50376
1980
80

50376
1980
80

N° d'ordre : 249

THÈSE

présentée à

L'UNIVERSITÉ DES SCIENCES ET TECHNIQUES DE LILLE

pour obtenir le titre de

DOCTEUR INGENIEUR

(Traitement de l'Information)

par

Patrick PLANCKE

ETUDE DES RESEAUX CELLULAIRES DE MICROPROCESSEURS. APPLICATIONS AU CALCUL ANALOGIQUE ET AU TRAITEMENT PARALLELE A RECOUVREMENT D'ACTIVITES.

BIBLIOTHÈQUE UNIVERSITAIRE
Section de SCIENCES
LILLE

Thèse soutenue le 19 septembre 1980 devant la Commission d'Examen

Membres du Jury	MM. C. CARREZ	Président
	V. CORDONNIER	Rapporteur
	C. MICHEL	Examinateur
	L. CARREZ	Examinateur

PROFESSEURS 1ERE CLASSE

BACCHUS Pierre	Mathématiques
BEAUFILS Jean Pierre (dét.)	Chimie
BIAYS Pierre	G.A.S.
BILLARD Jean	Physique
BONNOT Ernest	Biologie
BOUGHON Pierre	Mathématiques
BOURIQUET Robert	Biologie
CELET Paul	Sciences de la Terre
COEURE Gérard	Mathématiques
CONSTANT Eugène	I.E.E.A.
CORDONNIER Vincent	I.E.E.A.
DEBOURSE Jean Pierre	S.E.S.
DELATTRE Charles	Sciences de la Terre
DURCHON Maurice	Biologie
ESCAIG Bertrand	Physique
FAURE Robert	Mathématiques
FOURET René	Physique
GABILLARD Robert	I.E.E.A.
GRANELLE Jean Jacques	S.E.S.
GRUSON Laurent	Mathématiques
GUILLAUME Jean	Biologie
HECTOR Joseph	Mathématiques
HEUBEL Joseph	Chimie
LABLACHE COMBIER Alain	Chimie
LACOSTE Louis	Biologie
LANSRAUX Guy	Physique
LAVEINE Jean Pierre	Sciences de la Terre
LEBRUN André	C.U.E.E.P.
LEHMANN Daniel	Mathématiques
Mme LENOBLE Jacqueline	Physique
M. LHOMME Jean	Chimie
M. LOMBARD Jacques	S.E.S.
M. LOUCHEUX Claude	Chimie
M. LUCQUIN Michel	Chimie
M. MAILLET Pierre	S.E.S.
M. MONTREUIL Jean	Biologie
M. PAQUET Jacques	Sciences de la Terre
M. PARREAU Michel	Mathématiques
M. PROUVOST Jean	Sciences de la Terre

.../...

Professeurs 1ère classe (suite)

M. SALMER Georges	I.E.E.A.
Mme SCHWARTZ Marie Hélène	Mathématiques
M. SEGUIER Guy	I.E.E.A.
M. STANKIEWICZ François	Sciences Economiques
M TILLIEU Jacques	Physique
M. TRIDOT Gabriel	Chimie
M. VIDAL Pierre	I.E.E.A.
M. VIVIER Emile	Biologie
M. WERTHEIMER Raymond	Physique
M. ZEYTOUNIAN Radyadour	Mathématiques

PROFESSEURS 2ème CLASSE

M. AL FAKIR Sabah	Mathématiques
M. ANTOINE Philippe	Mathématiques
M. BART André	Biologie
Mme BATTIAU Yvonne	Géographie
M. BEGUIN Paul	Mathématiques
M. BELLET Jean	Physique
M. BKOUCHE Rudolphe	Mathématiques
M. BOBE Bernard	S.E.S.
M. BODART Marcel	Biologie
M. BOILLY Bénoni	Biologie
M. BONNELLE Jean Pierre	Chimie
M. BOSQ Denis	Mathématiques
M. BREZINSKI Claude	I.E.E.A.
M. BRUYELLE Pierre (Chargé d"enseignement)	Géographie
M. CAPURON Alfred	Biologie
M. CARREZ Christian	I.E.E.A.
M. CHAMLEY Hervé	E.U.D.I.L.
M. CHAPOTON Alain	C.U.E.E.P.
M. COQUERY Jean Marie	Biologie
Mme CORSIN Paule	Sciences de la Terre
M. CORTOIS Jean	Physique
M. COUTURIER Daniel	Chimie
Mlle DACHARRY Monique	Géographie
M. DEBRABANT Pierre	E.U.D.I.L.
M. DEGAUQUE Pierre	I.E.E.A.
M. DELORME Pierre	Biologie
M. DEMUNTER Paul	C.U.E.E.P.
M. DE PARIS Jean-Claude	Mathématiques

.../...

M. DEVRAINNE Pierre	Chimie
M. DHAINAUT André	Biologie
M. DORMARD Serge	S.E.S.
M. DOUKHAN Jean Claude	E.U.D.I.L.
M. DUBOIS Henri	Physique
M. DUBRULLE Alain	Physique
M. DUEE Gérard	Sciences de la Terre
M. DYMENT Arthur	Mathématiques
M. FLAMME Jean Marie	E.U.D.I.L.
M. FONTAINE Hubert	Physique
M. GERVAIS Michel	S.E.S.
M. GOBLOT Rémi	Mathématiques
M. GOSSELIN Gavriel	S.E.S.
M. GOUDMAND Pierre	Chimie
M. GREVET Patrice	S.E.S.
M. GUILBAULT Pierre	Biologie
M. HANGAN Théodor	Mathématiques
M. HERMAN Maurice	Physique
M. JACOB Gérard	I.E.E.A.
M. JACOB Pierre	Mathématiques
M. JOURNEL Gérard	E.U.D.I.L.
M. KREMBEL Jean	Biologie
M. LAURENT François	I.E.E.A.
Mlle LEGRAND Denise	Mathématiques
Mlle LEGRAND Solange	Mathématiques (Calais)
Mme LEHMANN Josiane	Mathématiques
M. LEMAIRE Jean	Physique
M. LENTACKER Firmin	G.A.S.
M. LEVASSEUR Michel	I.P.A.
M. LHENAFF René	G.A.S.
M. LOCQUENEUX Robert	Physique
M. LOSFELD Joseph	I.E.E.A.
M. LOUAGE Francis	E.U.D.I.L.
M. MACKE Bruno	Physique
M. MAIZIERES Christian	I.E.E.A.
Mlle MARQUET Simone	Mathématiques
M. MESSELYN Jean	Physique
M. MIGEON Michel	E.U.D.I.L.
M. MIGNOT Fulbert	Mathématiques
M. MONTEL Marc	Physique
Mme NGUYEN VAN CHI Régine	G.A.S.
M. PARSY Fernand	Mathématiques
Mlle PAUPARDIN Colette	Biologie

Professeurs 2ème classe (suite)

M. PERROT Pierre	Chimie
M. PERTUZON Emile	Biologie
M. PONSOLLE Louis	Chimie
M. PORCHET Maurice	Biologie
M. POVY Lucien	E.U.D.I.L
M. RACZY Ladislas	I.E.E.A.
M. RICHARD Alain	Biologie
M. RIETSCH François	E.U.D.I.L.
M. ROGALSKI Marc	M.P.A.
M. ROUSSEAU Jean-Paul	Biologie
M. ROY Jean-Claude	Biologie
M. SALAMA Pierre	S.E.S.
Mme SCHWARZBACH Yvette (CCP)	M.P.A.
M. SCHAMPS Joël	Physique
M. SIMON Michel	S.E.S.
M. SLIWA Henri	Chimie
M. SOMME Jean	G.A.S.
Mle SPIK Geneviève	Biologie
M. STERBOUL François	E.U.D.I.L.
M. TAILLIEZ Roger	Institut Agricole
M. TOULOTTE Jean-Marc	I.E.E.A.
M. VANDORPE Bernard	E.U.D.I.L.
M. WALLART Francis	Chimie
M. WATERLOT Michel	Sciences de la Terre
Mme ZINN JUSTIN Nicole	M.P.A.

CHARGES DE COURS

M. TOP Géard	S.E.S.
M. ADAM Michel	S.E.S.

CHARGES DE CONFERENCES

M. DUVEAU Jacques	S.E.S.
M. HOFLACK Jacques	I.P.A.
M. LATOUCHE Serge	S.E.S.
M. MALAUSSENA DE PERNO Jean-Louis	S.E.S.
M. OPIGEZ Philippe	S.E.S.

Je suis très heureux d'exprimer ici mes remerciements

à *Monsieur le Professeur Christian CARREZ qui m'a fait l'honneur de présider ce jury,*

Monsieur le Professeur Vincent CORDONNIER dont le soutien et l'aide critique m'ont permis de réaliser cette étude,

Monsieur Claude MICHEL, Ingénieur à la S.I.N.T.R.A., et Monsieur Léon CARREZ, Professeur à l'I.S.E.N., qui ont bien voulu examiner ce travail et participer au jury.

Que Mademoiselle Bénédicte FIEVET qui a eu la lourde charge de dactylographier ce document,
Monsieur et Madame DEBOCK qui en ont assuré avec gentillesse et diligence la réalisation matérielle soient remerciés et assurés de toute ma gratitude.

A mon père

TABLE DES MATIERES

INTRODUCTION

CHAPITRE I - INTRODUCTION AUX ARCHITECTURES CELLULAIRES.

1.1 - LES TABLEAUX LOGIQUES CELLULAIRES
1.1.1 - Les travaux de R.C. Minnick
1.1.2 - Modèle de Akers
1.1.3 - Modèle de Kautz
1.1.4 - Réseaux cellulaires spécialisés
1.1.5 - Intérêt des tableaux cellulaires

1.2 - AUTOMATES ET ESPACES CELLULAIRES
1.2.1 - Définition.
1.2.2 - Espace cellulaire à 4 états
1.2.3 - Conclusion

1.3. - OBJECTIFS DE L'ETUDE
1.3.1 - Une définition à priori
1.3.2 - Objectifs et contraintes
1.3.3 - Extensibilité et outil de communication
1.3.4 - Insensibilité aux pannes

1.4 - MULTIPROCESSEUR, MULTIORDINATEUR ET CELLULARITE
1.4.1 - Machines tableaux
1.4.2 - CHOPP
1.4.3 - EGPA

1.5 - AUTRES PROPOSITIONS
1.5.1 - Proposition de Lipovsky
1.5.2 - Architecture "arbre"
1.5.3 - MU-NET

CHAPITRE 2 - ARCHITECTURES SPECIALISEES

2.1 - SCHEMA DE BASE
2.1.1 - Relation utilisateur - concepteur
2.1.2 - Squelette d'une cellule

2.2 - TABLEAUX CELLULAIRES POUR LE PRETRAITEMENT DE TABLEAUX DE DONNEES

2.2.1 - Introduction

2.2.2 - Opérations associatives ligne ou colonne

2.2.3 - Compression de tableaux de données

2.2.4 - Détermination des dimensions optimales

2.2.5 - Autres éléments intervenant dans le choix des dimensions

2.2.6 - Enchaînement des opérations élémentaires

2.2.7 - Logiciel d'interprétation

2.2.8 - Implantation des fonctions de service dans l'espace cellulaire.

CHAPITRE 3 - D.D.A. CELLULAIRE

3.1 - INTRODUCTION

3.2 - L'INTEGRATEUR

3.3 - AUTRES ELEMENTS DU D.D.A.

3.4 - PRECISION ET RAPIDITE

3.5 - INTERET D'UNE ARCHITECTURE CELLULAIRE

3.6 - FONCTIONS DE ROUTAGE

3.7 - PROPOSITION D'ARCHITECTURE

3.8 - COMPORTEMENT DE LA CELLULE

3.9 - SOLUTION CHOISIE A TITRE D'EXEMPLES

CHAPITRE 4 - MACROPIPE-LINES ET ESPACES CELLULAIRES

4.1 - MACROPIPE-LINES

4.2 - CARACTERISTIQUES D'un MACROPIPE-LINE

4.3 - DEBIT ET TEMPS D'EXECUTION

4.4 - ELASTICITE

4.5 - TOPOLOGIE

4.6 - PIPE-LINE ELASTIQUE ET RECONFIGURABLE

4.7 - DIVERSES OPTIONS POUR LA CONSTRUCTION DU TRAIN PIPE-LINE
4.7.1 - Première raison
4.7.2 - Seconde raison
4.7.3 - Troisième raison

4.8 - SCRUTATION DES VOISINS - DERIVE
4.8.1 - Stabilisation statique
4.8.2 - Stabilisation dynamique

4.9 - BASE DE L'ARCHITECTURE D'UNE CELLULE

4.10 - DEUX SOLUTIONS D'IMPLANTATION
4.10.1 - Solution multibus
4.10.2 - Solution cellulaire stricte

CHAPITRE 5 - CONSTRUCTION DE MACHINES EPHEMERES DANS UN ESPACE CELLULAIRE

5.1 - POSITION DU PROBLEME

5.2 - COMMUTATION DE MESSAGE OU RESERVATION DE CHEMINS

5.3 - PARALLELISME DE BLOC OU MACHINE EPHEMERE

5.4 - MACHINE GRAPHE

5.5 - OPERATIONS DE CONSTRUCTION DE LA MACHINE GRAPHE
5.5.1 - Eclatement
5.5.2 - Jonction

5.6 - IMPLANTATION DES OPERATIONS DE CONSTRUCTION
5.6.1 - Eclatement
5.6.2 - Jonction
5.6.2.a - Une solution centralisée
5.6.2.b - Procédure de recherche et validation de la validation de la jonction basée sur la notion d'état et de basculement d'état sur message de service

5.7 - CONTRAINTE PHYSIQUES
5.7.1 - Espace cellulaire illimité
5.7.2 - Absence de conflit entre exécution et construction

5.8 - CONSTRUCTION INDEPENDANTE DE L'EXECUTION
5.8.1 - Etats transitoires
5.8.2 - Etats terminaux
5.8.3 - Message constructeur
5.8.4 - Contrôle de la construction

5.9 - CONSTRUCTION DEPENDANTE DE L'EXECUTION

5.10 - TRAITEMENT DES MESSAGES PAR L'AUTOMATE DE NIVEAU INFERIEUR
5.10.1 - Degré de liberté - Ordre de scrutation des voisins libres
5.10.2 - Les messages
5.10.3 - Message type 3
5.10.4 - Message source
5.10.5 - Message R.J. (message constructeur intermédiaire)
5.10.6 - Etat C.J.
5.10.7 - Message CAND
5.10.8 - Message VAL
5.10.9 - Message LIB
5.10.10 - Message éclatement
5.10.11 - Message de type 4 ou lien

5.11 - PASSAGE DU CODE AUX CELLULES

5.12 - UN EXEMPLE
5.12.1 - Fonctionnement statique
5.12.2 - Fonctionnement dynamique
5.12.2.1 - fréquence d'injection = 0,5
5.12.2.2 - fréquence d'injection = 1

INTRODUCTION

Ce travail se situe dans le cadre des tentatives actuelles de construire de vastes ensembles à traitement réparti fonctionnellement. Ces tentatives ont en général pour principal objectif d'exploiter au mieux les possibilités apportées par l'amélioration quasi-permanente ces dernières années des techniques L.S.I. En effet sans les progrès réalisés par la micro-électronique, argumenter sur les moyens de réaliser ces architectures eut été une démarche purement académique. Historiquement il faut d'ailleurs noter que dans le domaine des vastes ensembles multiprocesseurs, ce furent des machines S.I.M.D.(20) tels Solomon, Illiac IV qui ont vu le jour à une époque où une unité centrale était encore matière noble. A côté de ces machines et de leurs successeurs, le développement extraordinaire des microprocesseurs ainsi que l'extension de leurs capacités autorisent sans aucun doute des solutions nouvelles et originales, basées sur la distribution à un grand nombre d'unités de contrôle et d'exécution du travail donné auparavant à une seule. Ceci se traduit par l'exécution simultanée des parties d'une même tâche voire de plusieurs et donc par un gain de temps substantiel à l'exécution. C'est du moins ce qu'on peut ressentir instinctivement à l'annonce des progrès de la micro-électronique. Cependant, entre posséder l'intuition ou peut-être même l'intime conviction qu'une vaste architecture multimicroordinateur est faisable et viable et proposer une architecture proche du stade de la réalisation d'immenses difficultés se présentent. C'est que le problème de faire travailler ensemble des dizaines voire des centaines de processeurs possèdent tant de facettes diverses qu'il parait difficile de l'appréhender que ce soit d'une façon analytique ou synthétique. C'est peut-être ce qui explique la prudence des constructeurs dont les matériels ont connu une évolution somme toute assez sage. Cette inertie vient sans doute en grande partie des investissements énormes mis dans les logiciels d'accompagnement et des problèmes de compatibilité qui s'ensuivent mais peut-être aussi d'une certaine expectative devant une révolution techniquement maîtrisée dans ses réalisations élémentaires mais non dans celui des applications complexes alors qu'à l'horizon profilent les promesses des connexions optiques ou

de la supra-conductivité. Ajoutons qu'il n'est pas dans notre propos de nier qu'il est naturel qu'existe un décalage entre les constructeurs soumis aux contraintes économiques et les laboratoires de recherche où l'imagination et l'innovation ont plus de poids qu'une rentabilité immédiate (9) . De ce fait l'éventail des solutions, voire des approches envisageables pour apporter des innovations dans cette perspective est immense et fait l'objet de nombreux travaux de recherche depuis plusieurs années. Notre étude se limite à une catégorie particulière, facile à cerner dans les grandes lignes de son architecture ainsi que dans certaines applications. En ce sens la cellularité mise en avant dans ce travail doit apparaître comme un choix à priori bien que basé sur des similitudes au niveau de l'intention dans divers domaines de l'activité informatique comme les logiques cellulaires ou les automates cellulaires. C'est donc le concept de *cellulaire* qui sera le tronc commun des diverses propositions que l'on pourra rencontrer dans cette thèse. Il faut ajouter que l'exploration des notions de cellularité est apparue comme tellement vaste qu'il a fallu effectuer des choix. Ceux que nous avons fait après un premier examen sont apparus comme prometteurs ce qui ne signifie pas qu'ils soient exclusifs. Ils avaient de surcroit à nos yeux le mérite de se situer dans des contextes ouverts. Après un rapide passage en revue des notions de cellularité telles qu'elles apparaissent dans les logiques cellulaires et les automates cellulaires, l'étude se scinde en deux parties bien que l'on passe de l'une à l'autre de façon graduée. L'une se consacre à la réalisation de machines spécialisées telles les D.D.A. (Analyseurs Digitaux Différentiels), des périphériques lourds de prétraitement de tableaux de données, des machines spécialisées dans le traitement d'applications typiquement cellulaires. L'autre aborde, non sans une constante référence aux précédentes, le cas d'architectures moins spécialisées orientées vers des traitements macro-pipeline ou l'exploitation de parallélismes de bloc.

CHAPITRE 1

INTRODUCTION AUX ARCHITECTURES CELLULAIRES

Le concept de cellularité n'est pas neuf en informatique puisqu'il est à la base de deux courants de recherche déjà anciens : les automates cellulaires et les tableaux logiques cellulaires. Ces domaines dont les connexions doivent se chercher plus au niveau des intentions qu'à celui des réalisations à de rares exceptions près (41) vont nous servir d'introduction pour cerner et préciser ce que nous attendons de la cellularité. Il est cependant important de noter dès à présent que notre démarche ne peut se rattacher à l'un ou l'autre des domaines précités bien qu'ils soient tous deux d'intéressantes sources d'inspirations.

1.1 - LES TABLEAUX LOGIQUES CELLULAIRES

L'intérêt pour les tableaux logiques cellulaires est apparu au début des années 60 et n'a fait que croître au fur et à mesure que l'on percevait les possibilités présentes et à venir des méthodes d'intégration dans l'industrie des semi-conducteurs. Il serait bien sûr abusif de dire que ce concept de tableau ou réseau cellulaire est apparu avec la microélectronique car il existait de façon implicite dans des réalisations plus anciennes comme par exemple les matrices de relais ou diodes utilisées en téléphonie. Cependant c'est incontestablement les années qui ont suivi l'avènement des premiers circuits intégrés qui ont poussé certains chercheurs dans cette voie pour exploiter au mieux les possibilités d'intégration qui s'annonçaient en constante progression.

1.1.1 - Les travaux de R.C. Minnick (47)

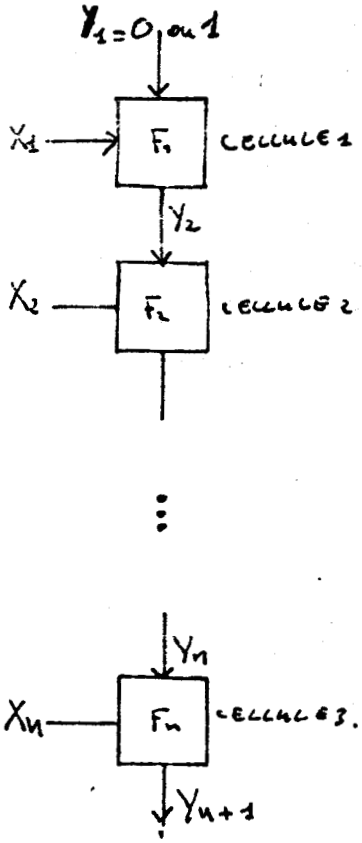
R.C. Minnick a été un des précurseurs en ce domaine. Il s'était référé aux travaux de K.K. Maître (43) qui avait étudié une "cascade" composée de cellules à deux entrées et une sortie, cette dernière réalisant une

fonction logique des entrées parmi seize possibles (cf. fig. 1.1). L'un des résultats obtenus est que le nombre de fonctions de n variables réalisables par une cascade de n cellules et $N = 0,8\ (3.6^{n-1} + 2)$. A partir de là, Minnick va montrer qu'un tableau cellulaire à deux dimensions dont la cellule de base est réduite car elle ne réalise que neuf fonctions différentes, peut réaliser n'importe quelle fonction logique.

