma 3.13.

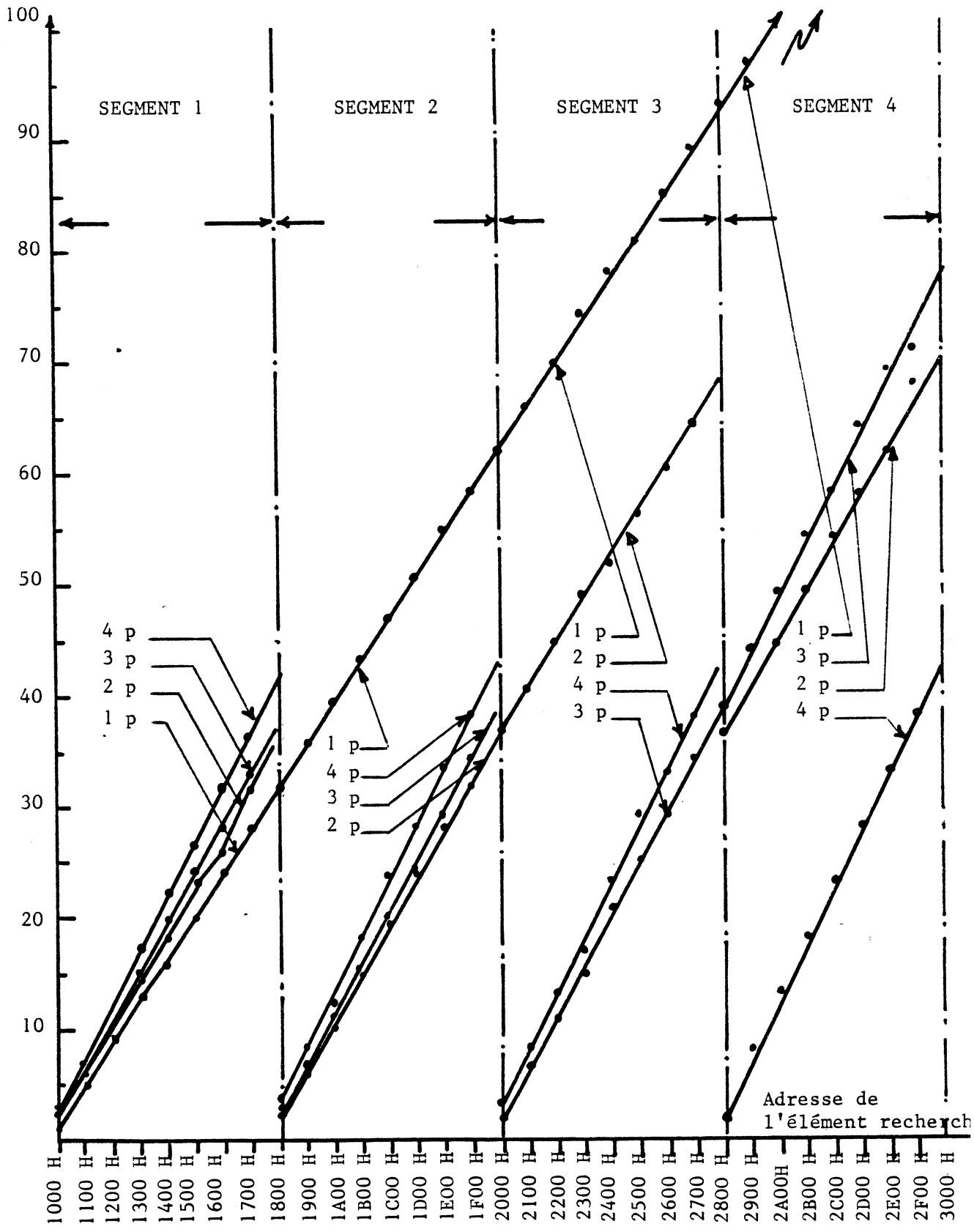

TEMPS D'EXECUTION
(1 unité = 1,17 millisecondes)
100
90
80
70
60
50
40
30
20
10
SEGMENT 1
SEGMENT 2
SEGMENT 3
SEGMENT 4
4 p
3 p
2 p
1 p
1 p
4 p
3 p
2 p
1 p
2 p
4 p
3 p
1 p
3 p
2 p
4 p
Adresse de
l'élément recherch
1000 H
1100 H
1200 H
1300 H
1400 H
1500 H
1600 H
1700 H
1800 H
1900 H
1A00 H
1B00 H
1C00 H
1D00 H
1E00 H
1F00 H
2000 H
2100 H
2200 H
2300 H
2400 H
2500 H
2600 H
2700 H
2800 H
2900 H
2A00H
2B00 H
2C00 H
2D00 H
2E00 H
2F00 H
3000 H

Schéma 3.14.

CONCLUSION

CONCLUSION

A la suite de ces expérimentations on peut affirmer que le parallélisme peut être performant pour certains types de programmes (traitement d'images, traitement de la parole,...) et sous certaines conditions.

Cependant, subsistent encore les problèmes d'écriture et de mise au point des programmes parallèles. Les méthodes classiques sur monoprocesseur ne sont plus valables. Si les résultats des programmes sont les mêmes, les traces d'exécution peuvent être différentes. Dans notre cas, on a pu retrouver les détails d'exécutions des différents processeurs grâce aux informations de parallélisme qu'on a décodées dans les blocs de contrôle.

Cette réalisation nous a permis d'évaluer un certain nombre d'algorithmes parallèles sur une machine construite à base d'unités de traitement commercialement disponible.

La structure multimicroprocesseur élaborée présente une grande fiabilité par le fait que les cartes processeurs de base sont entièrement banalisées. On pourrait réduire les risques de défaillance des processeurs d'instructions en leur faisant exécuter des séquences de test en ligne quand ils sont libres dans leurs boucles d'attente. C'est alors qu'il faudrait développer les mécanismes d'autoconnexion et de déconnexion du système dans l'éventualité d'une panne.