Il l'appelera "cellule à point de coupure", se référant au fait que trois ou quatre connexions, ouvertes ou fermées en des endroits bien choisis du schéma TTL ou DTL (fig. 1.2) suffisent à implémenter les différentes fonctions logiques dont le tableau 1.I donne le résumé (a, b, c, d sont les positions des coupures possibles).

désignation fonctions	a	b	c	d	Z
0	0	0	0	0	1
1	0	0	0	1	$\overline{Y}$
2	0	0	1	0	$\overline{X} + \overline{Y}$
3	0	0	1	1	$\overline{X}\,\overline{Y}$
4	0	1	0	0	$X + Y$
5	0	1	0	1	$X\,\overline{Y}$
6	0	1	1	0	$X \oplus Y$
7	0	1	1	1	0
13	1	1	0	1	X = S, Y = R (flip-flop)

Tableau 1.I



ig. : 1.1. cascade de Maitra

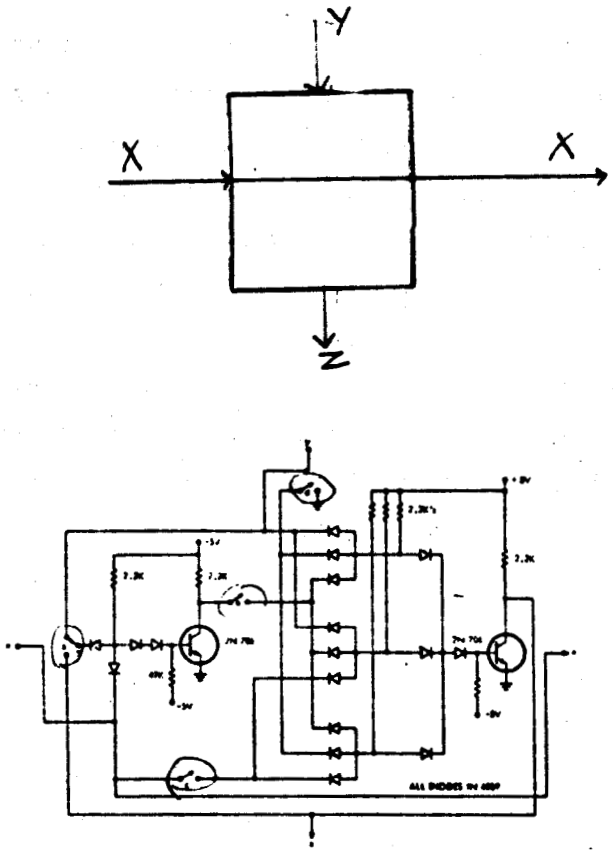


Fig : 1.2 Cellule de Minnick = représentation et implantation DTL

X_4	X_3	X_2	X_1	u_5	u_4	u_3	u_2	u_1
0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	1	0
0	0	1	0	0	0	0	1	1
0	0	1	1	0	0	1	0	0
0	1	0	0	0	0	1	0	1
0	1	0	1	0	0	1	1	0
0	1	1	0	0	0	1	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	0	1	0	0	1
1	0	0	1	1	0	0	0	0

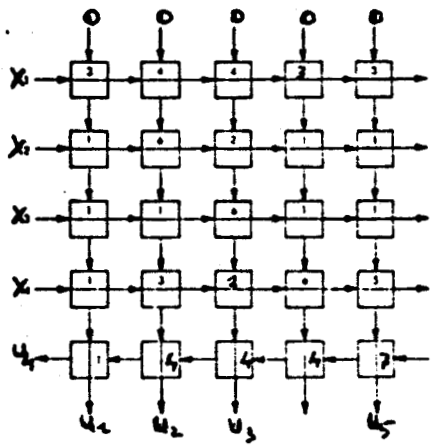


Fig. : 1.3 Tableau réalisant la fonction d'incrementation u = X + 1

BUS LILLE

Les coupures qui déterminent l'état des cellules étaient faites au tout dernier stade de la fabrication. Une fois celle-ci effectuée, on ne pouvait revenir en arrière. La possibilité de programmer la fonction réalisée dans une cellule élémentaire par un ou deux bits de mémoire enlevait cette rigidité tout en permettant de voir la cellule comme une cellule mémoire élémentaire. C'est notamment la voie suivie par Kautz qui conduit notamment à la conception de tableaux mémoires spécialisées (mémoire associative, trieuse, etc...). Une autre voie était de simplifier les fonctions logiques de la cellule tout en gardant au tableau son universalité, c'est-à-dire la possibilité de produire n'importe quelle fonction logique. C'est notamment la voie suivie par Akers.

1.1.2 - Modèle de Akers (1)

La cellule de base est très simple. Elle possède trois entrées et deux sorties qui en fait sont identiques comme le montre la figure 1.4.

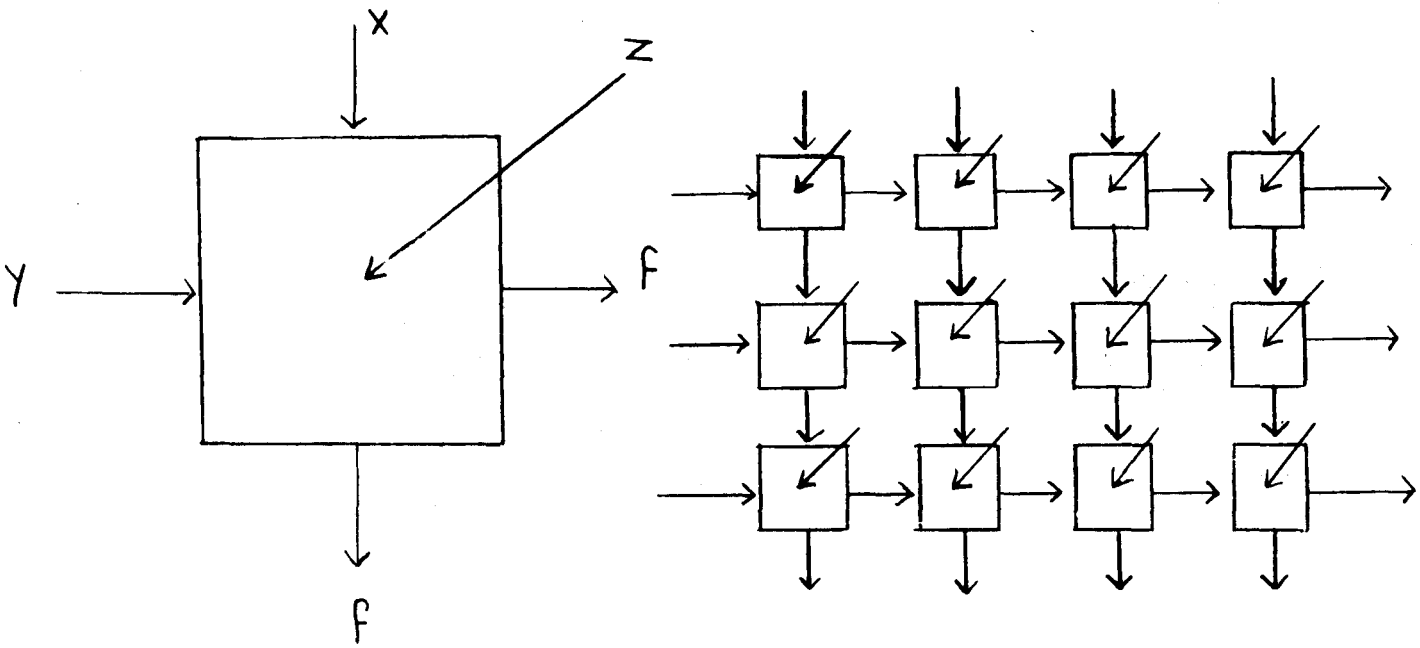
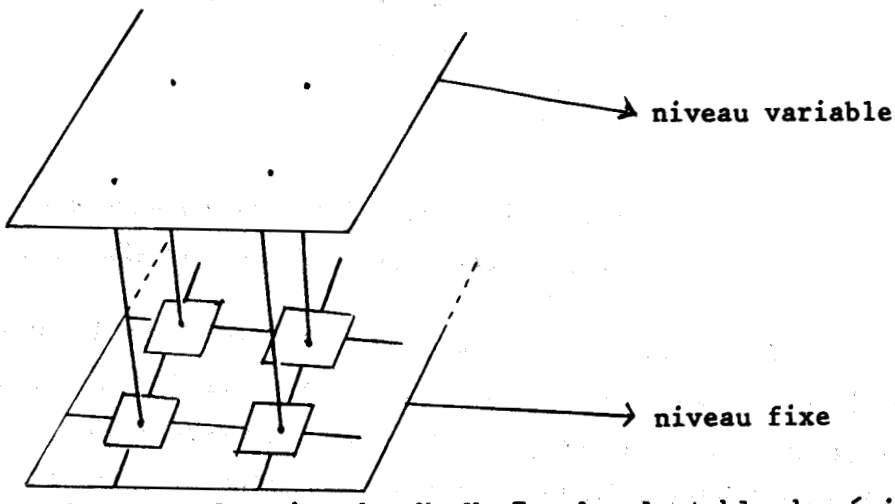


Fig. :1.4 - Reprèsentation symbolique de la cellule et d'un tableau de Akers.

Il est commode de voir le tableau comme la superposition de deux niveaux : un niveau logique (fixé), un niveau d'entrée (entrée Z).



La sortie f est fonction des X, Y, Z selon la table de vérité ci-dessous :

X	Y	Z	f
0	0	0	0
1	0	0	-
0	1	0	0
1	1	0	1
0	0	1	0
1	0	1	-
0	1	1	1
1	1	1	1

Les barres dans la colonne f indiquent que l'on interdit les combinaisons d'entrée telles que $X \leq Y$. On peut montrer qu'il suffit pour cela que la ligne supérieure ait ses entrées X forcées à zéro et la colonne de gauche ses entrées Y à 1. Les entrées X et Y ne sont pas accessibles et ce sont les entrées Z qui sont connectées aux variables booléennes dont le tableau réalise la fonction F qui sort au coin bas et droit.

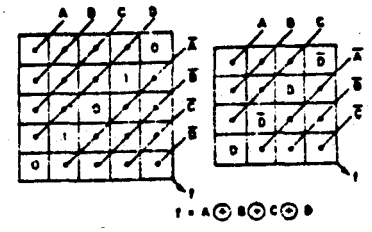
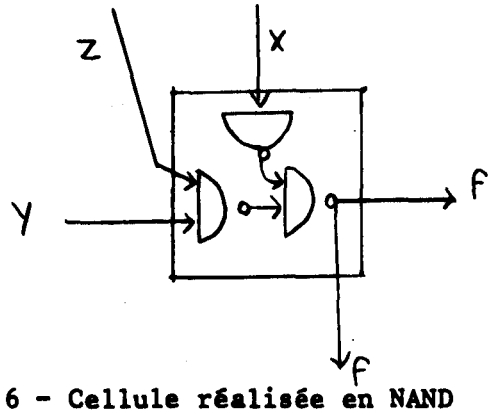


Fig. : 1.5 - Réalisation d'une fonction de quatre variables = les deux tableaux réalisent la même fonction.

Il est possible de prouver l'universalité du tableau, au sens donné à ce mot dans le paragraphe précédent. Il faut noter aussi qu'il n'existe pas d'algorithme permettant de minimiser à coup sûr le nombre de cellules employées à la réalisation d'une fonction donnée.

Trois portes NAND suffisent à implanter la cellule :



$f = X + Y \cdot Z$

Fig. : 1.6 - Cellule réalisée en NAND

Notons que d'autres formes sont possibles puisque la table de vérité correspond aussi, étant donné les occultations qui y ont été pratiquées à :

$$f = Y \circ (X + Z)$$
$$X \circ Y + X \circ Z + Y \circ Z$$
$$X \circ \overline{Z} + Y \cdot Z$$

Des modèles assez proches de ce dernier ont été proposés par S.N. Kukrya et I. Ngo chen (36) et C. Sung (65), cette liste étant loin d'être exhaustive.

1.1.3 - Modèle de Kautz "Logique dans la mémoire ou mémoire dans la logique" ?

C'est la question qui vient à l'esprit avec cette autre conception d'un espace cellulaire, développée notamment par W.H. Kautz (32)
En fait dès qu'on se donne la possibilité de mémoriser un ou quelques bits dans une cellule, un même tableau peut être regardé comme de la logique enrichissant un tableau mémoire ou comme un tableau logique dont les éléments sont programmés à partir des valeurs données aux bits mémorisés. On peut donc dans le cas présent modifier la fonction réalisée par un tableau cellulaire étant bien entendu que cela n'est possible qu'en compliquant l'architecture de la cellule par rapport aux propositions précédentes.

La figure 1.7 donne l'exemple d'une cellule et d'un tableau cellulaire réalisant la fonction de mémoire trieuse, c'est-à-dire rangeant par ordre décroissant les mots qu'on y injecte.

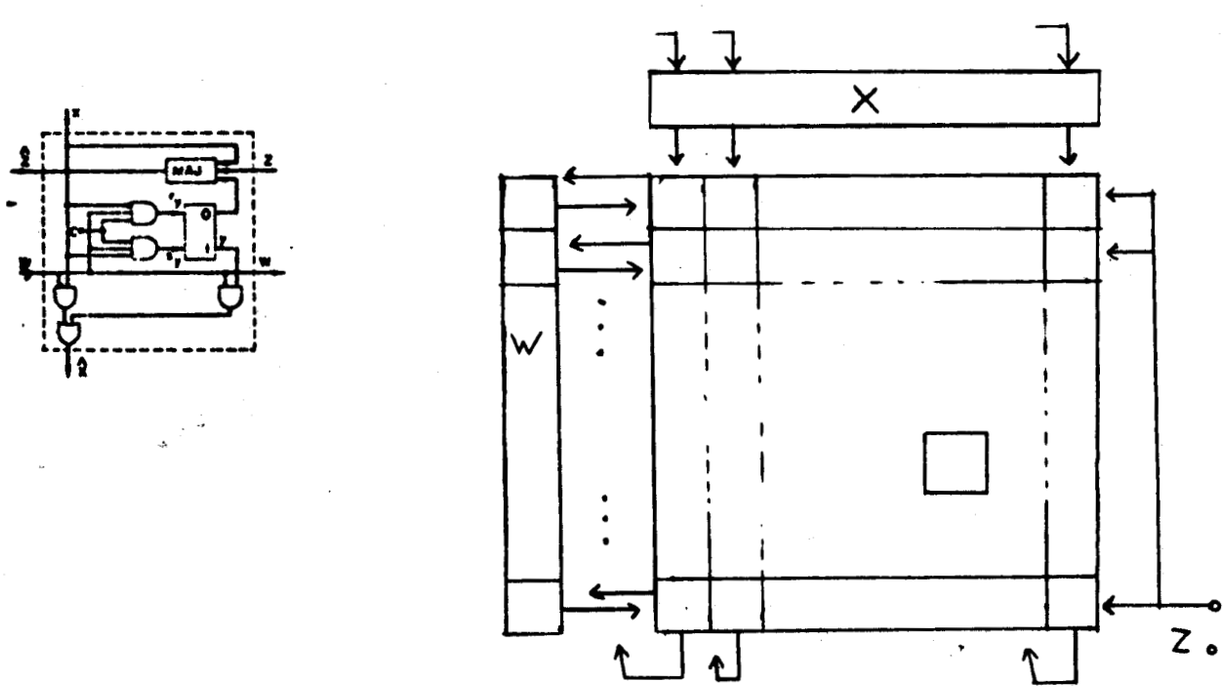


Fig. : 1.7 - "sorting array de Kautz."

$\hat{X} = \overline{\omega} \cdot X + \omega \cdot Y$

$Sy = \omega \cdot c \cdot X \qquad Ry = \omega \cdot c \cdot \overline{X}$

$Z = M(X, \overline{Y}, Z) = X \cdot \overline{Y} + Z \cdot X + Z \cdot \overline{Y}$

Y : contenu du flip-flop

1.1.4 - Réseaux cellulaires spécialisés

L'universalité des tableaux logiques cellulaires proposés ci-dessus est mise en avant parce que naturellement un ordinateur n'est qu'un vaste assemblage de circuits logiques combinatoires et séquentiels. D'où la tentation de trouver la cellule ou la brique élémentaire permettant de le construire tout entier. En fait, dans la pratique il apparait qu'il faille surtout considérer un tableau logique cellulaire des types précédents (hormis celui de Kautz) comme des schémas d'aide à la conception des circuits intégrés dans la première phase de cette dernière. Ensuite on optimise le nombre de portes pour améliorer le rapport nombre de portes élémentaires/surface semi-conducteur et passer à la réalisation effective du circuit intégré (21).

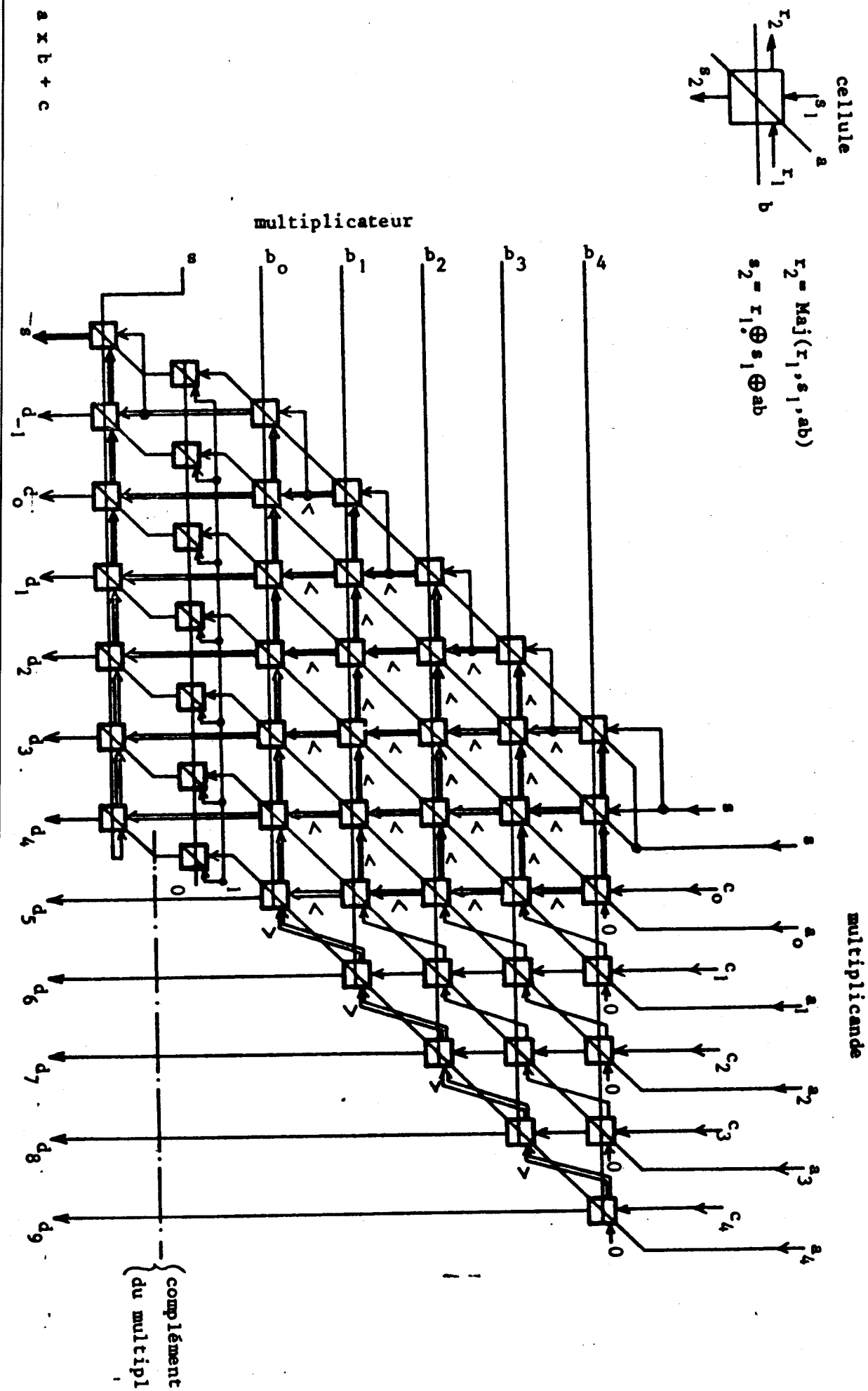
Il est aussi des structures cellulaires qui ne visent pas cette universalité. Tout d'abord il peut exister des tableaux cellulaires spécialisés, réalisant notamment des classes intéressantes de microprogrammes (31) , du traitement d'image tels CLIP 2 et 3 (42) etc...

Ensuite il existe d'autres types de réseaux cellulaires dont la raison d'être est d'optimiser le temps de réalisation d'une fonction donnée, notamment en calcul logique et arithmétique, se présentant souvent sous forme arborescente (2, 21, 24, 69).

La figure 1.8 donne un exemple de multiplicateur cellulaire (21).

Ces structures n'étant pas toujours régulières, on ne peut leur donner le nom de tableaux mais de réseaux.

Fig. : 1.8 - Multiplicateur complet d'entiers signés (complément à 2)
6 bits x 6 bits : 49 cellules.



d = a x b + c

BUS LILLE

Leur but principal est la conception de circuits spécialisés très rapides.

1.1.5 - Intérêt des tableaux cellulaires

Les promoteurs de ces structures ont essentiellement pour objectif de faciliter la conception, la fabrication, la testabilité voire l'utilisation des circuits qui en découlent.

Ces structures observent une règle fondamentale dans la conception des circuits intégrés qui est de minimiser le nombre des connexions (ou plus) d'un puce par rapport à la complexité interne de cette dernière. Un autre avantage appréciable est que la modularité naturelle d'un tableau cellulaire permet de le décomposer facilement en sous-blocs qui correspondent à autant de puces aujourd'hui mais à une seule demain. C'est en ce sens que l'on peut dire qu'une extrême régularité permet de suivre facilement la technologie puisqu'un nouveau bond dans les possibilités d'intégration ne remet pas forcément en cause la structure logique de systèmes existants.

On peut ajouter qu'une structure cellulaire possède à priori le pouvoir de s'accommoder d'une défaillance d'un ou plusieurs éléments. Il faut préciser qu'en fait ceci n'est vrai que dans certains cas : Par exemple dans le cas de la mémoire spécialisée de Kautz, une panne ponctuelle peut se traduire par une simple diminution de la capacité mémoire. Il n'irait pas de même dans le cas d'un tableau de Akers intégré de façon à réaliser une fonction donnée. En effet même si l'on s'est donné la possibilité d'isoler une cellule défectueuse, ce qui compliquerait la fabrication du chip, le tableau se trouverait amputé d'une ligne et d'une colonne. Il pourrait donc resservir pour réaliser une autre fonction logique mais certainement plus la même qu'avant la panne.

L'intérêt est nul au stade de l'utilisation. Il le serait moins en cours de fabrication si on a prévu l'existence de lignes et colonnes redondantes qui remplaceront celles défectueuses avant le stade du cablage des entrées Z (cf. fig. 1.4 et 1.5).

Ceci dit, si cette qualité d'insensibilité aux pannes mise en avant par certains promoteurs des tableaux logiques cellulaires est très relative, elle se retrouvera avec beaucoup plus de netteté dans une architecture cellulaire de microprocesseurs.

1.2 - AUTOMATES ET ESPACES CELLULAIRES

Contrairement aux logiques cellulaires qui se sont développées en prise directe sur une technologie précise, les automates cellulaires sont nés de motivations plus générales, plus diverses aussi ainsi qu'une démarche plus formelle. Parmi ces motivations voisinent côte-à-côte le souci de trouver un outil pratique pour certains traitements (par exemple la reconnaissance de formes ou le traitement d'image), des recherches sur Generi (ex. : longueur des jardins d'Eden (58) et souvent un mélange des deux tel les L-systèmes du Professeur Lindenmayer fondateur de la botanique théorique (39).

Historiquement, d'après Arbib (4) , c'est à Von Neumann que l'on peut attribuer les premiers travaux sur ce qui allait devenir la théorie des automates cellulaires encore que ce terme précis n'apparut que plus tard. Dans un souci évident de se rapprocher du vivant, il avait étudié un espace cellulaire à deux dimensions dont la cellule élémentaire pouvait occuper 29 états différents. Cet espace cellulaire avait la propriété d'être autoreproducteur et universel, c'est-à-dire de traiter n'importe quelle fonction calculable, au sens de la machine de Türing par exemple.