La réduction de la durée d'exécution des programmes parallèles sur le système multimicroprocesseur est assez satisfaisante. Elle a été évaluée pour 2, 3 et 4 processeurs à 60 %, 45 % et 30 % respectivement de la durée d'exécution par un seul processeur.

Ces expérimentations nous ont permis de mettre en évidence la plupart des problèmes de réalisation des systèmes multimicroprocesseurs (Test & Set, arbitrage du bus, interruptions...) et du traitement des programmes parallèles (éclatement de processus, synchronisation, communications interprocesseurs, privation...). Pour la plupart de ces problèmes des solutions ont été proposées et réalisées.

Des mesures plus précises des performances de la machine sont en cours (en utilisant une carte de mesure spécialisée). Les résultats seront présentés ultérieurement dans une étude détaillée de l'organisation matérielle et du coût de la réalisation par M. MIZAVAZIRI H.
Ce prototype représente un outil concret pour l'enseignement du parallélisme et des programmes parallèles. C'est aussi une référence dans les futures réalisations multimicroprocesseurs.

Une étude de l'extension du langage PASCAL (existant au laboratoire sur 68000) pour le traitement des algorithmes parallèles sera menée de pair avec une réalisation d'un multimicroprocesseur à base de ce processeur. Ce microprocesseur dispose de l'instruction Test & Set d'accès exclusif qui nous évitera la réalisation câblée de ce mécanisme.

Les mémoires caches sont souvent utilisées dans les récents microordinateurs pour accélérer l'accès à la mémoire. Une étude suivie de la réalisation du cache-mémoire dans le système multimicroprocesseur sera donc nécessaire pour améliorer les performances du système multimicroprocesseur.