Une des préoccupations de Von Neumann était incontestablement de rendre un système de traitement assez flexible et bien organisé pour s'affranchir de défaillance partielle. Il faisait remarquer que "si chaque erreur doit être détectée, expliquée et corrigée, un système vivant ne vivrait pas plus d'une milliseconde" c'est également une préoccupation que l'on trouvera dans la suite de ce mémoire. En effet, si en 1950, année d'où date cette remarque la technologie n'était pas prête à suivre cette voie, il en va autrement aujourd'hui (67).

Savoir maintenant si les automates cellulaires peuvent être une source directe d'inspiration pour un ingénieur concepteur est difficile. R. Vollmar dans un article de synthèse (68) sur ce sujet est très prudent en répondant à cette question, formulée d'ailleurs un peu différemment : "Est-ce que des connexions entre les calculateurs parallèles et les espaces cellulaires existent ?" Il y fait une réponse en deux temps : "la théorie classique des automates donne les automates d'état fini et avec "marques" comme modèle de calculateurs universels. En ce sens les automates cellulaires peuvent être considérés comme les modèles pour des processeurs en tableaux. Les résultats de cette théorie ne sont pas applicables directement mais mon opinion est que l'étude de ces sujets facilitent pour le moins la pensée parallèle ("parallel thinking" dans le texte)".

A.R. Smith (68) inscrit la théorie des automates cellulaires dans le cadre plus large de la théorie des polyautomates qu'il définit comme plusieurs automates interconnectés opérant en parallèle pour former un automate plus grand. Un espace cellulaire a une définition plus restrictive. Pour sa part, Vollmar écrit qu'intuitivement un espace cellulaire est obtenu par de nombreux automates semblables attachés à des points à valeur entière d'un espace euclidien. Chacun des automates est connecté à un nombre fini d'autres automates selon un certain schéma.

Tous travaillent simultanément selon la même fonction de transition.

La définition suivante ne fait que formaliser la précédente et nous ne la donnons que pour mémoire : Un espace C est déterminé par :

- l'alphabet d'état A, c'est-à-dire l'ensemble fini et non vide des états qu'une cellule peut occuper. Un état spécial q_0 (appelé aussi état de repos) est distingué par le fait qu'il reste inchangé tant que les cellules directement voisines sont toutes à l'état q_0.

- la dimension d de l'espace.

- l'index N de voisinage, c'est-à-dire un n-uple d'éléments distincts de Z^d qui décrivent les positions relatives des cellules connectées directement à une cellule donnée.

- la fonction de transition globale F qui détermine de façon bijective (sous certaines conditions) la fonction de transition locale.

Dans le paragraphe qui suit, est décrit un petit exemple qui servira d'illustration à ces définitions.

1.2.2 - Espace cellulaire à 4 états de F. Nourai et R.S. Kashef

Il s'agit d'un espace cellulaire à 4 états qui peut supporter un calculateur universel qui serait en l'occurrence une machine à deux registres proposée par Minsky (48) . Il n'est pas auto-reproducteur comme c'est le cas du modèle proposé par E.F. Codd (14) mais qui lui possède 8 états et est un affinage de la cellule à 29 états de Von Neumann.

La structure qui soutient le modèle est un espace à deux dimensions que nous pouvons supposer infini, divisé en unités carrés désignés par un doublet $(i,j) \in I \times I$, $I \in N$. Chaque carré unitaire contient une copie de la même machine à 4 états et s'appelle une cellule. Les quatre états seront notés 0, 1, 2, x.

On peut considérer qu'à chaque cellule est associé cinq voisins : elle-même et les quatre cellules adjacentes non diagonales $(i + 1,j)$, $(i - 1,j)$ $(i, j + 1)$, $(i, j - 1)$.

Les cellules changent d'état à des instants discrets $t = 0, 1...$ suivant une fonction f qui est la fonction de transition définie dans le paragraphe précédent. Cette règle de transition n'est autre qu'un ensemble de 43 transitions instables pouvant se mettre sous la forme d'un sixtuple ordonné :

$$(v_t(i,j),\ v_t(i + 1,j),\ v_t(i,j + 1),\ v_t(i - 1,j),\ v_t(i, j - 1),\ v_{t+1}(i,j)).$$

Comme on le voit le dernier élément du sixtuple donne l'état résultant. Puisque le premier donne l'état initial, une transition sera dite stable si le premier et le dernier élément du sixtuple sont identiques, instable si l'inverse est vrai. L'état 0 est l'état q_0 défini auparavant.

Si on ajoute que la règle de transition est isotropique, la table 1.II la résume entièrement.

(0,0,0,2,x,x)	(1,2,x,x,x,2)	(2,1,x,x,x,x)	(x,1,x,x,x,1)
(0,0,0,0,2,x)	(1,1,x,2,x,x)	(2,x,x,x,x,x)	(x,0,0,0,1,2)
(0,0,0,1,1,1)	(1,2,2,x,x,2)	(2,1,1,x,x,x)	(x,1,1,x,x,1)
(0,0,1,x,1,2)	(1,0,2,0,x,x)	(2,2,x,2,x,x)	(x,1,x,1,x,2)
(0,1,x,x,x,x)	(1,0,0,2,x,x)	(2,1,1,1,x,x)	(x,0,0,1,x,1)
(0,0,1,0,x,x)	(1,0,0,x,x,0)	(2,0,0,0,x,1)	(x,0,1,0,x,1)
(0,1,x,1,x,x)	(1,0,0,0,0,0)	(2,0,x,0,x,1)	(x,0,1,0,2,1)
(0,0,1,1,x,1)	(1,0,0,0,x,0)	(2,0,0,0,0,0)	
(0,1,1,1,x,x)	(1,0,x,0,x,x)	(2,0,x,1,x,x)	
(0,0,2,0,x,x)	(1,x,x,x,x,0)	(2,2,x,x,x,1)	
(0,0,2,x,x,x)	(1,1,1,1,x,2)		
(0,1,2,1,x,1)	(1,0,x,x,x,2)		
	(1,0,1,0,x,0)		
	(1,0,0,1,x,0)		

Tableau 1-II : Règle de transition du modèle de Nourai et Kasheff.

Remarquons maintenant que si deux cellules consécutives dans une triple couche de cellules à l'état x sont respectivement dans l'état 2 et 1 à l'instant t, nous aurons à l'instant t + 1 la situation shématisée par la figure suivante :

1	2	3	4	5	6	7
x	x	x	x	x	x	x
x	x	2	1	x	x	x
x	x	x	x	x	x	x

à t

$\longrightarrow$

x	x	x	x	x	x	x
x	x	x	2	1	x	x
x	x	x	x	x	x	x

à t + 1

Nous n'irons pas plus avant dans cet exemple. On comprend maintenant aisément que l'on puisse construire à partir de rangées de cellules semblables à celles ci-dessus des "éléments primitifs" qui auront chacun une propriété particulière quant à la transmission ou à l'intéraction des états portant l'information.

La configuration ci-dessus est la plus simple et pourrait se nommer un "fil" ou un "lien" puisqu'il y a un simple transfert gauche-droite d'une information élémentaire. On pourra ainsi former des jonctions, des retards, des croisements. Avec les éléments primitifs définis à partir de la règle de transition, on peut construire les organes de base (ou, ou exclusif, horloge, inverseur, etc...) d'une machine universelle.

1.2.3 - Conclusion

L'exemple choisi dans le paragraphe précédent amène quelques observations. Tout d'abord une application immédiate des résultats obtenus à la conception d'une architecture de machine est impensable actuellement. La table de transitions est relativement complexe et nécessiterait pour être implanté par la logique binaire que nous connaissons de nombreuses portes élémentaires par cellule. Or il faut plus de cent soixante cellules pour réaliser une porte OU. Il a été choisi plus pour son intérêt de démonstration que d'un point de vue économique. Il faut d'ailleurs rester prudent

dans ce genre d'affirmation car les exemples de récupérations tardives par le monde technique des résultats fondamentaux sont assez nombreux pour ne voir là qu'un divertissement de chercheurs. Peut-être que l'avenir nous réserve des ordinateurs basés sur une architecture cellulaire développée à partir d'un composant dont conjointement la composition et les lois de la physique auront déterminé une fonction de transition "naturelle" ?

A court terme, des retombées beaucoup moins aléatoires des recherches entreprises dans le cadre des automates cellulaires peuvent être attendues dans le domaine des traitements d'image ou de reconnaissance des formes (10,68). Par ailleur elles peuvent faire surgir des algorithmes parallèles pouvant s'exécuter sur des machines parallèles qui ne seraient pas obligatoirement cellulaires. D'autre part si une application directe de la théorie des automates cellulaires à la conception de machines parallèle parait difficile bien que les motivations initiales soient fort proches, étudier une structure multimicroordinateur qui serait dotée, partiellement ou totalement, d'un comportement d'espace cellulaire ne doit pas être dénué d'intérêt. Ce n'est pas tant du point de vue des traitements mais plutôt celui des relations entre cellules et entre celles-ci et le monde extérieur qu'il y a de grandes chances de récupération de certains résultats acquis dans le domaine des automates cellulaires. On peut penser notamment aux diverses présentations et solutions du "firing squad synchronisation problem" et d'une façon générale à tout problème de propagation dans un espace cellulaire. Dans des cas précis, la facilité apportée par la technologie L.S.I. pour réaliser des cellules à la fois denses (monochip) et puissantes (programmables ou microprogrammables) permet d'envisager la réalisation éventuelles d'automates cellulaires de grandes dimensions. La cellule de CLIP 3 (42) est un précurseur de telles réalisations dans le domaine du traitement d'image.

1.3 - OBJECTIFS DE L'ETUDE

Après avoir passé en revue deux définitions de la cellularité couramment admises dans la littérature, on précise les objectifs de notre étude en expliquant ainsi le choix du concept de cellularité par rapport à ces objectifs.

1.3.1 - Une définition à priori

Dorénavant nous appelerons espace cellulaire un ensemble de modules élémentaires de traitement, tous identiques appellées cellules, connectées en un nombre fini de points à un nombre fini de voisins dans un espace euclidien de dimension 2 ou 3. Cette identité suggère la conformité physique, les mêmes microprogrammes de base et les mêmes mécanismes de connexions avec les voisins. De plus le maillage est régulier, c'est-à-dire identique à lui-même par translation.

La cellule comportera le plus souvent un microprocesseur travaillant sur le schéma classique généralement attribué à Von Neumann mais il s'agit là d'un choix à priori parce que les microprocesseurs sur le marché relèvent tous de ce principe. D'autres solutions pourraient fort bien être envisagées dans des cas particuliers. Un corollaire important découlant immédiatement de la définition donnée est qu'il n'existe pas d'outil indépendant de communication à proprement parler dans un espace cellulaire. Si deux cellules non voisines veulent se passer des messages, elles ne peuvent le faire que par l'intermédiaire d'une chaîne de cellules voisines. On peut aussi exprimer cela en disant que l'outil de communication est réparti, qu'il utilise les cellules elles-mêmes qui peuvent donc être indifféremment des noeuds de traitement ou de routage, faisant de la commutation de message. On peut noter qu'une architecture employant par exemple STUD (15,17) processeur cellulaire de communication, ne serait pas elle-même une architecture cellulaire dans notre définition.

1.3.2 - Objectifs et contraintes

Le premier objectif est la réalisation d'une architecture de machine, bénéficiant le plus largement possible des progrès acquis dans les techniques d'intégration dont l'aboutissement est la production massive de microprocesseurs, basée sur une structure en réunissant un très grand nombre. Une contrainte immédiate apparait car pour bénéficier à plein de la baisse constante des prix des composants semi-intégrés-conducteurs, il ne faut pas la contrer par d'autres facteurs à l'évolution inverse. On peut penser en particulier aux connexions, alimentations mais aussi en sus de ces coûts de fabrication à ceux de la maintenance et de l'utilisation. Une architecture cellulaire respecte cette contrainte (35,44).

Un second objectif est de procurer à une structure multimicro-ordinateur une certaine insensibilité à des pannes ou défaillances ponctuelles. Là aussi une architecture cellulaire apparait intéressante sous réserve d'avoir doter individuellement chaque cellule d'un logiciel de service assez riche.

Un troisième objectif est de minimiser les interventions de contrôle externes à l'espace cellulaire. La contrainte à respecter dans ce cas est que la réduction ou l'absence d'un contrôle externe soit compatible avec la fonction dévolue à l'espace cellulaire. Ce ne peut donc faire l'objet que d'une discussion coup par coup.

Un autre objectif est la réalisation d'une architecture facilement extensible.

1.3.3 - Extensibilité et outil de communication

Un des grands avantages de la cellularité peut se résumer dans le mot extensibilité. Cela signifie que si on utilise une architecture cellulaire d'une taille donnée, on peut accroître à tout instant la capacité

de cette dernière par l'adjonction de nouvelles cellules sans aucune remise en cause de la construction initiale. A cet égard on peut dire qu'une architecture cellulaire est extensible à l'infini alors qu'une architecture non cellulaire est limité par le moyen, quelque soit ce dernier, qui assure les communications entre les éléments de la structure.

Prenons quelques exemples :

1 - LE BUS

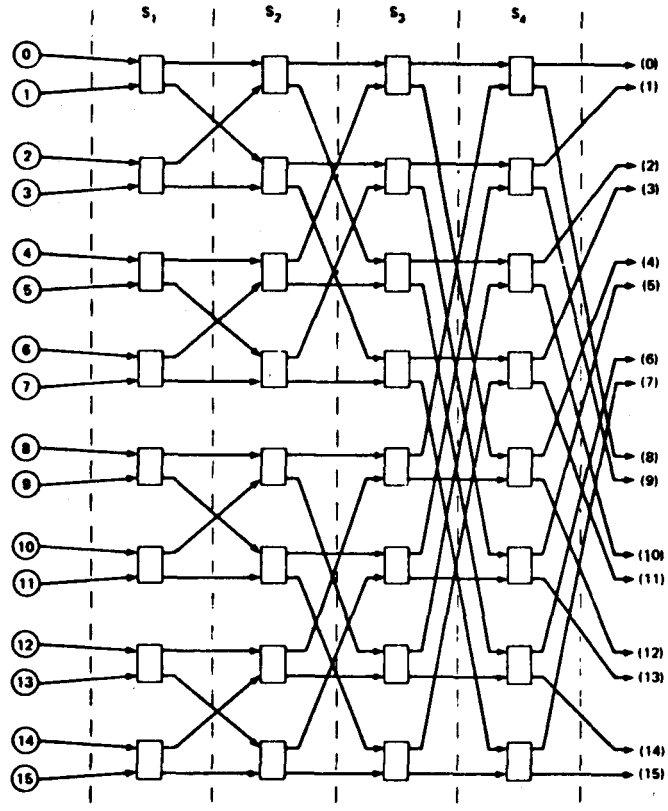
C'est un outil universel, très apprécié car très pratique pour rendre modulaire un système à composants peu nombreux. Il est extensible sous certaines conditions et de façon limitée. Augmenter en de grande proportion le nombre d'objets connectés à un bus appartenant à un système déjà construit est illusoire car s'il ne sature pas, il devient très lent.

2 - MATRICE DE CONNEXION (CROSS BAR)

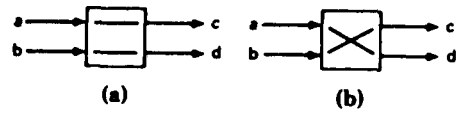
C'est l'outil intrinsèquement le plus rapide. Cependant il est d'une extensibilité nulle et doit être purement remplacé si on veut accroître le nombre d'éléments du système qui l'employait.

3 - RESEAU DE CONNEXION (37,52)

C'est un outil de communication qui possède nombre d'attraits : il est universel mais beaucoup moins onéreux que le précédent d'autant plus qu'il se prête à une intégration poussée. N'est-ce pas un cas particulier de tableaux logiques cellulaires ? Cependant il n'est pas vraiment extensible car un tableau comme celui montré par la fig. 1.9 possède $n \cdot \frac{N}{2}$ éléments où N est le nombre de processeurs connectés et $n = \log_2 N$.



The indirect binary 4-cube array.



The switch node. (a) Direct connection. (b) Crossed connection.

Fig. 1.9 : Réseau de connexion (52)

Il doit être clairement admis qu'un outil de communication quelqu'il soit apporte une grande souplesse car il permet de relier un processeur à un grand nombre d'autres éléments, si ce n'est à tous. Cette souplesse disparait dans une architecture cellulaire. Cependant il faut remarquer que plus le nombre d'éléments utilisés dans une structure s'accroît, plus l'outil devant les réunir devient ou complexe ou lent, donc plus cher (à la construction ou à l'exploitation) et sujet aux pannes.

Cette qualité de souplesse dans ce cas n'est plus aussi apparente. C'est pour cela qu'il faut en général réserver l'emploi d'outils de communications à des fédérations petites ou moyennes de processeurs d'autant plus performants qu'ils sont moins nombreux. Il faut tempérer cette affirmation

en précisant que les tableaux de commutation paraissent intéressants pour de grandes configurations notamment SIMD mais on perd tout de même le caractère d'extensibilité. (61).

En fait une architecture cellulaire intégrant outil de traitement et outil de communication dans la même structure apparait comme l'interprétation extrême du principe qui voit en tout traitement de l'information qu'une succession de mouvements d'information à différents niveaux, le plus bas étant la manipulation du bit dans les circuits de l'U.C.

1.3.4 - Insensibilité aux pannes

A priori, une architecture cellulaire apparait comme très riche en possibilités du point de vue de l'insensibilité à d'éventuelles défaillances ponctuelles. Si elle est organisée sous forme de tableaux, la régularité extrême de la structure se prêtera facilement à des procédures de tests externes pour l'identification d'éventuels éléments défaillants. Par exemple dans le cas d'un maillage carré à 2 D, on peut envoyer un message de test de haut en bas, puis de gauche à droite. L'intersection de la ligne et colonne défaillante donnera l'élément en panne.

En dehors de ces possibilités liées à la structure, le seul fait de voir en une cellule un microordinateur complet lui permet d'assurer un rôle interne de surveillance. Elle peut détecter elle-même ses voisines en panne, les effacer de sa liste de correspondance et signaler la défaillance à l'extérieur. Une redondance certainement inévitable du fait de l'intégration en un seul outil des outils de traitement et de routage doit permettre une dégradation très progressive et lente des possibilités du système.

1.4 - MULTIPROCESSEUR, MULTIORDINATEUR ET CELLULARITE

Confrontons pour clore ce chapitre de présentation quelques architectures ou propositions d'architecture existantes et la cellularité.

1.4.1 - Machines tableaux (Array processors)

On peut citer Solomon (23) , Illiac IV (6) , DAP (54) , Anma (50) , Siemens (34) STARAN (57)

Ce ne sont pas des architectures cellulaires de microordinateurs par rapport à nos définitions précédentes. Elles apparaissent en fait constituées d'une unité centrale commandant à plusieurs "processeurs élémentaires" composés d'une A.L.U. et éventuellement de possibilités de stockage de données. La cellule n'est donc en aucun cas un ordinateur complet. Comme les processeurs élémentaires présentent une topologie de tableaux, on peut les désigner sous le vocable de "machines tableaux". Elles se rangent dans la catégorie des machines SIMD selon la classification de Flynn

Leurs domaines d'applications privilégiées sont tous les systèmes basés sur les équations aux différences partielles (relaxation). Elles sont couplées à une machine classique (B 6500 pour Illiac IV) à qui sont réservées les tâches administratives du pilotage macroscopique et des E/S.

Des marchés potentiels pour ce genre de machine sont des domaines tels la météorologie, la physique des plasmas, la programmation linéaire. Elles atteignent des performances inégalables pour les vastes opérations SIMD mais n'ont guère de possibilité pour des applications MIMD voire celles dont le parallélisme est peu élevé mais dont l'exploitation n'est cependant pas dénuée d'intérêt surtout si elles sont répétitives. C'est plutôt dans cette voie que nous nous engageons dans la suite de l'étude.

Notons enfin que D.A.P. apporte une notion intéressante que l'on retrouve notamment dans les propositions de Lipovski (40, 41).

Celle-ci est de considérer que sur le tableau des processeurs élémentaires est plaqué un tableau mémoire qui fait partie intégrante de la mémoire du calculateur hôte. Il y a donc d'une certaine façon adaptation de la mémoire au traitement et non l'inverse (4 K bits par élément d'un tableau 64 × 64).

1.4.2 - CHOPP (64)

C'est une proposition de machine MIMD supportant un O.S distribué, sans machine hôte, avec pour objectif d'associer 10^5 à 10^6 processeurs. On lie les processeurs selon la topologie d'un n-cube, c'est-à-dire que chaque processeur est placé au coin d'un cube à n-dimensions. Si on désigne un processeur comme origine et qu'on lui assigne une adresse binaire (0, 0,...,0), les autres processeurs sont repérés par un nombre binaire à n bits, où chaque position de bit est considéré comme une coordonnée pour chaque dimension. Deux processeurs immédiatement voisins ont des adresses qui ne diffèrent que d'un seul bit. La "distance" séparant deux processeurs est obtenue en faisant la somme bit à bit mod 2 des adresses et en comptant le nombre de bits à 1 (distance de Hamming). Si deux processeurs sont à une distance m, m ! chemins différents de longueur m les réunissent. Les communications se font par paquets. Un processeur possède n voisins : par exemple avec une configuration à 16 K processeurs, chaque processeur possède 14 voisins. Bien que fort proche de notre approche, la différence fondamentale est que le nombre de voisins pour un processeur varie en fonction du nombre total de processeurs employés. Il n'y a pas d'extensibilité au sens où nous l'avons défini.

1.4.3 - EGPA (8, 26, 28)

C'est une réelle architecture cellulaire conçue, d'après ses auteurs, dans le but de réaliser un système avec une flexibilité importante, c'est-à-dire capable de s'ajuster à la structure de l'application. Cet objectif est réalisé par une architecture en pyramide dont la figure donne un exemple. Chaque niveau ou tranche de la pyramide peut être considéré comme une organisation cellulaire. La microprogrammation des cellules exploitant cette structure permet d'appréhender les problèmes selon des modes d'opérations horizontaux (normal) ou verticaux (i.e. tel une machine associative du type STARAN. La configuration existante est cependant composée de grosses cellules (A.E.G. 80-60) beaucoup plus imposantes que celles que nous comptons envisager. Notons aussi que dans la configuration de la figure 1.11, cinq processeurs jouent le rôle d'un système moniteur pour seize processeurs de traitement. C'est peut-être beaucoup, bien que si on passe à des dimensions plus grandes, le rapport entre processeurs actifs et processeurs systèmes devient beaucoup plus favorable. Il reste que l'on envisagera de plaquer le parallélisme d'une application donnée dans un tableau cellulaire dans des cadres plus restreints et plus simples d'emploi que la solution proposée par EGPA.

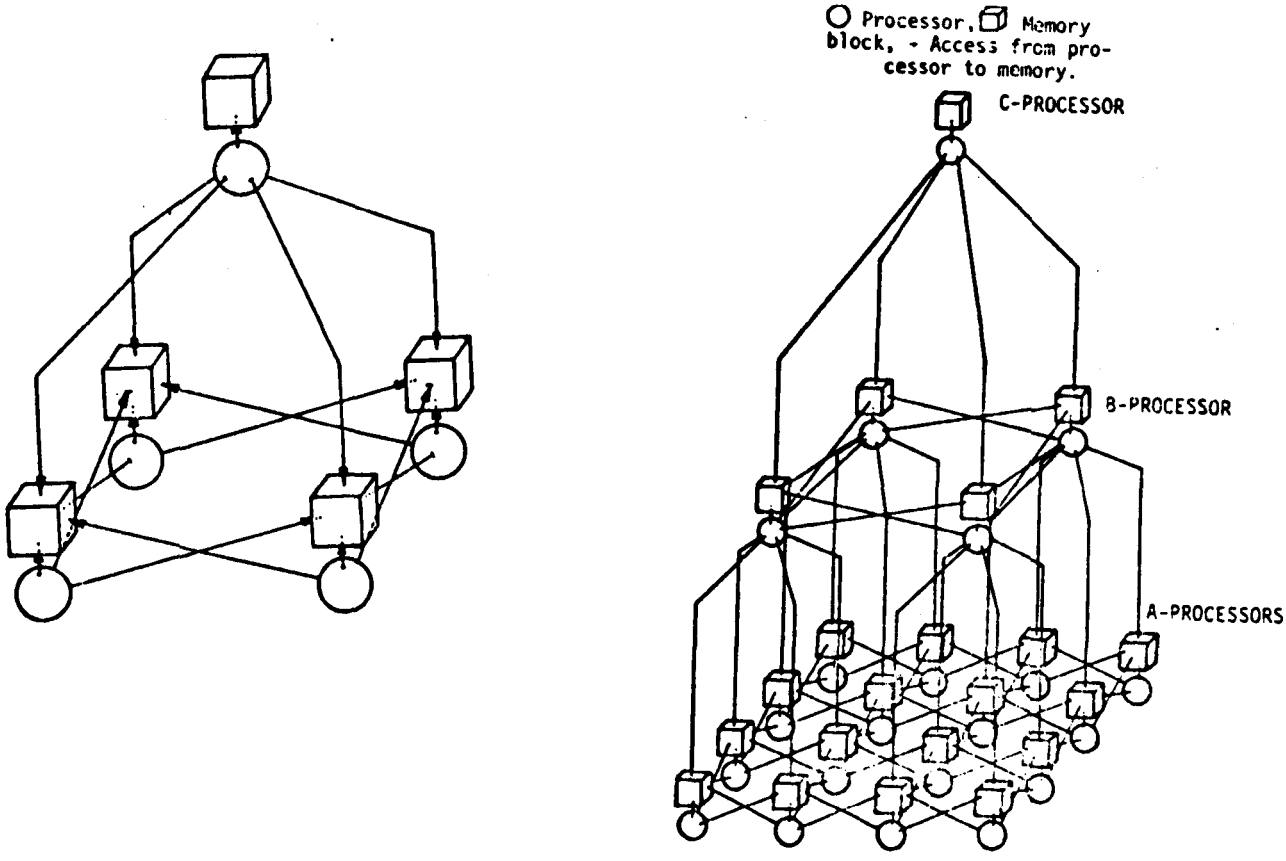


Fig 1.11 La pyramide EGPA (à 5 ou 21 processeurs)

1.5 - Autres propositions

Il s'agit de propositions et non de réalisations effectives.

1.5.1 . - PROPOSITION DE LIPOVSKI (40, 41)

Il s'agit d'un tableau de microprocesseur à structure variable. Il le baptise de cellulaire mais en fait la cellularité est moins stricte que celle de notre définition et apparait plus comme une structure logique aisément manipulable que comme la véritable base de l'architecture. Il envisage en effet de connecter les processeurs par un réseau de commutation par paquet (P.S.N). Partant du principe (mais sera-t-il toujours vrai ?) que le coût de la mémoire domine celui du traitement, il suggère une architecture qui partagerait efficacement l'espace mémoire entre les tâches. Une cellule n'est plus une entité indivisible mais un couple unité de contrôle et traitement/unité de mémoire qui peut être scindé. Une unité de contrôle peut utiliser les unités de mémoire d'un certain nombre de voisins dans une sous-structure appellée "espace de tâche". Une tâche se définit alors comme une subroutine utilisant une même structure de donnée, pouvant en fait être tant un programme entier qu'une seule instruction. Ce sont bien sûr deux extrêmes. L'espace mémoire est divisé en pages. Les pages de données et d'instructions sont séparées. Ceci permet dans le cas de traitement S.I.M.D qu'un seul processeur décode les instructions pour un espace de tâche donné alors que tous exécuteront des données particulières. Cela signifie qu'à l'intérieur d'un cycle lecture-décodage instruction/exécution, des liens entre cellules s'établissent dans la première phase et se masquent dans la seconde. Ainsi les pages d'instructions sont disséminées dans tout l'espace de tâche et on peut optimiser l'occupation de la mémoire. Si on ajoute que les retenues et dépassements peuvent se répercuter entre A.L.U de cellules voisines, (emploi de processeurs en tranche) la longueur des mots sur lesquels on travaille est variable, adaptable à la demande. Ce serait donc une structure très riche mais il n'y a pas de proposition pour la gestion globale de cet espace et il importe de noter que les performances de ce dernier dépendent étroitement de l'outil de communication choisie.

1.5.2. - ARCHITECTURES "ARBRE"

On peut les considérer comme des cas particuliers d'architectures cellulaires. Un exemple en est le X-tree de **Berkeley** (19,51,59)
C'est une structure arborescente normale à laquelle on ajoute des chemins supplémentaires pour raccourcir les temps de communication, augmenter la tolérance aux pannes et uniformiser le trafic.

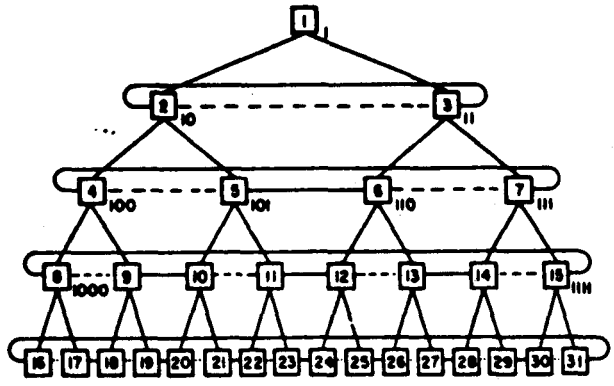


Fig. : 1.12 - Le X-tree.

En effet, il faux noter que dans une architecture arbre simple, Ce sont les processeurs du niveau 2 qui assurent le plus gros trafic. La fonction de routage est donc mal répartie (19). C'est donc une des raisons des chemins supplémentaires du X-tree. On peut aussi considérer cette approche comme une méthodologie d'acquisition d'une certaine topologie donnant naissance à un réseau cellulaire qui n'aurait d'arbre que le nom.

1.5.3 - MU.NET

Développé au "laboratory for computer science" du M.I.T, le MU.NET est dans sa définition la plus large une architecture multiprocesseur où chaque processeur communique directement avec un nombre limité de voisins. Ce qui entre tout-à-fait dans notre cadre. Actuellement (25) une maquette bâtie

autour de plusieurs LSI 11 est expérimentée mais en fait l'étude, selon les propos mêmes de ses auteurs, ne fait que commencer. Il apparait d'ailleurs que le principal but visé est surtout le développement de méthodologie software et des principes d'organisation pour utiliser de tels réseaux, au pluriel car aucune topologie n'est avancée en préférence à d'autres.

CHAPITRE 2

ARCHITECTURES SPECIALISEES

Nous allons d'abord envisager la construction de tableaux cellulaires spécialisés, voués à la réalisation de quelques tâches précises mais paramétrés de façon à garder une certaine souplesse à l'intérieur d'un cadre prédéfini.

2.1 - SCHEMA DE BASE

Indépendamment de la nature des traitements à effectuer, notons qu'il existe deux façons de concevoir les échanges entre cellules voisines :

α) *Echange synchrone* : Il y a un cycle unique (préhension de l'information, traitement, commande) commun pour tout le réseau et donc à priori une seule horloge (plusieurs sous-cycles suivant les diverses directions si l'espace cellulaire est anisotrope).

β) *Echange asynchrone* : Les mécanismes d'échange sont plus complexes car il faut mettre en place un protocole qui permette les communications entre voisins sans risque d'interblocage. Ce peut être une forme mieux adaptée à certains traitements. Par exemple une cellule ne réveille ses voisins que dans certains cas pour leur signaler un changement d'état ou demander une information. Il faut aussi définir le graphe de communication. Pour cela is s'agira d'abord de connaître le nombre de voisins par cellule et d'organiser les cellules en respectant la topologie de ce qu'il serait commode d'appeler "l'espace utilisateur", ce dernier se définissant comme la représentation symbolique de l'ensemble des objets abstraits ou réels sur lequel travaille l'utilisateur de la machine ou déduit de l'organisation théorique qu'a établi le concepteur en vue de réaliser un traitement donné.

Deux options sont possibles pour réaliser cela :

1° Le tableau est cablé de façon à recopier l'espace utilisateur : pas de reconfiguration possible mais c'est la solution la plus économique quant au matériel employé et la plus rapide.

2° Les cellules ou des sous-ensembles de cellules sont reliés entre eux par un outil de communication qui peut réaliser des configurationsdifférentes (37 , 52) . Un tel ensemble, cellules-outil de communication est apte à générer plusieurs tableaux cellulaires "virtuels" pour autant d'applications, ne serait-ce que la mise au point d'un tableau du premier type.

γ) *Relation avec l'extérieur* : Il y a trois grandes possibilités, chacune possédant quelques variantes, d'échanger de l'information entre un espace cellulaire et l'extérieur.

- Accès par les bords, entrée et sortie différentes (une variante est que certaines cellules soient considérées comme des points d'entrée et d'autres des points de sortie).

- Accès par les bords, entrée-sortie confondues.

- Accès direct dans chaque cellule.

2.1.1 - Relation utilisateur-concepteur

Si nous ne possédons pas l'outil de communication capable de réaliser une topographie à la demande, tous les éléments constitutifs du tableau cellulaire doivent être précisés à partir des désiratas du futur utilisateur d'un tableau si celui-ci est voué à une application précise. L'utilisateur (qui peut ne faire qu'un avec le concepteur-constructeur) va donc "quantifier" son problème et le constructeur, indépendamment du contenu sémantique de celui-ci va y puiser certains critères qui sont :

- nombre de voisins par cellule.

- nature des relations entre voisins.

- nature des traitements internes, définir le rapport volume traitement : volume communication.

- isotropie ?

- uniformité ? (sinon il faut définir plusieurs "types" de cellules, l'hétérogénéité fonctionnelle s'opposant à l'homogénéité du matériel).

A partir de ces données, il est possible de définir l'architecture du tableau, c'est-à-dire le type de processeurs à utiliser, les tailles mémoire microprogramme (ROM) et mémoire en travail (RAM, registres "Bloc-note" etc.), nature des liens entre voisins (Bus, registres tampons, liaison série) ou //) et avec l'extérieur.

2.1.2 - Squelette d'une cellule

Un moyen simple de réaliser les transferts d'information entre cellules est de considérer que chaque cellule atteint ses voisins par l'intermédiaire d'un ou plusieurs registres en accès direct où elle est toujours seule à écrire pendant la phase de préhension de l'information (mode synchrone) ou après avoir prévenu le destinataire (mode asynchrone). L'unité centrale va ensuite chercher ces informations selon le programme qu'elle déroule. La communication avec l'extérieur, si elle existe en dehors des cellules périphériques, pourrait être établie sur le même modèle.

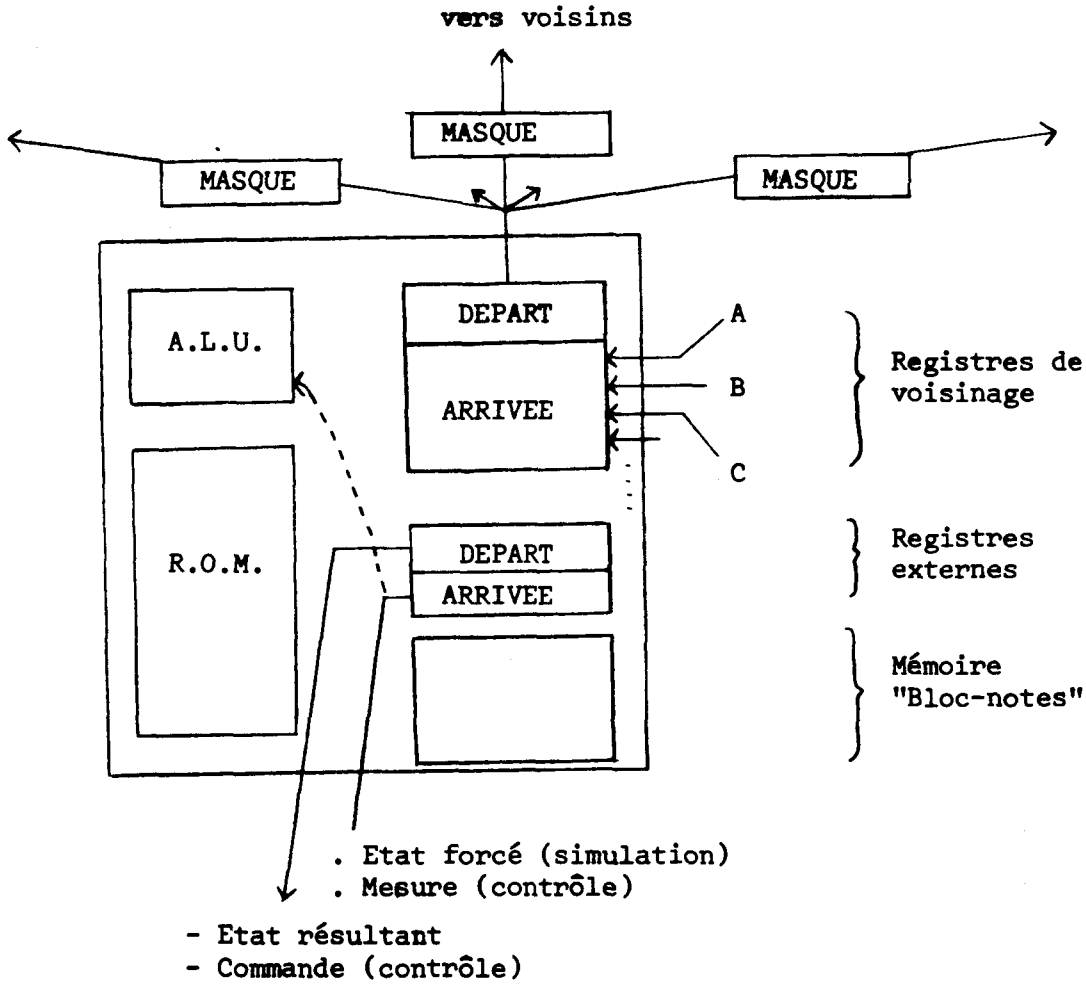


Figure 1 : Squelette d'une cellule type.

Un exemple de réalisation (bien que réalisé en éléments discrets) est le P.C.C. développé ces dernières années à I.E.E.A. (15,17, 18).

Pour illustrer notre propos, nous allons développer deux exemples d'applications où le concept de "cellularité" apparaît comme une voie intéressante, sans prétendre qu'elle soit la meilleure au point de vue des performances, pour la conception de machines spécialisées : D.D.A. (Digital Différential Analyser) et prétraitement de tableaux de données avant leur analyse. Nous les avons choisis car ils se situent dans des contextes différents tout en faisant référence à des domaines déjà largement travaillés (38,63)

Nous ne pensons pas que ce soit là les seules voies à explorer mais il faudrait une collaboration entre informaticiens et gens de l'art pour apprécier les possibilités réelles de réalisation d'une architecture pour un domaine d'activité donné. De nombreuses simulations sont basées sur une approche cellulaire (22, 60) mais il est difficile d'apprécier l'intérêt d'une implantation matérielle. Il est certain néanmoins que si le temps, sous quelque forme que ce soit, joue un rôle important sinon primordial pour une application de type cellulaire donnée, l'emploi de tels architectures devient envisageable. On peut espérer aborder ainsi des domaines qui n'avaient pas été touchés par l'informatique ou du moins d'une façon encore très perfectible.

Prenons l'exemple d'un tableau cellulaire qui modélise un certain espace utilisateur. Si ce modèle est bon et qu'il existe un sous-ensemble de points observables dans l'espace utilisateur, le tableau cellulaire pourra suivre l'évolution de l'espace utilisateur, et le plus souvent anticiper sur l'état futur de ce dernier. Cela peut servir d'aide à la décision, humaine ou intégrée dans un système gouverné par des méthodes heuristiques. Par exemple on pourrait assister au développement d'un "contrôle flou" de l'espace utilisateur par le tableau cellulaire dans les cas où on ne peut trouver d'algorithme précis pour remédier à une situation donnée.

2.2 - TABLEAUX CELLULAIRES POUR LE PRETRAITEMENT DE TABLEAUX DE DONNEE

2.2.1 - Introduction

On explore ici les possibilités offertes par un tableau cellulaire de microprocesseurs pour le prétraitement et le stockage de données dont l'organisation préférentielle est un tableau et où le singleton (une seule donnée), la ligne ou la colonne sont des entités susceptibles d'être manipulées comme formant un tout selon les désiratas de l'utilisateur. On retrouve ainsi la dualité de Kautz (32) entre tableau mémoire et tableau logique ou de traitement. Par exemple en analyse de données, si les indices de ligne correspondent à des variables et les indices de colonne à une observation, on peut vouloir manipuler séparément lignes ou colonnes pour obtenir la valeur moyenne d'une variable (opération ligne) ou l'écart moyen d'une observation donnée (opération colonne).

Quatre comportements sont privilégiés :

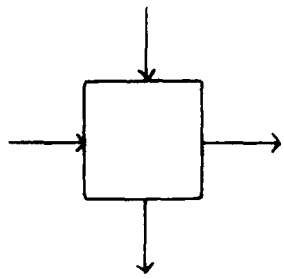
- Opérations sur un singleton : un traitement est appliqué à chaque élément du tableau (s'il est identique et appliqué simultanément retrouver un pseudo. SIMD).

- Opérations de type cellulaire : un traitement sur une cellule ne tient compte que de l'élément stocké et de ses voisins immédiats.

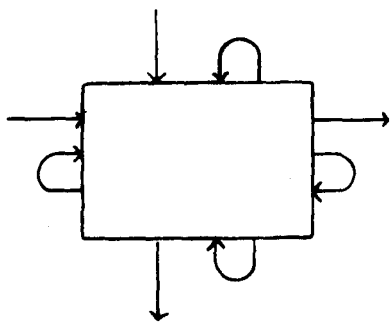
- Opérations entre deux colonnes ou entre deux lignes élément à élément.

- Opérations concernant tous les éléments d'une même ligne ou d'une même colonne.

Si on néglige les opérations de type cellulaire, on peut s'écarter du schéma initial pour simplifier l'architecture d'une cellule et envisager une cellule à deux entrées, deux sorties.



Ce qui nous donne le schéma de principe suivant pour le tableau :

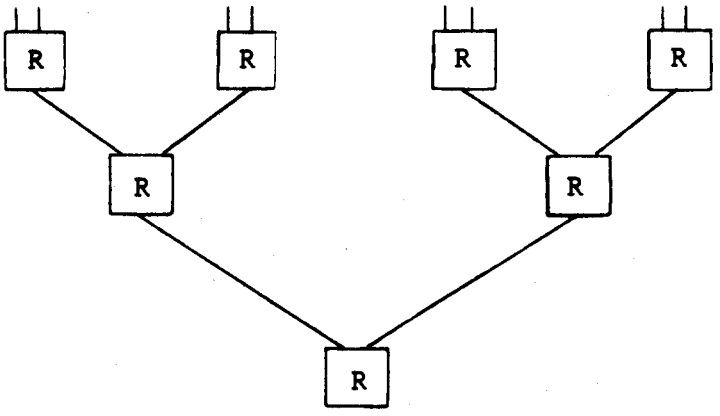


La clôture torroïdale du tableau permet à une cellule de rang supérieur à une autre de toucher éventuellement cette dernière sans avoir recours à des chemins de données bi-directionnels ce qui alourdirait le logiciel de service d'une cellule.

Un objectif final de cette étude est l'emploi de microordinateurs monochips dont les précurseurs actuels sont par exemple le Texas 9 940 (67) 16 Bits, 2KO Rom, 128 octets RAM) mais dont les successeurs auront des possibilités de mémorisation beaucoup plus élevées

L'accroissement du nombre de connexions par puce pourrait amener des facilités d'implantation pour des architectures cellulaires (registres E/S intégré dans la puce et accessible de l'intérieur sans occupation du bus interne par exemple). En fonction des données et des traitements associés le processeurs est choisi : par exemple le 9 940 possède multiplication et addition cablées alors que des opérateurs flottants très fortement intégrés voient le jour (29).

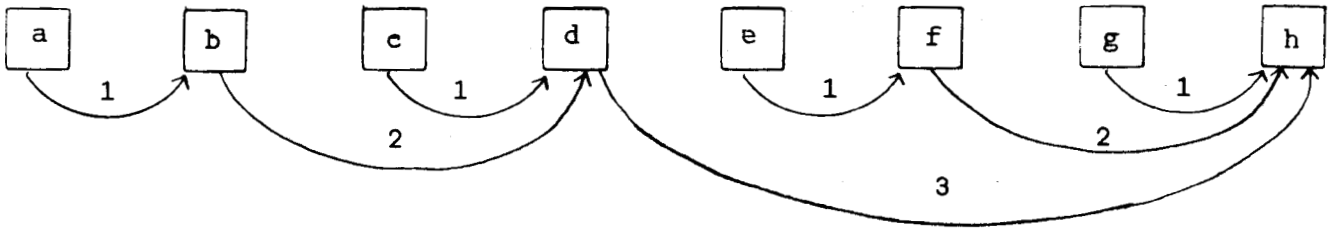
On envisage alors de charger un tableau cellulaire d'effectuer des opérations du type moyenne arithmétique, géométrique, détection de minima tout traitement qui nécessite des opérations élémentaires de façon répétitives, opérations de surcroit souvent associatives (au sens algébrique du terme) (a R b) Rc = a R (b R c) et donc exécutables selon une arborescence binaire en $\log_2$ étapes.



2.2.2 - Opérations associatives ligne ou colonne

Supposons que nous ayons rangé ligne par ligne les données, une ligne comprenant p données avec une donnée par cellule (une cellule de traitement = une cellule de mémoire). On veut réaliser une opération associative du type précédent sur les éléments de la ligne :

Exemple



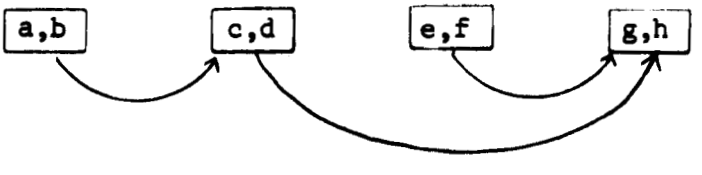
Si θ est le temps d'exécution de R.

Si τ est le temps de transmission d'un opérande d'une cellule à une autre immédiatement voisine, le temps d'exécution sera :

$$T_{XQ} = \theta \cdot \log_2 p + (p - 1)\,\tau \qquad (1)$$

Il est évident que la méthode de rangement qui fait correspondre une cellule mémoire à une cellule de traitement n'est pas optimal dès que τ n'est pas négligeable devant $\frac{\theta}{p} \cdot \log_2 p$.