oo00000oo

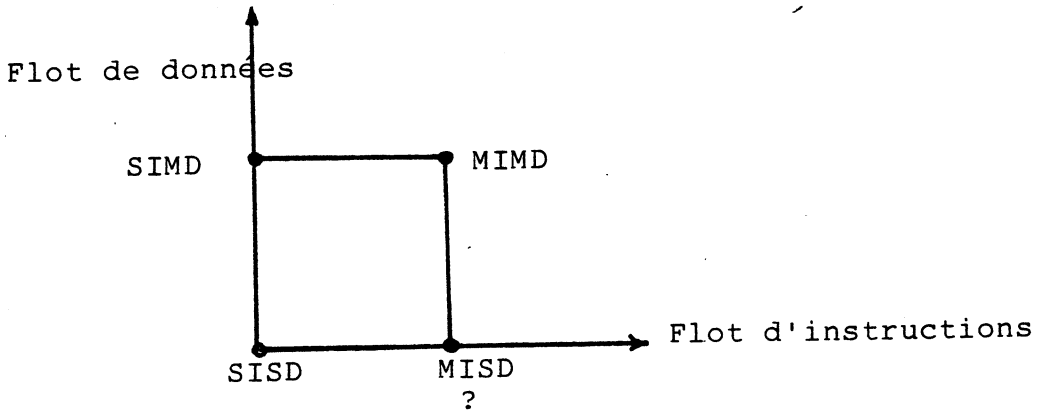
ANNEXE

CLASSIFICATION D'ERLANGEN POUR LES ARCHITECTURES MULTIPROCESSEURS

Les architectures multiprocesseurs ont été classifiées par FLYNN selon leurs flots d'instructions et leurs flots de données (schéma 1.2.1.). Les chercheurs de l'Université d'ERLANGEN travaillant dans le domaine de la classification pensent que la classification de FLYNN (FLY-72) n'est pas entièrement satisfaisante pour les raisons suivantes:

1. Une ambiguité concernant la classe M.I.S.D. (laissée vide par FLYNN) et à laquelle on a affecté les machines "Pipelines".

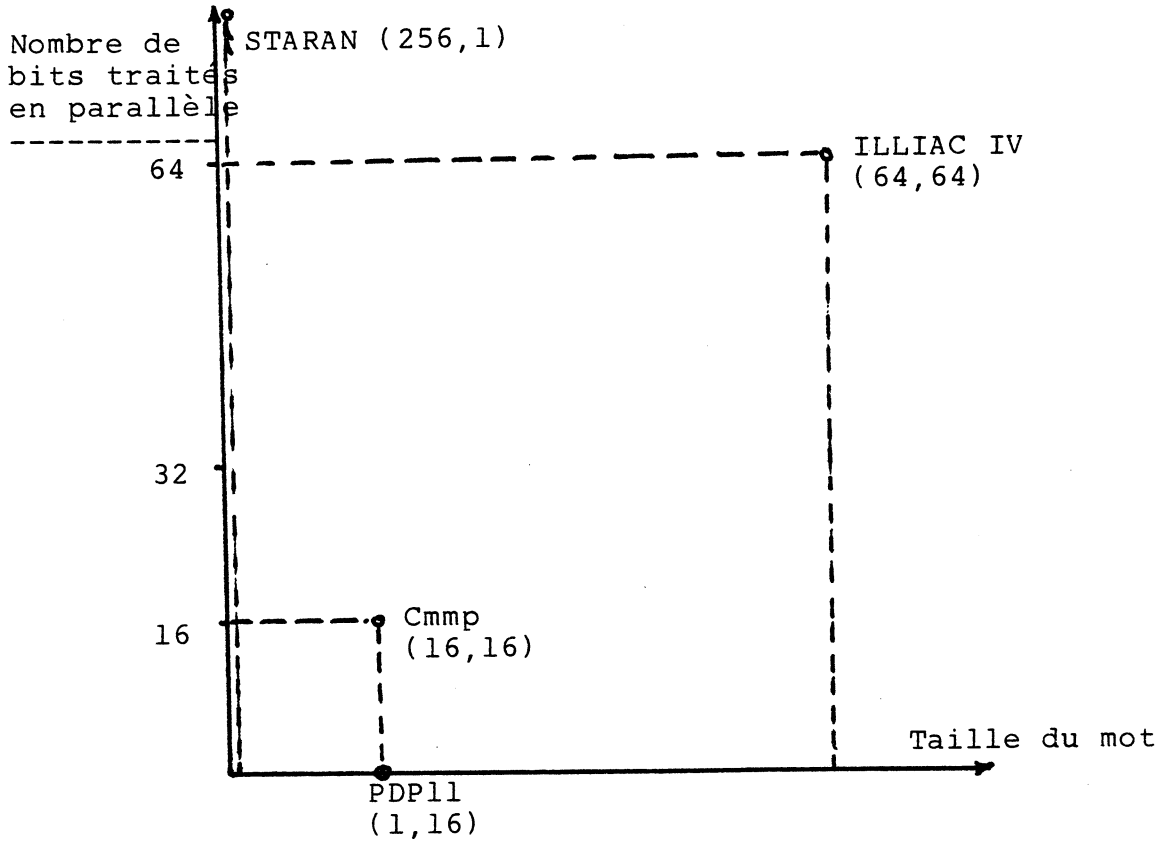
2. La puissance, la taille et les types de connections entre les différents processeurs ne sont pas pris en compte dans cette classification.