La méthode suivante est déjà bien meilleure

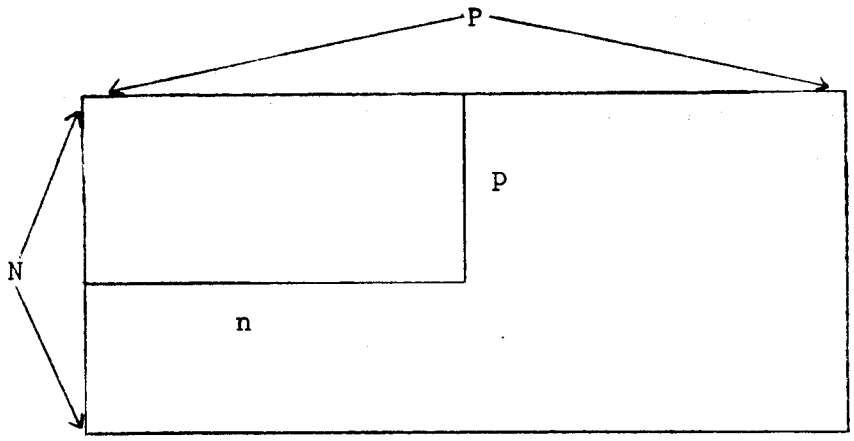


$$T_{XQ} = \theta \cdot \log_2 p + (\frac{p}{2} - 1)\,\tau. \qquad (2)$$

En faisant ce rangement, on a compressé le tableau de données dans le tableau cellulaire avec un rapport deux. On va fouiller de suite un peu plus avant cette notion de compression.

2.2.3 - Compression de tableaux de données

Quelque soient les moyens dont on dispose, les dimensions de la machine cellulaire risque souvent d'être très en deça de celles des tableaux de données. Il est donc important que cette machine cellulaire puisse émuler des tableaux de toute taille.



Soit $N \times P$ les dimensions d'un tableau de données.

Soit $n \times p$ les dimensions du tableau cellulaire.

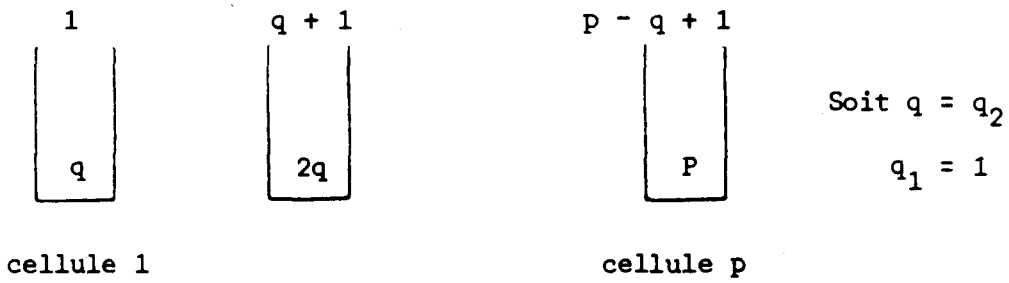
Soit q le facteur de compression égal à

$$\frac{N \times P}{n \times p}$$

c'est-à-dire si $N = q_1\, n$, $P = q_2\, p$, $q = q_1 \cdot q_2$

Dans une cellule on va ranger q éléments de $T(N \times P)$ et en quelque sorte une cellule remplace un tableau $(\frac{N}{n}, \frac{P}{p})$ possédant q éléments.

Revenons à notre traitement associatif ligne.



A l'intérieur d'une cellule, si θ est le temps pris pour exécuter R

$$T_{XQ} = (q - 1)\ \theta \qquad (3)$$

On utilise ensuite ces résultats partiels selon la procédure vue au paragraphe précédent :

$$T_{XQ} = (q - 1)\ \theta + \theta \cdot \log_2 p + (p - 1)\ \tau.$$

$$T_{XQ} = [(q - 1) + \log_2 p)]\ \theta + (p - 1)\ \tau. \qquad (4)$$

Si q = 2 on retrouve (2).

On peut donc comparer cette solution avec celle utilisant un monoprocesseur puis celle ou une cellule est liée à un seul élément du tableau (T_N).

$$T_S = (pq - 1)\ \theta \qquad (5)$$

$$T_N = (\log_2 p + \log_2 q)\ \theta + (pq - 1)\ \tau \qquad (6)$$

$$T_C = [(q - 1) + \log_2 p]\ \theta + (p - 1)\ \tau \qquad (7)$$

Comparons (4) et (6). On peut déjà noter que si

$a = \frac{\theta}{\tau} > 1$, $T_N - T_C > 0$ dès que $p \geq a$

En effet $T_N - T_C = \underbrace{\theta \cdot \log_2 q}_{> 0\ \forall q} + \underbrace{(\frac{p}{a} - 1)\ (q - 1)}_{\geq 0}$

Si donc on connait $a = \frac{\theta}{\tau} > 1$, on sait déjà qu'une solution compressée quelconque telle que $p > a$ est plus rentable au point de vue temps d'exécution qu'une solution éclatée.

2.2.4 - Détermination des dimensions optimales

La figure 2-II donne les courbes de variation en fonction de p variant de 1 à N de T_C, N étant constant et supposé égal à 10.000, θ étant fixé à 100, $a = \frac{\theta}{\tau}$ variant de 0,1 à 100 et paramétrant ces courbes.

Par exemple, dans une solution avec registre tampon et se basant sur le T 9 940 a = 1 pour une multiplication et 0,1 pour une addition.

On peut y voir de suite l'intérêt de travailler avec un p près du p optimal.

Soit $N = pq$

(6) peut s'écrire avec $q = \frac{N}{p}$,

$$T_C = [(\frac{N}{p} - 1) + \log_2 p]\,\theta + (p - 1)\,\frac{\theta}{a}$$

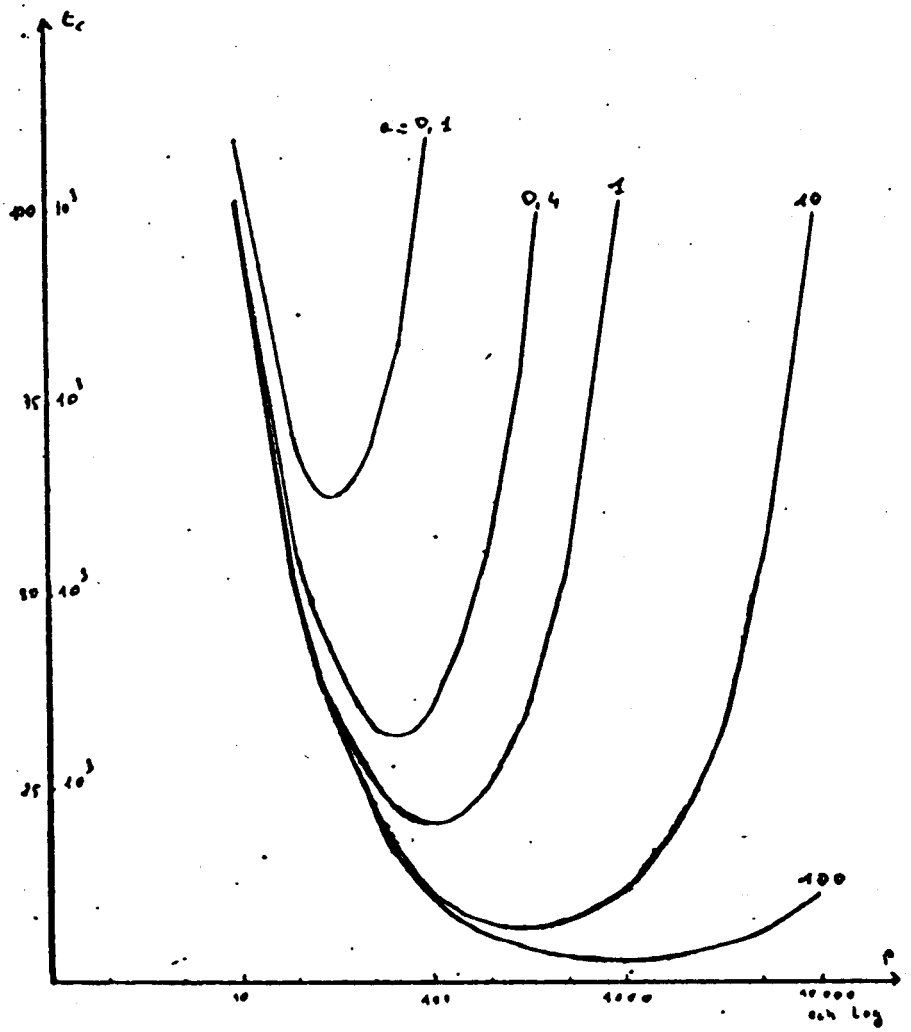
$$T_C = \theta \cdot \frac{N}{p} + \theta \cdot \log_2 p + \frac{\theta}{a}\,p + K \qquad (8)$$

$$T'_C = \frac{\theta \cdot N}{p^2} + \frac{\theta}{p} + \frac{\theta}{a}$$

P o pt est la valeur de p pour laquelle cette dernière s'annule.

$$P \circ pt = \frac{-\theta + \sqrt{\theta^2 + \frac{4\theta^2 N}{a}}}{2 \cdot \frac{\theta}{a}}$$

$$\boxed{P \circ pt = \frac{a}{2}\left(\sqrt{1 + \frac{4N}{a}} - 1\right)} \qquad (9)$$



BUS LILLE

Fig. : 2-II TC = f(P)

Si $\frac{N}{a} >> 1$

$P \circ pt \sim \sqrt{a \cdot N}$

Il faut noter que cette valeur ne dépend que de a et N.

Pour a = 1 et N = 10.000, $P \circ pt \simeq 100$

Les tableaux suivants comparent les trois solutions correspondant aux formules (5), (6) et (7).

Solution bijective

a	0,1	0,4	1	10	100	1000
T_B	10.000.329	2.501.080	1.001.230	101.319	11.328	2.328,7
$R = \frac{T_B}{T_C}$	158	76,6	49	14,4	3,9	
$Q = \frac{T_A}{T_C}$	15,8	30,6	49	142	346	les gains sont à multiplier par le le nombre de lignes traitées simultanément
$Q/R = \frac{T_A}{T_B}$	0,1	0,4	1	9,86	88	

Tableau 2-I.

On voit que même pour des temps de transfert élémentaires supérieurs au temps de calcul élémentaire, une solution cellulaire compressée est beaucoup plus rapide qu'une solution monoprocesseurs à temps CPU égal ce qui est bien sûr une vue de l'esprit.

Par ailleurs la comparaison entre le temps d'exécution sur la machine hôte ($T'_A = (pq - 1)\ \theta'$ avec $\theta' << \theta$ en tenant compte de la vitesse de la machine hôte) et le traitement ligne sur tableau cellulaire serait aussi fallacieuse. Il est plus juste de faire intervenir les coûts respectivement de deux options :

On peut définir par exemple :

$$\text{coût/s} = \frac{\text{Prix achat + fonctionnement (maintenance)}}{\text{temps total utilisation}}$$

Si C est le coût/s de la machine hôte et C celui du tableau cellulaire, le rapport :

$$\frac{T'_A \cdot c}{T_C \cdot C}$$

est beaucoup plus significatif que les précédents.

Si par exemple, un centre de calcul a de gros traitements du type analyse de donnée, il va consacrer un gros pourcentage du temps CPU à des travaux annexes que le tableau cellulaire aurait pu effectuer au moins aussi rapidement, l'amortissement de ce dernier étant obtenu par le gain d'heures CPU disponibles pour d'autres traitements. (1 monochip = ordre de 100 F pièce ⟹ Prix tableau cellulaire = un tableau 20 × 5 = 10^4 francs de composants nobles + prix circuiterie et composants annexes + prix étude + prix fabrication).

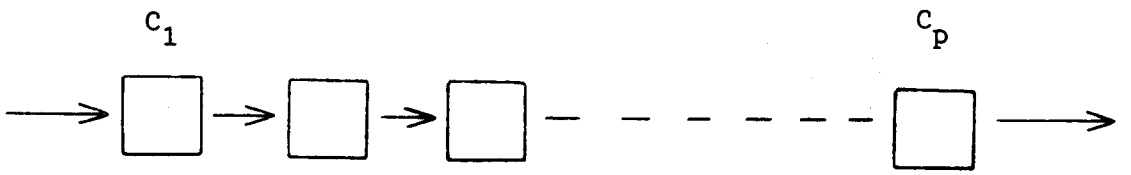
Il n'est pas bien sûr évident que la balance penche actuellement pour une solution de ce type mais l'évolution des VLSI et l'arrivée de composants de technologie MOS possédant des opérations flottantes cablées peut rapidement modifier ce point de vue dans les prochaines années pour peu qu'ils soient produits en grande série.

* Si chaque cellule peut stocker 128 mots ⟶ tableaux de 2560 × 640 peuvent être stockées.

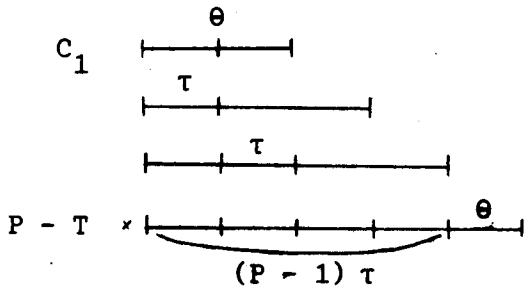
2.2.5 - Autres éléments intervenant dans le choix des dimensions

Même si les traitements de type associatifs seront une part importante du travail du tableau cellulaire que nous étudions, il faut s'occuper des cas où on distribue une même donnée à tous les éléments d'une ligne ou d'une colonne en vue d'un traitement interne à la cellule. On peut penser en particulier à une opération de normalisation des données.

Soit donc notre cascade de cellules.



On injecte à C_1 une donnée que C_1 duplique, envoie à son successeur et utilise pour un traitement donné.



Soit θ le temps d'exécution.
Soit τ le temps de la retransmission d'un message élémentaire d'une cellule à une autre.

Le temps T d'exécution total, c'est-à-dire l'instant écoulé entre l'injection en C_1 de la donnée initiale et la fin du traitement de cette donnée dans C_p est égal à $(P - 1)\ \tau + \theta$(9) dans une cascade de type B.

Si on a compressé les données internes.

$$T = (p - 1)\ \tau + q \cdot \theta \qquad (10)$$

A N constant.

(9) s'écrit : $((N - 1)\ \frac{1}{a} + 1)\ \theta$

(10) s'écrit : $((p - 1)\ \frac{1}{a} + \frac{N}{p})\ \theta$

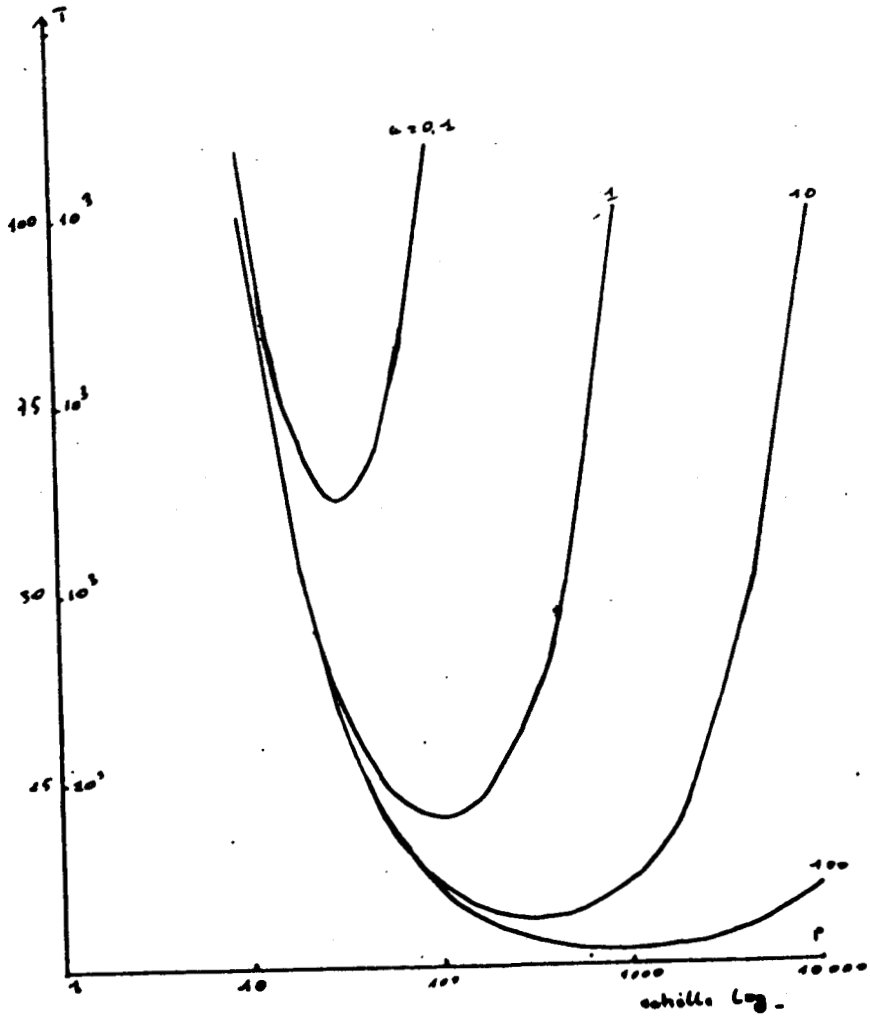


FIG. 2. III T = f(P).

LILLE

La figure 2-III donne les variations de (10) pour diverses valeurs de a.

Le tableau 2-III donne les valeurs de (10) pour le p optimal déterminé auparavant :

a	0,1	1	10	100
P o pt	31	100	316	1 000
T_C (10)	62.240	19.900	6.314	2.019
T_C (9)	9.999.100	1.000.000	100.090	10.099
$H = \frac{T_B}{T_C}$	161	50,25	15,8	5

Tableau 2-II.

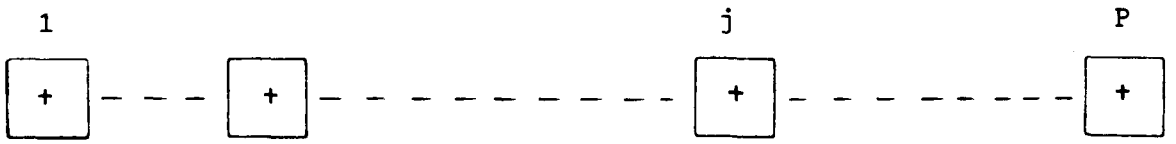
Ici encore le P o pt est meilleur qu'une solution éclatée (non compressée).

Il était important de voir ce comportement car c'est celui qui va régler à la fois le changement des données initiales et le routage des ordres définissant le traitement d'une cellule. L'opération devient alors ou l'exécution d'une routine de chargement ou d'une routine qui va initialiser la cellule pour une fonction particulière. La nature du problème nous impose des solutions simples.

2.2.6 - Enchaînement des opérations élémentaires

Il est intéressant de remarquer que pour un traitement associatif ligne ou colonne, on a nul besoin d'une synchronisation proprement dite pour les cellules d'une même cascade.

En effet, une cellule peut connaître à partir de sa position dans la cascade (paramètres internes) les opérations élémentaires (calcul, transfert qu'elle doit effectuer. Si on ajoute à cela le caractère anisotrope de la transmission des messages et le fait qu'on ne peut écraser les messages à l'entrée d'une cellule, le $i^{ème}$ message reçu sera le $i^{ème}$ message attendu sans aucune ambiguité possible (on s'interdit bien sûr l'injection de messages externes pendant l'exécution d'une tâche).



$j = 2^y \cdot A$ A n'étant pas divisible par 2.

La cellule aura y opérations à effectuer.

Une opération R à l'intérieur d'une cellule sous-entend l'enchainement suivant :

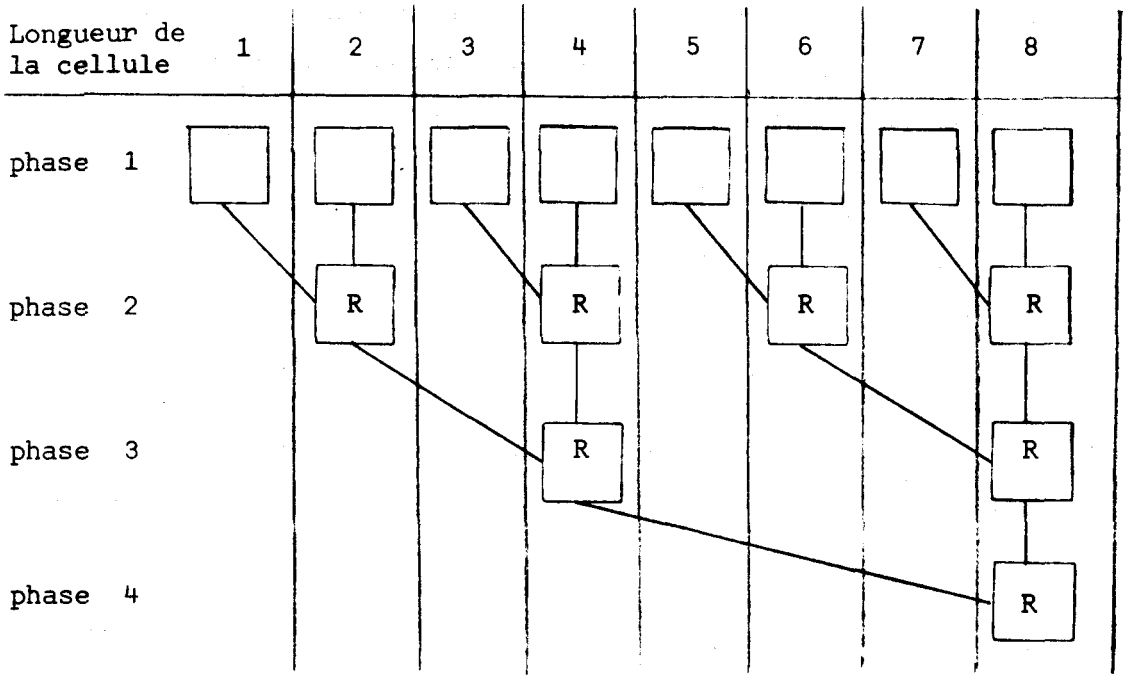
- Si le tableau est compressé, traiter les P/q éléments en mémoire.

E.T.I. - Si y = 0 passer le résultat à droite et se libérer sinon attendre un opérande, le traiter avec celui calculé précédemment.

- Décrémenter y et revenir à E.T.I.

Les opérations sur les singletons et ligne à ligne ou colonne à colonne élément par élément par élément peuvent se faire aussi en synchrone avec des composants spécialisés permettant des configurations SIMD pour d'autres utilisations.

La figure 2-II explique cette propriété intéressante :



cellule 8 a $\log_2 8 = 3$ opérations R à effectuer

cellules impaires = aucune

cellule 4 = 2 opérations

cellule 2, 6 = 1 opération.

S'il y a compression de facteurs q :

Une cellule aura : ((q - 1) + y) opérations à effectuer.

2.2.7 - Logiciel d'interprétation

Le travail de l'espace cellulaire peut se schématiser dans une hiérarchie à trois niveaux. Au niveau le plus bas, on trouve les opérations élémentaires, c'est-à-dire les opérations arithmétiques ou logiques et leurs diverses options (signe, flottant, simple/double précision) et les opérations de routage (passage d'un message sans exécution, éventuellement routage dirigé type STUD) (17). Au niveau juste supérieur, on trouve des tâches de complexité moyenne qui consistent comme on l'a vu au paragraphe précédent en la concaténation d'un certain nombre d'éléments de l'ensemble précédent. Ce serait par

exemple SOM (origine, taille tab de données) qui est la somme de tous les éléments de la ligne i. Coiffant ces deux premiers niveaux on trouve des tâches complexes composées d'un certain nombre des précédents

Exemple : normalisation d'une variable :

(1) Somme de tous les éléments d'une ligne (pour chaque ligne) (opération ligne).

(2) Division dans la dernière cellule de chaque ligne du résultat obtenu par P, nombre d'éléments dans une ligne. (opération singleton).

(3) Retransmission de la moyenne obtenue à tous éléments de la ligne. (transfert).

(4) Division dans chaque cellule de l'élément stocké par la valeur moyenne obtenue précédemment. (opération singleton).

On voit fort bien que cette opération de normalisation pourrait être une partie d'un traitement encore plus complexe. Nous classons dans le niveau III ce dernier dès qu'il dépasse la complexité d'une opération ligne ou colonne, tout comme le précédent, le niveau III désignant ainsi des opérations interprêtables "globalement" par la machine hôte comme nous allons le voir.

2.2.8 - Implantation des fonctions de service dans l'espace cellulaire

Le premier niveau d'exécution est implanté dans l'ensemble des cellules. C'est impératif pour bénéficier des gains en temps d'exécution dont nous avons discuté au paragraphe précédent. Remarquons de suite qu'à un niveau donné d'exécution, on peut faire correspondre un niveau d'interprétation chargé de faire la liaison entre celui-ci et le niveau supérieur. Il est élémentaire dans ce cas car il s'agit au plus du décodage de quelques instructions

niveau assembleur. Par contre l'implantation des deux niveaux suivants nous amène à un éventail assez large de solutions. On peut considérer ici le tableau cellulaire comme un périphérique lourd. Le problème devient alors de savoir comment répartir les deux niveaux supérieurs entre la machine hôte et l'espace cellulaire. Cela dépend pour une grosse part des possibilités internes d'une cellule. Par exemple on remarquera dans le cas du D.D.A. que la cellule choisie, en dehors des opérations élémentaires nécessaires pour assumer la fonction D.D.A., n'a que des possibilités très limitées du fait de l'absence de possibilité d'interruption, de mécanismes internes permettant l'appel de sous-programmes et du nombre restreint de registres de travail. Dans le cas présent, nous nous orientons d'emblée vers des processeurs qui sont de réels microordinateurs intégrés sur une puce et travaillant de préférence sur 16 bits (ex. T 9 940). Les limitations possibles viennent alors principalement des tailles des mémoires que l'on peut mettre dans une cellule et conséquemment du prix de revient voulu pour une cellule.

On envisage rapidement trois solutions.

1ère solution

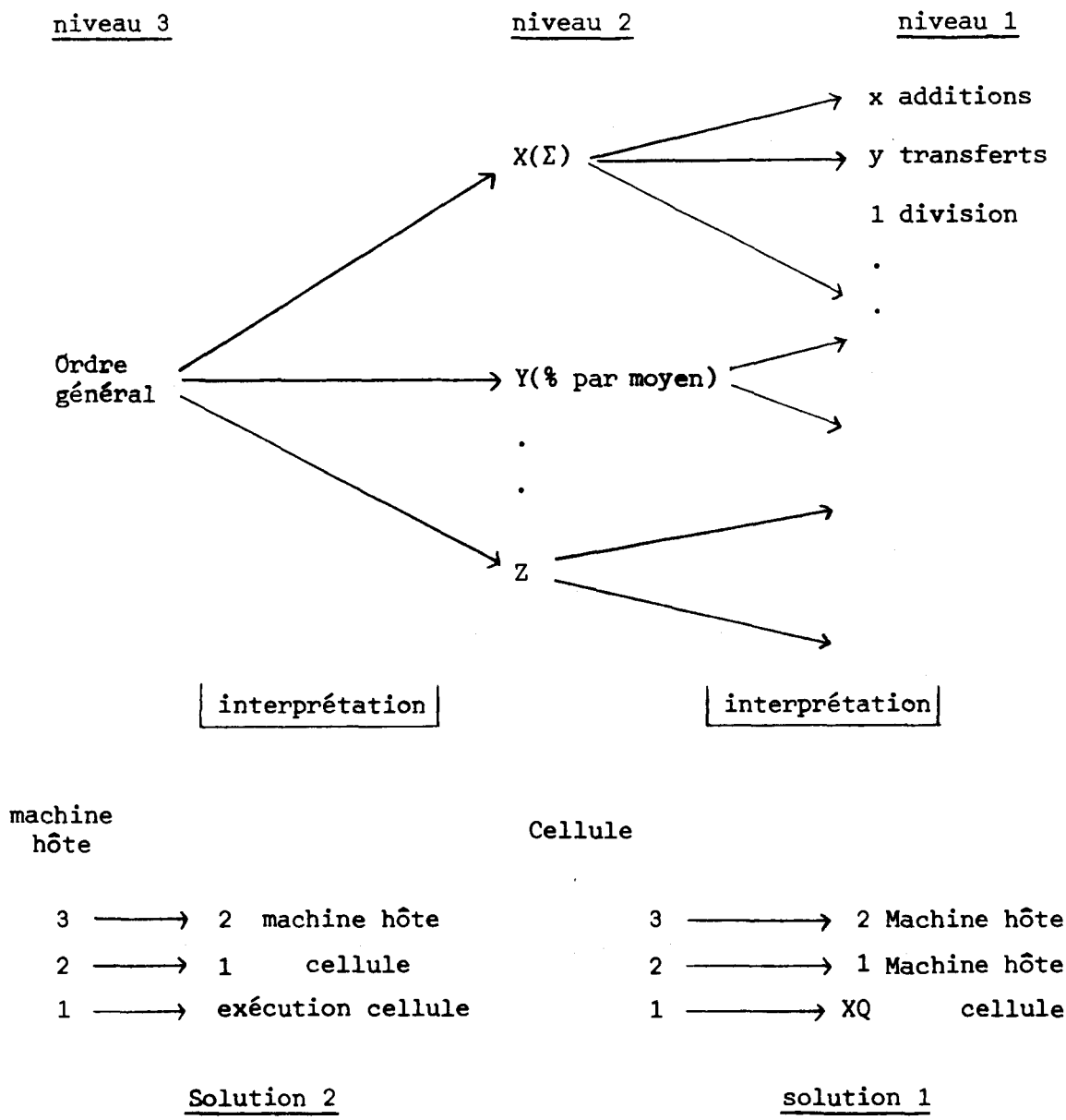
Seules les opérations élémentaires (niveau I) résident dans les cellules.

Elles sont commandées par la machine hôte qui gère la totalité des deux niveaux supérieurs. C'est une solution que l'on a pour mémoire car elle apparait nettement en contradiction avec un de nos principaux objectifs définis dans l'introduction qui est de réduire autant que faire se peut le rôle d'un contrôle externe.

2ème solution

Compte-tenu de ce qui a été montrer dans le paragraphe 2.2.6, il est très facile de ramener à l'intérieur d'une cellule le deuxième niveau

d'interprétation du traitement. Une cellule acquiert donc un message d'ordre dont les paramètres sont globaux, c'est-à-dire identiques pour toutes les cellules (origine du tableau de données dans le tableau cellulaire, taille du tableau de données, mode (ligne ou colonne, cellulaire), facteur de compression). Un programme de service interne à toutes les cellules dépouille ce message et génère une suite d'opérations élémentaires dont le nombre et l'ordre sont déduits à la fois des paramètres globaux et de la position de la cellule dans le tableau (paramètre interne). L'enchaînement de ces traitements du niveau 2 est réalisé par la machine-hôte. Cette dernière se trouve donc déchargée de calculs dépendant en partie de paramètres internes à chaque cellule ce qui laissait présager une longue série d'exécution de la routine d'interprétation 2.



L'intérêt était donc grand de répartir la tâche de passage du niveau 2 au niveau 1 entre toutes les cellules.

La machine hôte reçoit les fins d'exécution des tâches du niveau 2 et fait l'enchaînement en lançant les cellules à partir de messages conventionnels paramétrés.

3ème solution

$$\left.\begin{array}{l} 3 \longrightarrow 2 \\ 2 \longrightarrow 1 \\ 1 \longrightarrow XQ \end{array}\right\} \quad \text{cellule}$$

Les cellules reçoivent des ordres globaux à partir desquelles elles sont capables de traduire l'enchaînement des tâches du niveau 2 et à fortiori celles du niveau 1. L'intérêt est que la machine-hôte voit son rôle réduit à lancer des messages d'ordre globaux. Cependant la routine d'interprétation des ordres du niveau 3 en tâches du niveau 2 est globale, c'est-à-dire qu'elle ne fait intervenir aucun paramètre interne aux cellules. L'intérêt de distribuer cette tâche est donc moindre. Elle implique surtout la distribution de la routine à toutes les cellules alourdissant le logiciel de service de cette dernière. Cette option ne peut donc pas se justifier avec des considérations basées seulement sur les temps de traitement comme on l'a fait jusqu'ici.

Exemple de réalisation

C'est la solution 2 du paragraphe précédent qui est proposée. Le microordinateur employé pour la cellule est basé sur le Texas 9 940 compte-tenu du fait qu'il possède multiplication et division cablées ($\tau \simeq 34\ \mu s$)

Pour architecture, organigrammes : cf. : annexe II.

CHAPITRE 3

D.D.A CELLULAIRE

3.1 - INTRODUCTION

Le D.D.A. est un outil moderne du calcul analogique puisqu'il naquit en 1949-50 (Machine MADDIDA de Northrop). On peut le définir comme la transposition dans la technologie digitale des calculatrices analogiques. Comme ces dernières, il a pour but de résoudre des systèmes d'équations différentielles et procède tout comme elles en réunissant par un cablage divers blocs qui constituent les opérateurs élémentaires de l'analyse schématique qu'on a fait à priori. En dehors des applications temps réels, il s'est trouvé concurrencé par les calculateurs scientifiques dans la résolution d'équations différentielles mais il apparait toujours comme intéressant dans des applications conciliant rapidité d'exécution et précision devenant alors un concurrent direct des calculatrices analogiques. Grosso-modo les domaines d'utilisation de D.D.A. peuvent être :

- solutions des équations différentielles
- simulation des systèmes continus (aviation, missile)
- problèmes spéciaux :
  - navigation inertielle
  - calculateur balistique
  - contrôle numérique de machines outils
  - contrôle de contours
  - filtres digitaux
  - transformations linéaires (Fourier, Walsh).

3.2 - L'INTEGRATEUR

L'élément de base d'un D.D.A. est l'intégrateur.

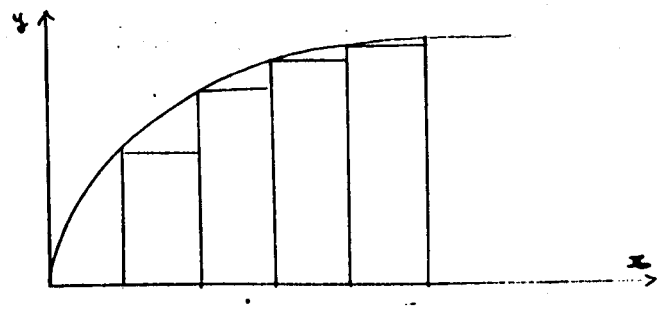
En effet un D.D.A. calcule les intégrales au sens de Stieljes par une quadrature

$$I = \int_a^b y(t)\, dx(t) = \sum_{k=0}^{n-1} A_k\, y(t_k)$$

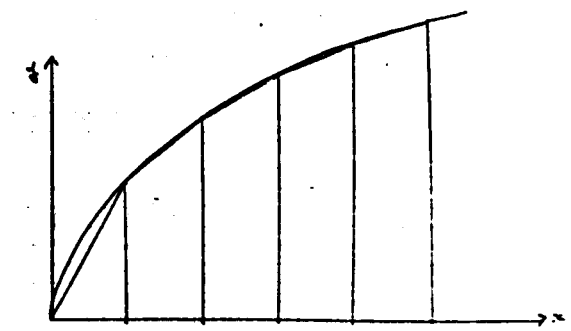
t_0, t_1 ———— t_n sont les abscisses

A_0, ———— A_n sont les coefficients de la formule de quadrature.

La plus simple, est celle dite rectangulaire ou d'Euler

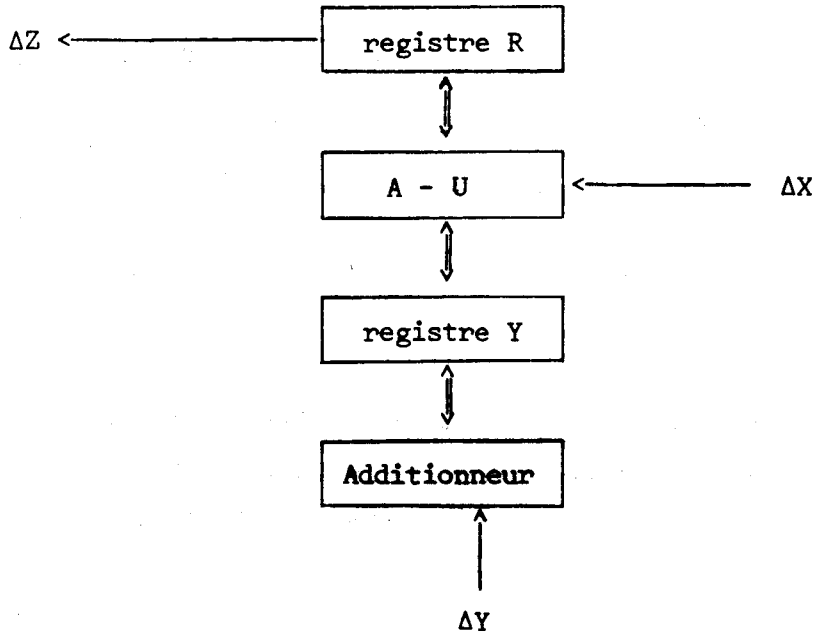


D'autres sont envisageables et réalisables dans un D.D.A. (Runge-Kutta, Heun).



On peut d'ailleurs fort bien envisager de concevoir un D.D.A. capable d'offrir à l'utilisateur le choix de plusieurs formules de quadrature.

Le bloc diagramme d'un intégrateur est celui donné par la figure :



Son fonctionnement se résume comme suit :

A chaque arrivée d'un incrément ΔX, le contenu du registre Y est ajouté à celui de R. Quand celui-ci déborde, l'overflow produit la sortie d'un incrément ΔZ. Y est tenu à jour par ΔY qui s'additionne à Y. ΔX, ΔY et ΔZ peuvent éventuellement être signés.

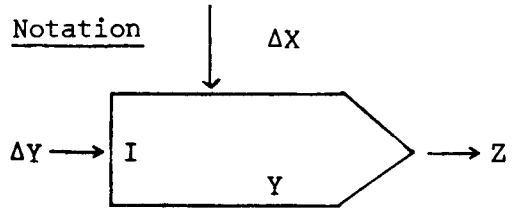
On peut écrire :

$$\Delta Z = \frac{1}{2^n} \cdot Y \cdot \Delta X \qquad \text{Si R possède n bits.}$$

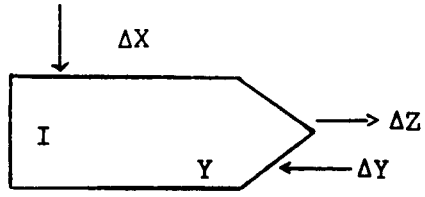
L'échelle de l'intégration peut se régler par un facteur K pour obtenir

$$\Delta Z = \frac{K}{2^n} \cdot Y \cdot \Delta X$$

Notation



a/



b/

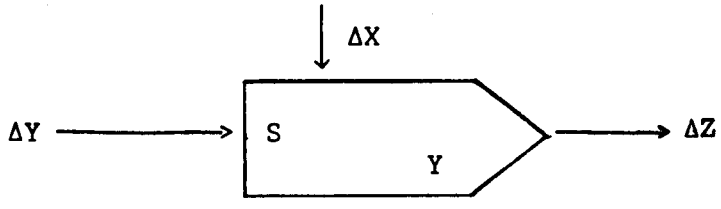
Dans la littérature on peut trouver les deux notations a/ et b/. On emploiera par la suite la notation "a" car c'est celle qui donnera un graphe le plus aisément transcriptible dans un espace cellulaire donné.

3.3 - AUTRES ELEMENTS DU D.D.A.

Si l'élément fondamental est l'intégrateur, d'autres opérateurs sont nécessaires ou pour le moins utiles.

Fonction servo

notation

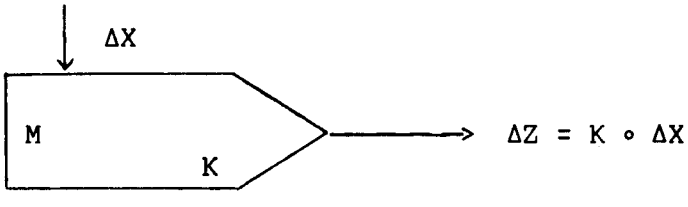


Si $Y \cdot \Delta X > 0$, le servo génère un incrément ΔZ positif.

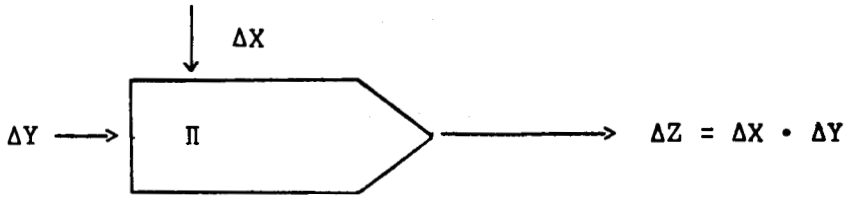
Il sera négatif dans le cas contraire.

Somme toute, le servo est un intégrateur simplifié.

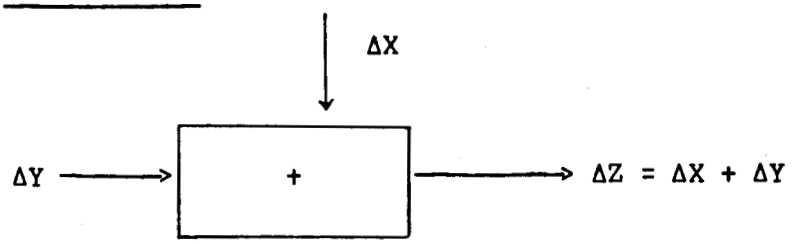
Multiplicateur par une constante



Multiplicateur par une variable

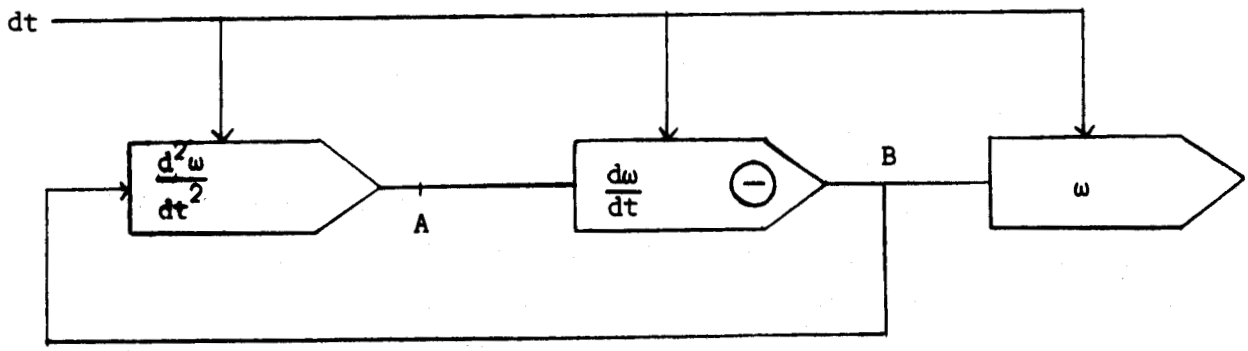


Additionneur



3.4 - PRECISION ET RAPIDITE

Un exemple va nous servir pour mieux apprécier l'antagonisme possible entre ces deux qualités à l'intérieur d'un D.D.A.



$$\frac{d^2\omega}{dt^2} = -\,\omega.\ \text{fonction sinus}$$

Supposons que l'on veuille obtenir ce sinus avec une précision de quatre chiffres décimaux au moins. Cela signifie qu'en A et B les différentielles des lignes trigonométriques qui sortent doivent être inférieures à 10^{-4}. En binaire cela équivaut à une valeur comprise entre 2^{-13} $(1,22 - 10^{-4})$ et 2^{-14} $(0,61 - 10^{-4})$ donc 2^{-14} si on suit l'énoncé du problème. Cela signifie qu'avec un incrément minimal tenant sur un bit, il faut que les registres Y possèdent 14 bits au moins. Si le processeur choisi manipule des registres de 8 bits, il faut qu'il travaille sur deux registres et il y aura donc augmentation du temps de traitement par rapport à une précision inférieure mais compatible avec un format de 8 bits et donc d'une précision inférieure à 0,4 %.

3.5 - INTERET D'UNE ARCHITECTURE CELLULAIRE

Outre les intérêts généraux dont nous avons discutés dans l'introduction et qui ont trait à la fabrication ou la maintenance, nous allons mettre en lumière les arguments qui peuvent plaider en faveur du cellulaire pour la réalisation de D.D.A. parallèles.

Un schéma analogique dont on s'inspire directement si on utilise des D.D.A. se définit comme un graphe orienté. Les noeuds sont les opérateurs et les arcs figurent les liaisons physiques établies entre ceux-ci. A un problème donné va donc correspondre un graphe précis et la programmation, ce mot étant pris au sens large, va revenir à :

primo spécifier les opérateurs
secundo à établir les liaisons entre-eux.

Cette opération devient ardue quand le problème est important car la complexité du graphe croît certainement plus vite que celle du problème. Quand le cablage est effectué manuellement, la méthode est non seulement lente mais aussi source d'erreurs. Pratiquement la définition du graphe peut être aidée par un programme sortant sous forme codée le plan de cablage

Une étape de plus vers la facilité de la mise en oeuvre est à nouveau franchie s'il existe un moyen physique manipulable par programme et reliant les opérateurs

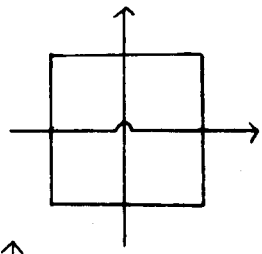
Ces deux opportunités sont d'ailleurs déjà proposées (27,33,46)

Il apparait aussi fort possible de définir un opérateur universel en ce sens qu'il est capable de réaliser à la demande chacune des opérations D.D.A. définies dans les paragraphes précédents. L'emploi de la microprogrammation apparaît alors tout indiqué pour permettre cette spécialisation. Une solution pour permettre la transcription des graphes inhérents à autant de problèmes divers est de concevoir un graphe physique aux connexions cablées et capable de les supporter tous. L'idéal à cet égard pouvant paraître que chaque opérateur possède une liaison potentielle avec tous les autres mais bien sûr le cablage de cette solution est irréaliste. On doit alors se rabattre ou sur un outil de communication programmé permettant d'établir ces liaisons, type "réseaux de connexions" (12, 33) ou alors sur un graphe de connexité réduite et bornée mais dont on aura enrichi les possibilités des opérateurs de façon à pouvoir y inscrire un graphe quelconque. Ces opérateurs universels et enrichis de fonctions annexes de routage sont nos cellules.