CLASSIFICATION DE FLYNN

Schéma 1.2.1.

Il en est de même pour la classification proposée par FENG (FEN-72). Cette classification est basée sur la taille du mot (nombre de bits par mot) et le nombre de ces mots traités en parallèle (qui est fonction du nombre de processeurs de traitement). On lui reproche de ne pas distinguer entre les différents niveaux de traitements (processeurs autonomes, unités arithmétiques et logiques ou circuits logiques élémentaires).



CLASSIFICATION DE FENG

Schéma 1.2.2.

La méthode de classification proposée par l'Université d'ERLANGEN (HAN-81) a pour objectif de définir les différentes classes d'architectures multiprocesseurs selon les caractéristiques suivantes:

1. Nombre des unités arithmétiques et logiques (UAL notées D).

2. Nombre des unités de contrôle des programmes (PCU notées K)

3. Taille des mots traités en parallèle (W)

4. Connexions interprocesseurs (S)

98

5. Blocs mémoires (M)

6. Modes d'opérations (Φ) pour distinguer entre les machines dirigées par les données, classiques ou Princeton, etc...

7. Restrictions du temps de connexions pour exprimer le fonctionnement des différents processeurs par rapport au temps.

Cette université a proposé une classification utilisant les trois premiers critères ci-dessus, c'est-à-dire qui distingue effectivement entre les trois niveaux de traitements:

1. Les unités de contrôle des programmes PCU: Elles utilisent des compteurs de programmes et des registres. Elles interprètent les instructions selon un microprogramme.

2. Les unités arithmétiques et logiques UAL: Elles exécutent les séquences des micro-instructions générées par les CPU et génèrent des micro-opérations.