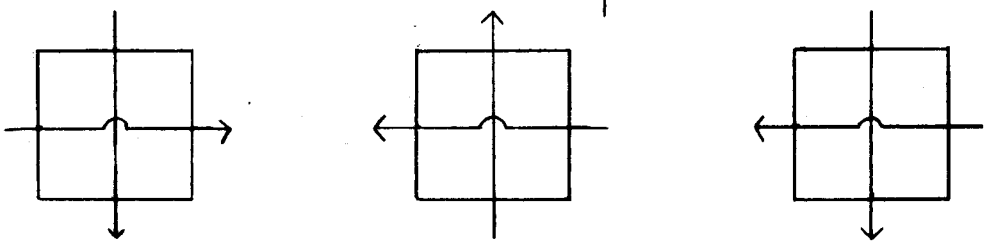
3.6 - FONCTIONS DE ROUTAGE

Il y en a trois plus une qui réalise une fonction de calcul.

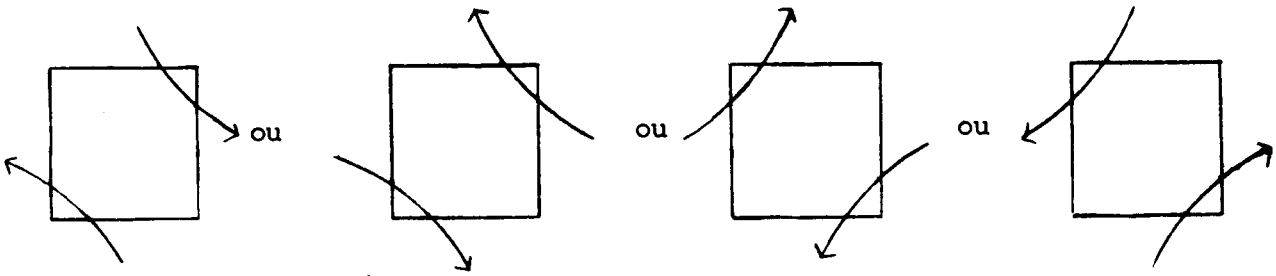
1/ Routage simple



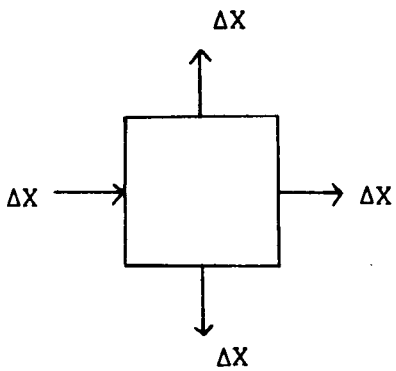
ou



2/ Routage dirigé (changements de direction)

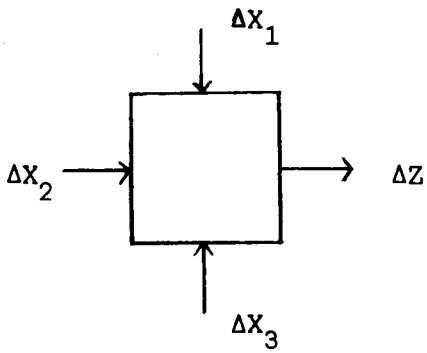


3/ Eclateur (1, 2 ou 3 sorties)



on spécifie l'entrée et les sorties avant exécution.

4/ Additionneur - concentrateur

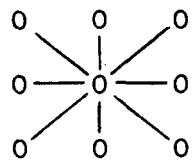


$$\Delta Z = \Delta X_1 + \Delta X_2 + \Delta X_3$$

On spécifie les entrées et la sortie avant exécution.

Comme on le voit, on s'est placé dans le cas d'un maillage carré à deux dimensions. Il est certain que l'emploi de maillages de connexité plus forte diminuerait le nombre de cellules de routage pour un même graphe théorique, ainsi que la longueur des chemins, générateur de distorsion s'ils sont trop longs car alors on ne peut plus négliger les temps de propagation des incréments.

Exemple : 2 D, maillage à 8 voisins. (voisinage de Moore).



On double les possibilités de routage des noeuds et on porte à 1 pour 7 le rapport de concentration et d'éclatement.

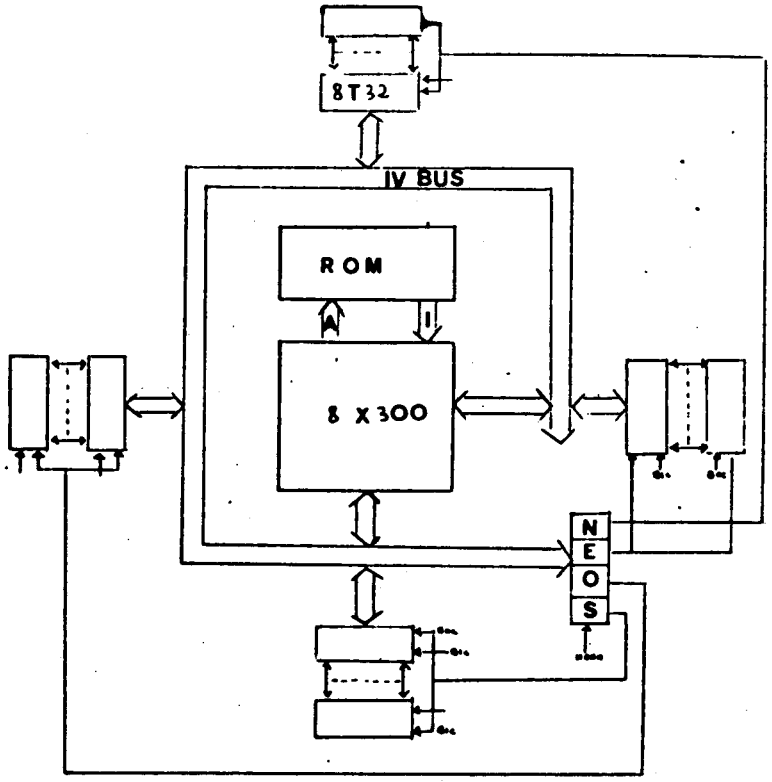
Cependant on alourdit la cellule de base et on la ralentit. C'est apparent dans l'exemple qui suit.

3.7 - PROPOSITION D'ARCHITECTURE

Il nous faut chercher une grande vitesse d'exécution. D'autre part la simplicité des traitements exécutables au niveau de la cellule, leur nombre relativement restreint font qu'on s'oriente vers une solution microprogrammée.

L'interprêteur 8 X 300 de signetics parait à cet égard assez intéressant. Il est rapide puisqu'un opérateur élémentaire peut prendre moins de 300 ns et le jeu de micro-instructions disponibles suffit amplement pour réaliser les différentes fonctions que nous avons passé en revue précédemment (62).

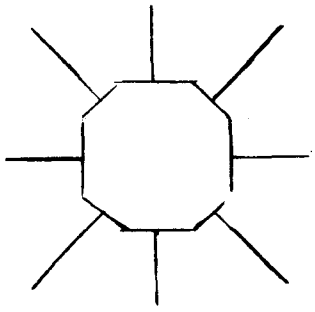
Dans une configuration à 2 D et un maillage carré, une cellule aurait la configuration type suivante :



Les cellules communiquent par l'intermédiaire de registres tampons commandés eux-mêmes par les signaux apparaissant sur un registre externe qui peut être du même type que les précédents (8T 32 ici).

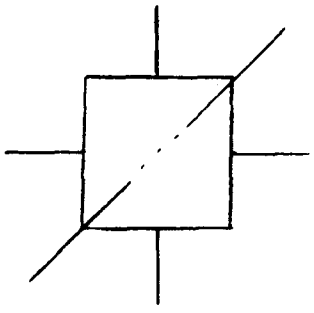
Si par exemple une cellule veut communiquer avec son voisin EST, c'est le bit 1 du registre de communication qui est positionné et sa lecture commande l'entrée du buffer OUEST de la cellule EST et la sortie du buffer OUEST de la cellule émettrice. Ajoutons que cette configuration est loin d'épuiser les possibilités de la cellule puisqu'on peut adresser jusqu'à 2 X 256 = 512 lignes entrée-sortie alors qu'ici une quarantaine sont utilisées.

BUS LILLE



2 D 8 voisins

72



3 D cube

56

Si on se permet d'échanger 16 bits à la fois :

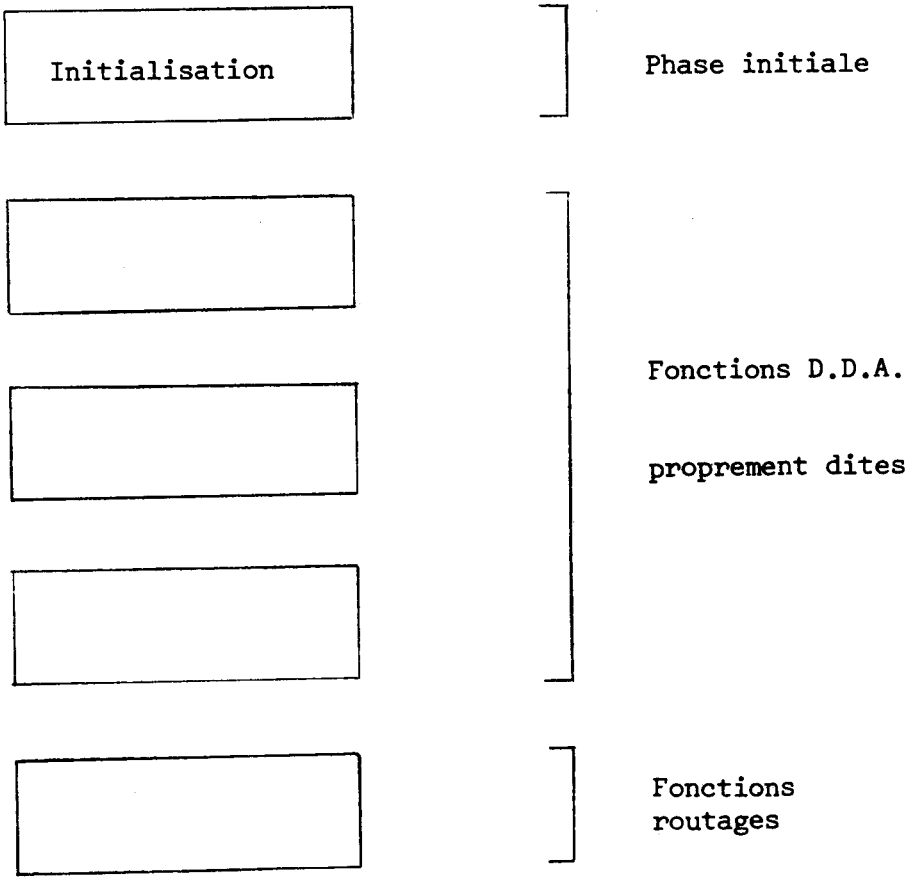
2 D	carré	72
2 D	8 voisins	136
3 D	cube	104 bits.

On pourrait peut-être essayer d'aller plus loin dans la taille des chemins mais le 8 X 300 ne possède que 8 registres de 8 bits. Ce ne serait donc possible qu'à condition d'ajouter une mémoire R.A.M. externe.

D'autre part le processeur travaillent sur 8 bits, communiquer des messages plus volumineux accroît le nombre de cycles, opérations nécessaires au transfert. Il est donc à priori peu rentable d'essayer d'accroître la taille des chemins de donnée.

3.8 - COMPORTEMENT DE LA CELLULE

Le logiciel d'une cellule active doit être capable de supporter trois sortes de comportement possible :



Cependant avant la phase d'initialisation proprement dite les cellules consultent en permanence leurs buffers d'entrée-sortie afin de consulter d'éventuels messages les concernant (ϕ_1). Pour le savoir elles

BUS

comparent sa référence à une référence interne. S'il n'y a pas de correspondance, le message est transmis à la cellule opposée à la voisine émettrice de ce dernier.

On pourrait éventuellement envisager un dispositif de routage analogue à S.T.U.D. dans une autre option. Quand une cellule trouve la coïncidence entre un message incident et sa référence interne, elle passe alors dans la phase d'initialisation proprement dite quelque soit la façon dont on a conçu cette dernière, la cellule doit acquérir au cours de cette phase :

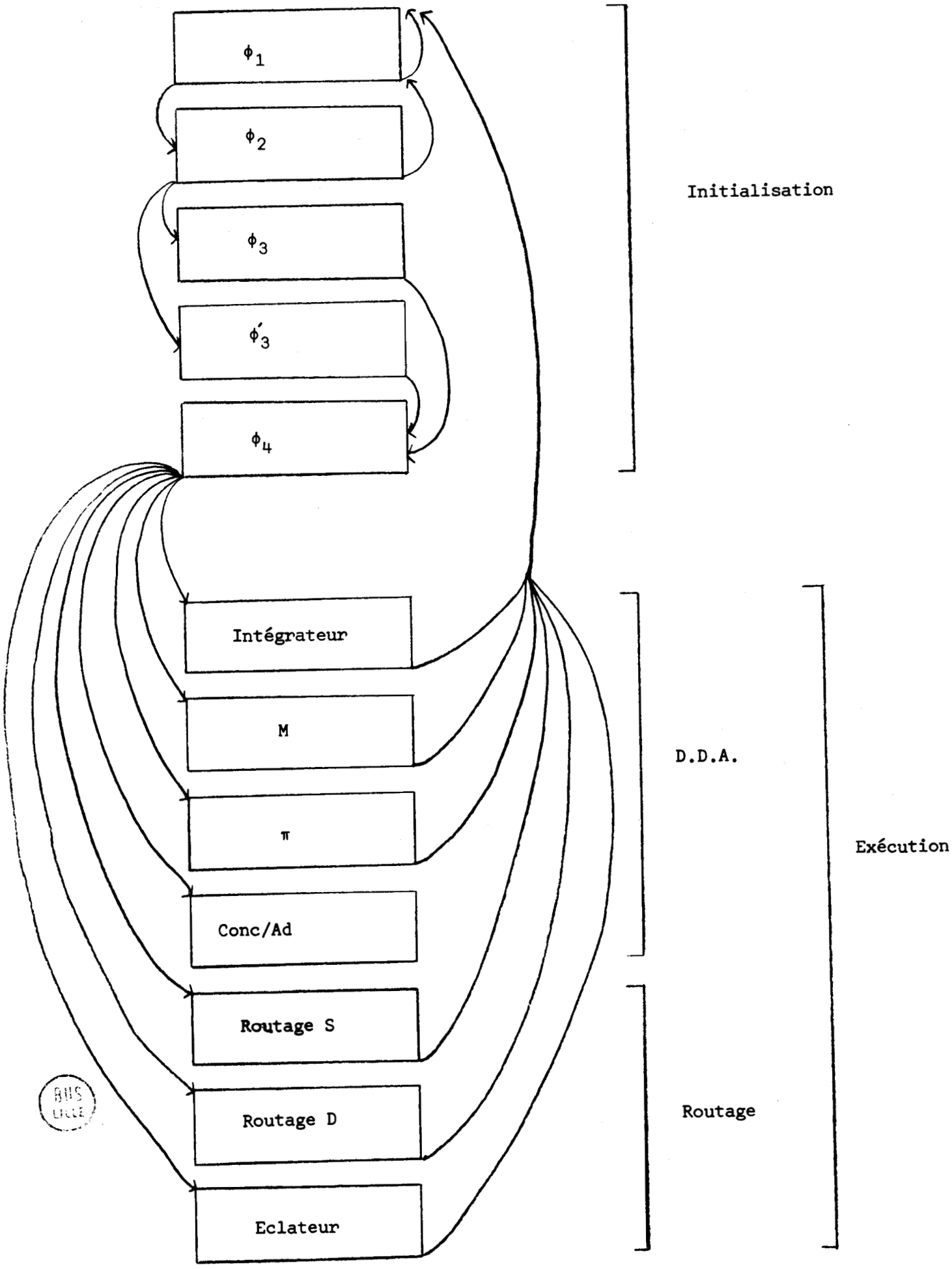
1° la référence de la fonction à exécuter (ϕ_2)

2° les paramètres nécessaires à cette exécution (ϕ_3)

Soit un exemple rédigé en termes triviaux : cellule (1,4) réalise l'intégration simple précision, ΔX vient de Nord, ΔY vient d'ouest, ΔZ est destiné à sud, la valeur initiale de Y est 0.

Quand une cellule a réuni tous les paramètres déterminant la fonction à réaliser, elle attend la réception d'un signal signifiant la fin de la phase d'initialisation générale, la dernière opération de la phase initiale consiste à placer dans le compteur de programme et le registre d'adresses à la valeur correspondant à la première instruction de la fonction réalisée par la cellule.

Le tableau qui suit résume ce comportement.


ϕ_1
ϕ_2
ϕ_3
ϕ'_3
ϕ_4
Initialisation
Intégrateur
M
π
Conc/Ad
D.D.A.
Exécution
Routage S
Routage D
Routage
Eclateur

BUS LILLE

Phase 1 : Phase d'attente.

Phase 2 : La cellule est élue. Attend la référence de la fonction à réaliser.

Phase 3 : Suivant la fonction à réaliser, les paramètres sont différents en genre et en volume. Il y a donc autant de phases 3 possibles que de fonctions réalisables.

Phase 4 : La cellule est parée pour l'exécution mais attend pour passer réellement en mode exécution que l'initialisation globale soit effective. Celle-ci est signalée par un message particulier.

3.9 - SOLUTION CHOISIE A TITRE D'EXEMPLE

- Maillage carré 2 D.

- Taille max des incréments 8 bits dont 1 bit désigne.

Cela signifie en particulier que l'on peut, au plus introduire un facteur K tel que $\Delta z = K. \Delta y$ égal à 128 au plus.

Cette contrainte est à fournir au programme qui dans une solution classique est chargée sur un calculateur associé au D.D.A. de résoudre les délicats problèmes des échelles à employer et des divers cadrages à effectuer, problèmes parfois inextricables dans des applications complexes.

En effet il faut concilier les contraintes suivantes :

- Y ne doit pas déborder.

- Compatibilité de la taille des incréments avec la précision désirée.

- Compatibilité des échelles entre variables reliées par une même formule.

- Calcul des valeurs initiales compte-tenu des échelles choisies.

Comme on le voit, le pré-traitement peut être d'importance dans les cas où par exemple une connaissance préalable de la solution n'existe pas. Ceci explique que l'association au D.D.A. d'une calculatrice universelle chargée de ce travail préparatoire est pratiquement indispensable dans un cadre opérationnel.

D'autre part nous dotons la cellule de deux modes : simple ou double précision, Y et R pouvant tenir sur 8 bits ce qui donne une précision de 0,4 % dans le cas d'équations différentielles sans coefficient. C'est aussi le mode de calcul "rapide". Dans le mode double précision Y et R tiennnent au plus sur 15 bits soit une précision de l'ordre de 3/100 000 dans les conditions précédentes. Le 8 X 300 et les microprogrammes de la fonction D.D.A. sont présentés en annexe (AN I)

CHAPITRE 4

MACROPIPE-LINES ET ESPACES CELLULAIRES

1 - MACROPIPE-LINES

Le concept de "pipe-line" (11, 16) a été une source importante d'innovations dans l'architecture des machines informatiques. Bien sûr, tout comme l'idée de cellularité, on le trouvait implicitement dans de nombreuses réalisations humaines, la division du travail et le Taylorisme étant des exemples parmi d'autres. Compte-tenu de notre volonté d'utiliser comme cellule des microordinateurs ou microprocesseurs existants, il ne sera question que de macropipe-line, c'est-à-dire que le caractère pipe-line du traitement apparaît au niveau de paquets ou blocs d'instructions, d'une certaine taille. C'est donc une définition qualitative.

D'autre part s'il importe de ne pas confondre ce type de traitement avec le découpage réalisé au niveau du cycle exécution-recherche des instructions, certains caractères apparaissent spécifiques à toute organisation pipe-line quelque soit son niveau. Ti Chi Chen (11) l'a défini comme une "technique de traitement visant de forts débits. Le pouvoir de traitement est décentralisé, distribué plus ou moins uniformément à travers le chemin de traitement". En effet si nous considérons n unité de traitement mises bout à bout, chacune exécutant une part du traitement global sur un train de données parcourant séquentiellement la chaîne et si on suppose le temps élémentaire de traitement identique pour tous les éléments, le débit est égal à l'inverse de ce temps élémentaire de traitement (ou au MAX des temps élémentaires s'ils sont différents). Ceci constitue en quelque sorte la relation de base mais on conçoit qu'à partir de là des réalisations très diverses peuvent être proposées selon en particulier ce qu'on entend par "élément" de traitement. Une solution généralement mise en avant est de constituer les chaînes pipe-lines à partir de composants spécialisés, le chemin d'une donnée

à travers l'ensemble de ces composants étant déterminé à partir d'un tableau de réservation (16) après avoir éliminé les conflits d'accès à un même élément au même instant. Cette possibilité très attrayante dans un pipe-line découpant une instruction existe bien sûr dans un contexte macropipe-line. Cependant dans un espace cellulaire et compte-tenu de caractères variables qui n'apparaissent pas à un niveau plus bas d'exécution, d'autres façons de procéder paraissent intéressantes à explorer.

2 - CARACTERISTIQUES D'UN MACROPIPE-LINE

Au niveau d'un élément va se dérouler un traitement composé par l'exécution de plusieurs instructions successives correspondant à un véritable programme que nous appelerons provisoirement "bloc d'instructions". La réunion ordonnée de ces blocs reconstituent une tâche précise pour laquelle on peut définir un ensemble de données initiales qui vont se transformer peu-à-peu pour donner un ensemble de résultats que l'on recueille. On voit qu'apparaissent des éléments variables qui n'existaient pas à un niveau plus bas : ce sont les données qui transitent et dont le nombre et le format peuvent varier. Il est normal aussi de trouver des propositions conditionnelles voire des boucles dont on ne connaît pas à priori la borne d'itération. La structure cellulaire proposée doit donc s'accommoder de ces nouveaux éléments, voire essayer de les gommer de façon à assurer soit le meilleur débit soit le meilleur temps d'exécution.

Compte-tenu de notre volonté d'éliminer la notion de contrôleur ou séquenceur central, les messages injectés dans l'espace cellulaire devront porter l'information nécessaire aux cellules pour assurer à la fois les traitements proprements dits mais aussi la constitution des trains pipe-lines et leur gestion. Ceci n'est possible qu'en dotant chaque cellule de certaines propriétés tant internes qu'externes de relation avec les voisins. C'est ce qu'on fait apparaître dans le modèle proposé.