3. Les circuits logiques élémentaires ELC: Elles traitent les micro-opérations générées par les UAL.

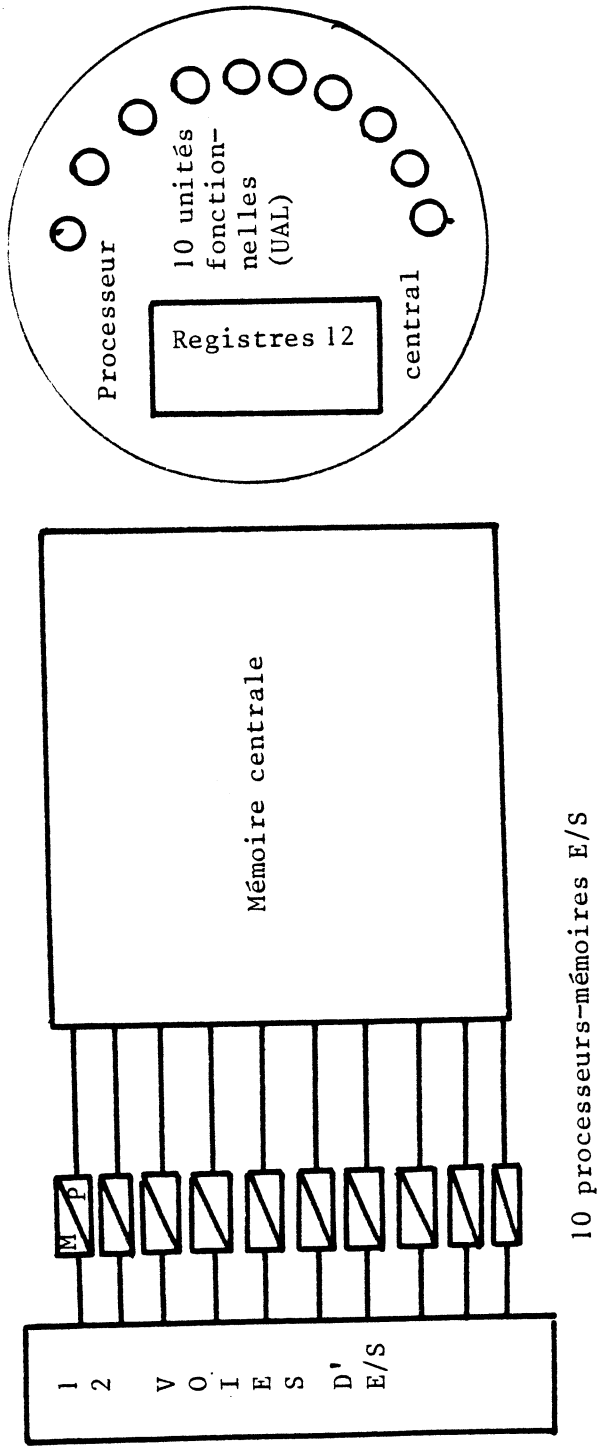
Ces trois niveaux constituent l'ensemble de traitement P, P étant le triplet (D, K, W).

Cette structure de traitement permet donc de définir des groupes d'architectures multiprocesseurs. Ainsi ILLIAC IV appartient au groupe T (1, 64, 64) et Cmmp appartient au groupe T (16, 1, 16). En général le type d'architecture est défini par le triplet T (K, D, W). Les opérateurs "+"; "X", et "V" seront utilisés pour désigner les connexions parallèles, pipeline et "ou-exclusif" (dans les architectures reconfigurables) respectivement.

Les machines "Pipelines" qui sont caractérisées par une multiplication en série des processeurs de contrôle des programmes et/ou des unités arithmétiques et logiques et/ou des circuits logiques élémentaires peuvent être représentées par cette classification comme suit:

$$T\ (\text{type}) = (k \times k',\ D \times D',\ W \times W')$$

K', D' et W' étant le nombre de stages pipeline des PCU, UAL et des ELC respectivement.



BLOCK DIAGRAMME DU CD6600: (10, 1, 12) x (1,1 x 10, 60)

Schéma 1.3.3.

Pour illustrer cet exemple de représentation prenons le processeur central du CD6600 (schéma 1.3.3.). Celui-ci est constitué d'un PCU et de 10 unités arithmétiques et logiques spécialisées fonctionnant en pipeline et traitant des mots de 60 bits. Sa représentation selon cette formule sera T (CD6600 central) = (1,1 x 10, 60).

On pourra considérer les calculateurs frontaux ou les unités d'E/S comme des constituants pipeline vis-à-vis du calculateur central. Ainsi l'on pourra représenter le CD6600 avec les 10 processeurs d'E/S traitant des mots de 12 bits (schéma 1.3.3.) comme suit:

T (CD6600) = (10, 1, 12) x (1,1 x 10, 60)

Les architectures reconfigurables pourront être représentées par l'ensemble des configurations possibles séparées par l'opérateur "V".

REMARQUE

Si cette classification résout l'ambiguité de la classe M.I.S.D de FLYNN, nous la trouvons trop détaillée et ne peut donc que constituer une représentation complète des caractéristiques d'un système. La classification de FLYNN restera donc à notre avis le groupement le plus logique et simple des architectures multiprocesseurs.

ooOOOOOoo

BIBLIOGRAPHIE

BIBLIOGRAPHIE

ARN 76: R.G. ARNOLD, E.W. PAGE
"A hierarchical, restructable multiprocessor architecture"
Proceeding of 3rd annual symposium on computer architecture-Janvier 1976

ADE 82: D. ADEHL, Sté T.I.T.N.
Projet SAMBA - Rapport technique 1982

BAE 76: J.L. BAER
"Multiprocessing Systems"
IEEE Transactions on computers - Décembre 76

FEN 72: T. FENG
"Some charactéristics of assiociative/parallel processing";
Proc. 1972 Sagamore
Comp. conference, Université de Syracuse 1972

FLY 72: M.J. FLYNN
"Some computer organizations and their affectiveness"
IEEE Transactions on computers - Septembre 1972

HAN 81: WOLFGANG HANDLER - Université d'Erlangen R.F.A.
"Standards, classifications & Taxonomy: Experience with E.C.S."
Workshop on taxonomy in computers architecture - Nürnberg - R.A.F. - Juin 1981

KU 72: D.J. KUCK, Y. MURAOKA, SHYH-CHING CHEN
"On the number of operations Simultaneously executable in Fortran - Like Programs & their resulting speed up".
IEEE Transactions on computers - Décembre 1972

KU 79: U. BARNARJEE, D.J. KUCK, SHYH-CHING CHEN, R.A. TOWLE
"Time and parallel processor bounds for Fortran - Like loops"
IEEE Transactions on computers - Décembre 1979

KRT 78: S.I. et S.P. KARTASHEV
"LSI modular computers, systems & networks computer (N11) 1978

LIP 77: G.I. LIPOVSKI, A. TRIPATHI
"A recent reconfigurable varistructure array processor"
Proceedings of 1977 international conference on parallel processing

MAS 78: E. DE MASSAS
"Etude, réalisation et utilisation d'outils logiciels adaptés à l'écriture parallèle des algorithmes".
Thèse de 3e cycle - I.N.P.G. - Septembre 1978

MAZ 74: G. MAZARE
Rapport MCS/ET/A8-1 - Décembre 1974

MAZ 78: G. MAZARE
"Structures multimicroprocesseurs: problèmes de parallélisme, définition et évaluation d'un système particulier".
Thèse d'état - I.N.P.G. - Juin 1978

MEH 79: S.K. MEHRA, J.C. MAJITHIA
"Recent trends in multiprocessor organisations"
Indian Inst. Electronics and Communications Eng. (25) 1979

NUT 75: R.D. ARNOLD, G.J. NUTT
"The architecture of a multiassociative processor"
Université du Colorado, Computer Science Dept.
Juin 1975

POL 82: M.H. POLCZYNSKY
Electronic design - Janvier 1982

RAG 79: S. RAGAB
"Etude de l'architecture interne du microprocesseur C.D.P. 1802 (Le COSMAC)"
Rapport DEA INPG - Juin 1979

SIG 81: F.C. KEMMERER, P.T. MUELLER, H.E. SHALLEY, H.J. SIEGEL, L.J. SIEGEL, S.D. SMITH
"PASM: A partitionable SIMD/MIMD system for image processing and pattern recognition"
IEEE Transaction on computers - Décembre 1981

TIB 79: L. BOUSSE, M. GOOSSENS, R. PATERNOSTER, J. TIBERGHEIN, A. VANEYNDONCK, E. VAN OOST
"A multimicroprocessor system with two shared busses"
1st European conference on parallel and distributed processing - Toulouse - Février 1979

TRU 82: V.A. TRUJILLO
"Fully reconfigurable multimicroprocessor"
Computing division - Los Almos - New Mexico
ICCC 82 - Octobre 1982

AUTORISATION DE SOUTENANCE

VU les dispositions de l'article 3 de l'arrêté du 16 avril 1974,

VU les rapports de présentation de Messieurs

. G. MAZARE, Professeur

. Y. DESWARTES, Responsable Projet SARGOS

Monsieur RAGAB Sarwat

est autorisé à présenter une thèse en soutenance pour l'obtention du diplôme de DOCTEUR-INGENIEUR, spécialité "Informatique".

Fait à Grenoble, le 19 mai 1983

Le Président de l'I.N.P.-G,

D. BLOCH

P.O. le Vice-Président,