3 - DÉBIT ET TEMPS D'EXÉCUTION

Parler en terme de pipe-line, c'est penser débit. Cependant il ne faut pas oublier ce paradoxe, apparent car basé sur une approximation grossière, que si le débit augmente à l'infini, le temps d'exécution fait de même. C'est que l'on doit tenir compte très rapidement de facteurs autres que le traitement proprement dit et qui sont les temps de service où on trouve côte-à-côte et entremêlés les temps de transit des mots entre les opérateurs et les temps d'exécution des routines de servitude si l'élément pipe-line est adaptatif comme on se propose de le faire ici. D'autre part, en dehors de pipe-lines classiques définis au niveau de l'instruction ou d'applications spécialisées (traitement du signal par exemple), le critère de performance déterminant n'est pas le débit optimal mais le temps total d'exécution minimal pour traiter tous les objets attachés à une tâche, en particulier lorsqu'on peut connaître ou tout au moins estimer leur nombre.

C'est ce qu'on démontre ici.

Soit T_0 le temps total pris pour exécuter la tâche séquentiellement.

Soit m le nombre d'éléments du pipe-line.

Soit τ le temps de service, fonction des temps de transfert et temps d'exécution des routines de servitude, nécessaire pour passer d'un élément à l'autre.

$$- \text{débit} = \frac{1}{\frac{T_0}{m} + \tau}$$

Soit n le nombre de données à traiter.

Soit T_{XQ} (m,n) le temps total d'exécution pour réaliser les n traitements.

Nous allons déterminer en fonction de m, pour T_0, τ et n constants la courbe traduisant les variations de T_{XQ}.

$$T_{XQ}(m) = T_0 + (m - 1)\tau + (n - 1)\left(\frac{T_0}{m} + \tau\right)$$

$$T_{XQ}(m) = (n - 1)\frac{T_0}{m} + m\tau + [T_0 + (n - 2)\tau]$$

$$\frac{d\,T_{XQ}}{dt} = \tau - \frac{(n - 1)\,T_0}{m^2} \quad \text{avec} \quad m > 1$$

Il existe une valeur de m annulant la dérivée.

$$\frac{d\,T_{XQ}}{dt} = \frac{m^2\,\tau - (n - 1)\,T_0}{m^2} \Longrightarrow \boxed{m_0 = \sqrt{(n - 1)\frac{T_0}{\tau}}}$$

Cette valeur est la valeur optimale assurant le meilleur temps d'exécution total.

Posons $k = \frac{T_0}{\tau}$

La figure 4-I montre l'allure des courbes T_{XQ} (m) pour différentes valeurs de k. Les échelles sont logarithmiques compte-tenu des disparités importantes d'échelles.

Le tableau I-4 illustre certains rapports intéressants. Remarquons surtout le rapport $\frac{\theta}{\tau}$ (1). On peut voir qu'il peut être très inférieur à 1 tout en permettant un gain fort appréciable. Ceci montre que la tentation de faire du rapport $\frac{\theta}{\tau}$ un indicateur sur la rentabilité d'une application pipe-line était mauvaise puisqu'une cellule peut passer individuellement beaucoup plus de temps à exécuter des routines de service que le traitement en cours sans pour autant enlever de l'intérêt à l'emploi du macropipe-line.

(1) θ : Temps d'exécution d'un bloc à l'intérieur d'une cellule $= \frac{T_0}{m}$

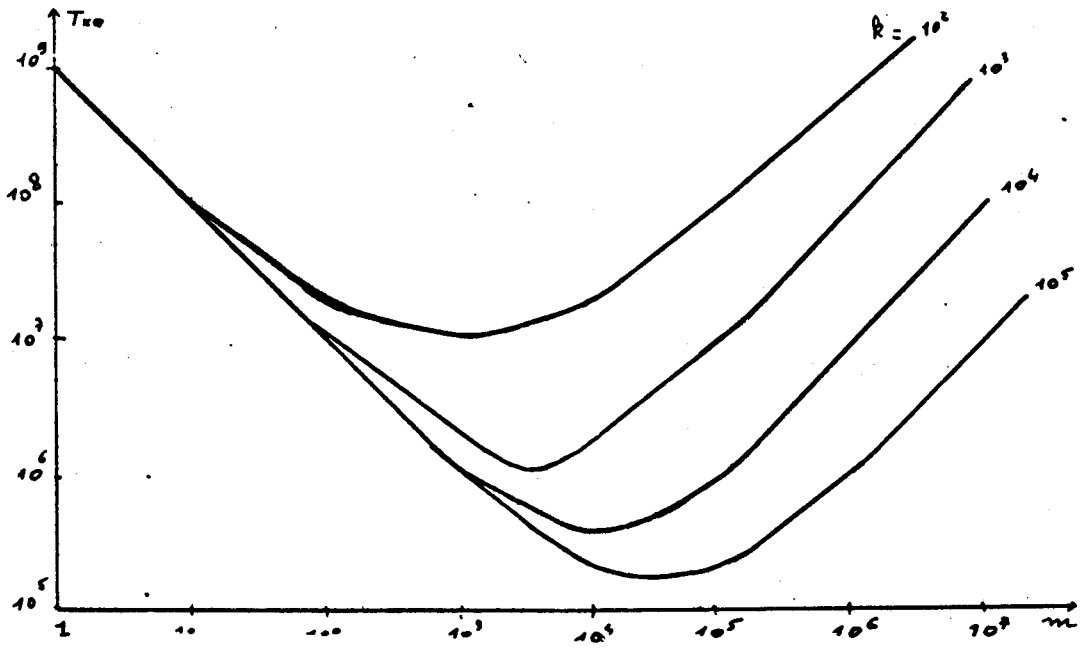


Fig. : 4.I TxQ = f(m)

BUS LILLE

τ		10^4	10^3	100	10	1	0,1
k	10	100	10^3	10^4	10^5	10^6	10^7
m_0	3,16	10^3	$3,16\ 10^3$	10^4	$3,16\ 10^4$	10^5	$3,16\ 10^5$
$\frac{T_S}{T \times Q} = \sigma(m_0)$		82,64	577,36	2 500	5 780,35	8 271,3	9 390,63
$\theta/\tau(m_0)$		0,1	0,316	1	3,16	10	31,6
$\sigma(10^3)$		10	45,4	10^3	10^3	10^3	10^3
$\theta/\tau(10^3)$		0,1	1	10	10^2	10^3	10^4
$m_0/10^3$		1	3,16	10	31,60	10^4	$3,16\ 10^2$
$\sigma(m_0)/\sigma(10^3)$		1	1,49	2,5	5,78	8,27	9,4

$\sigma = \frac{T_S}{T_{XQ}}$ Avec T_S temps équivalent pour traiter sur un microprocesseur le même nombre de données.

$\theta = \frac{T_0}{m}$

On peut voir aussi que si la valeur de m_0 parait difficile à obtenir pour des valeurs de k élevées, des valeurs de m de l'ordre de 10^2, 10^3 procurent déjà des gains intéressants. Ce n'est donc que pour des valeurs de k assez faibles que l'on a le plus grand intérêt à travailler avec des valeurs de m proches de m_0.

Le rapport k dépend :

- de la nature du traitement (T_0),

- du temps de transfert de mots élémentaires de cellule à cellule (débit, largeur des chemins),

- des routines internes assurant la sortie des messages ou leur émission.

- la politique de découpage d'une tâche en bloc.

Si le contrôle du premier paramètre échappe au concepteur (du moins tant que nous ne spécialisons pas (ex. : F.F.T), les autres requièrent notre attention. Le deuxième paramètre est avant tout un problème d'architecture alors que les 3ème et 4ème vont apparaître liés intimement par la suite.

4 - ELASTICITÉ

Une qualité intéressante d'une structure macropipe-line mise en avant par Cooper (14) est "l'élasticité". Il la définit comme le fait qu'aucun temps n'est perdu dans le pipe-line à attendre une donnée que ce soit à l'entrée ou à la sortie d'un élément du pipe-line. On emploiera plutôt ici le terme d'élasticité dans un sens plus large, marquant le fait que le débit de sortie s'adapte dynamiquement en débit d'entrée différé du temps de traversée ou reste fixé à une valeur moyenne déterminée à l'avance.

Dans le premier cas il pourrait s'agir de traitements sur des données saisies à la volée et dont le débit peut connaître de fortes variations, dans le second cas le débit de sortie est fixé par des conditions de récupération des résultats par exemple le débit d'écriture dans une mémoire secondaire.

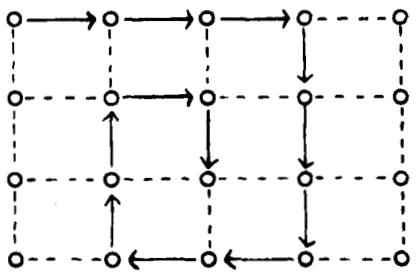
En effet rechercher un gros débit n'a de sens que tant que l'on puisse récupérer rapidement les résultats. Sinon on ne fait que transférer une file d'attente de données avant traitement en une file d'attente des résultats, situations qui si elles sont maintenues amènent rapidement la saturation du système. On peut d'ailleurs aussi prévoir un parallélisme géométrique entre des opérateurs I/O qui se répartissent la tâche.

5 - TOPOLOGIE

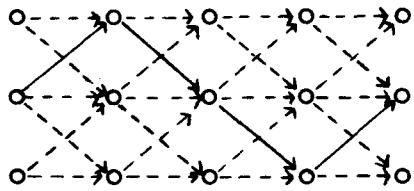
Si on veut l'affranchir d'un contrôle externe à l'espace cellulaire, il importe qu'une décision locale prise au niveau d'une cellule ne puisse être contradictoire avec une autre prise par un autre élément de l'espace cellulaire ce qui amènerait une situation bloquée. Il apparait que si on se place dans une topologie régulière et isotrope tel un maillage carré, chemins E/S confondus ou couplés, il parait difficile de respecter cette contrainte du moins en gardant au niveau de la cellule des routines de service simples.

1° Exemple

Une cellule choisit arbitrairement son voisin successeur.



Maillage isotrope



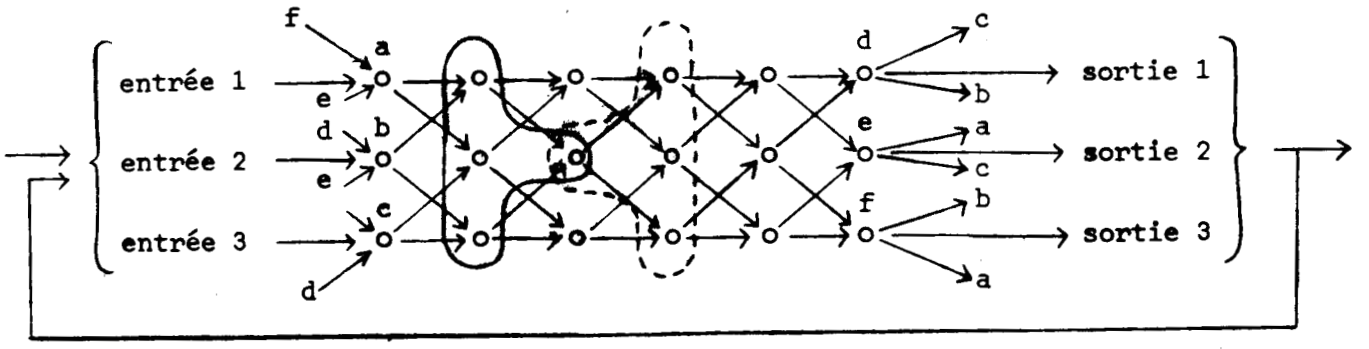
Maillage
diagonal
carré
anisotrope

Pas d'autoblocage possible dans le dernier cas.

Bien sûr c'est un exemple auquel on peut reprocher de n'être pas informatique parce qu'un choix arbitraire nécessiterait pratiquement un tirage au sort et donc l'exécution d'un programme adéquat. En fait il est plus simple de donner à une cellule la possibilité de sauter l'état de ses voisines selon un ordre de scrutation préétabli et d'élire la première cellule libre rencontrée, ce qui peut s'écrire :

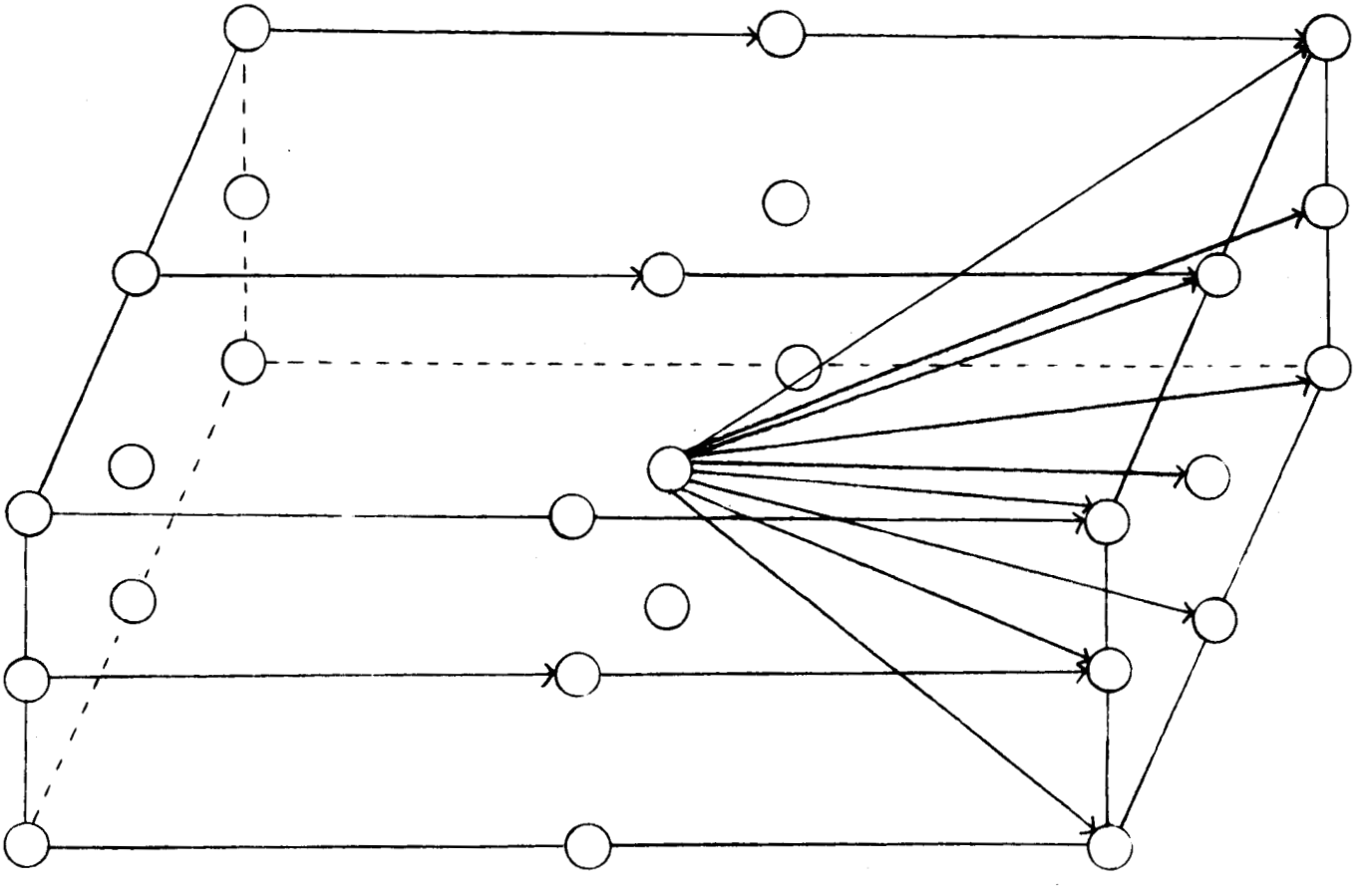
```
BOUCLE  Si  NORD  libre alors initialiser  NORD  sinon
        si  EST   libre alors initialiser  EST   sinon
        si  SUD   libre alors initialiser  SUD   sinon
        si OUEST libre alors initialiser OUEST sinon
        aller à BOUCLE fsi fsi fsi fsi
```

Employer un ordre de scrutation apporte déjà la notion d'un sens de propagation préférentiel (ici NORD). L'inscrire dans la topologie sert de garantie supplémentaire pour éviter tout risque de conflit dans le choix de successeurs entre éléments d'un même pipe-line pouvant conduire à des situations d'autoblocage. Inscrire un sens de propagation dans la topologie est réalisé par exemple dans la structure suivante :



Une cellule possède un ensemble de voisins successeurs distinct de celui des voisins prédécesseurs. Le P2 le fait de gauche à droite et l'émission ou la réception des messages se fait en des points différents, donc spécialisables et plus simples à concevoir. Faire de la première colonne le successeur de la dernière permet de traiter des problèmes de dimensions très supérieures à celui du tableau cellulaire afin par exemple d'approcher la valeur m_0 du paragraphe 3. On peut noter que le faisant on réintroduit un risque de blocage qui viendrait cette fois du fait qu'un message terminal bouclerait dans l'espace cellulaire sans sortir. Il parait judicieux alors de forcer le passage d'un message terminal dans la direction amenant obligatoirement la sortie (ici CENTRE).

Autre topologie possible, à 3D cette fois.



6 - PIPE-LINE ÉLASTIQUE ET RECONFIGURABLE

Cette même volonté de réduire le rôle d'un contrôle global externe rend peu intéressante la notion de synchronisme temporel puisque l'on veut appréhender des paramètres variables tels les largeurs des données, la présence de petites boucles internes aux cellules ou des propositions conditionnelles. C'est donc aux messages circulant dans l'espace cellulaire de porter l'information suffisante pour assurer la bonne exécution des tâches.

Un message circulant entre les cellules va comprendre au minimum :

- un identificateur : c'est un numéro référençant l'exécution en cours dans un contexte multi-tâche et le numéro d'entrée des données en cours dans cette tâche.

- un indicateur de fonction : référence le bloc d'instruction.

- opérande : il recouvre un ensemble de données dont une part est exécutable par le bloc référencé par l'indicateur, l'autre pouvant être réservée à des blocs d'indices plus élevés et donc simplement en transit.

Message type

ident.	indi.	OP	
		DI	DT

A l'intérieur d'une même tâche, on passe d'un bloc au suivant en incrémentant un numéro de référence (ou mieux en le décrémentant si on prend la convention de renuméroter 0 tout bloc terminal d'une tâche) tant qu'il n'y a pas rupture de séquence au niveau des blocs. Quand le cas se produit l'indicateur à inscrire dans le message ne peut être obtenu par calcul mais directement, c'est-à-dire qu'il est contenu nommément dans le bloc référencé précédemment.

7 - DIVERSES OPTIONS POUR LA CONSTRUCTION DU TRAIN PIPE-LINE

Tout comme dans la première partie, le comportement d'une cellule et ses relations avec ses voisines sont dictées par un programme de service qui peut se décomposer en un certain nombre de routines. Dans la perspective de résoudre des problèmes de complexité croissante, nous allons voir le cas où le comportement d'une cellule est dicté par une seule routine de base dont nous allons définir l'organigramme.

Initialement chaque cellule est dans un état que nous appelerons "état libre". Dans cet état, elle scrute périodiquement le buffer d'entrée afin d'y détecter la présence d'un éventuel message. Quand un message est présent, on le dépouille.

Ce dépouillement va consister essentiellement à :

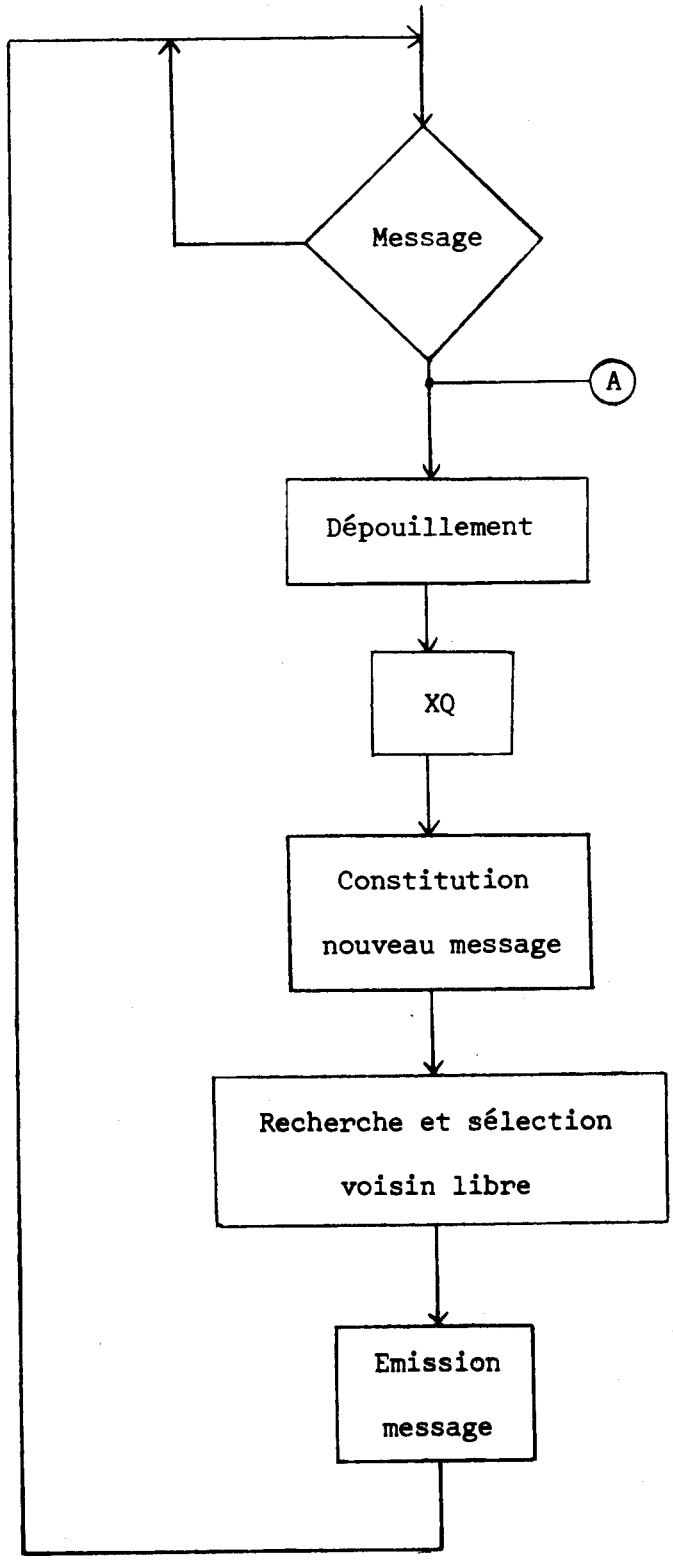
- prendre l'indicateur et le comparer avec une table de référence afin de trouver l'adresse de la première instruction exécutable du bloc.

- ranger identificateur et données en transit dans une zone protégée.

- ranger les données initiales dans une zone de travail.

On passe ensuite à l'exécution. Quand celle-ci est terminée la cellule reconstruit un nouveau message destiné à l'émission : l'identificateur est inchangé mais l'opérande est reconstruit en nouvelles données initiales et données en transit. (cf. fig. : 4.2)

La cellule recherche ensuite un voisin libre selon l'ordre de scrutation qui lui a été imposé et lui envoie son message. Elle repasse ensuite à l'état libre pour revenir scruter son buffer. Ce comportement est illustré par l'organigramme suivant :



Organigramme 1

BUS LILLE

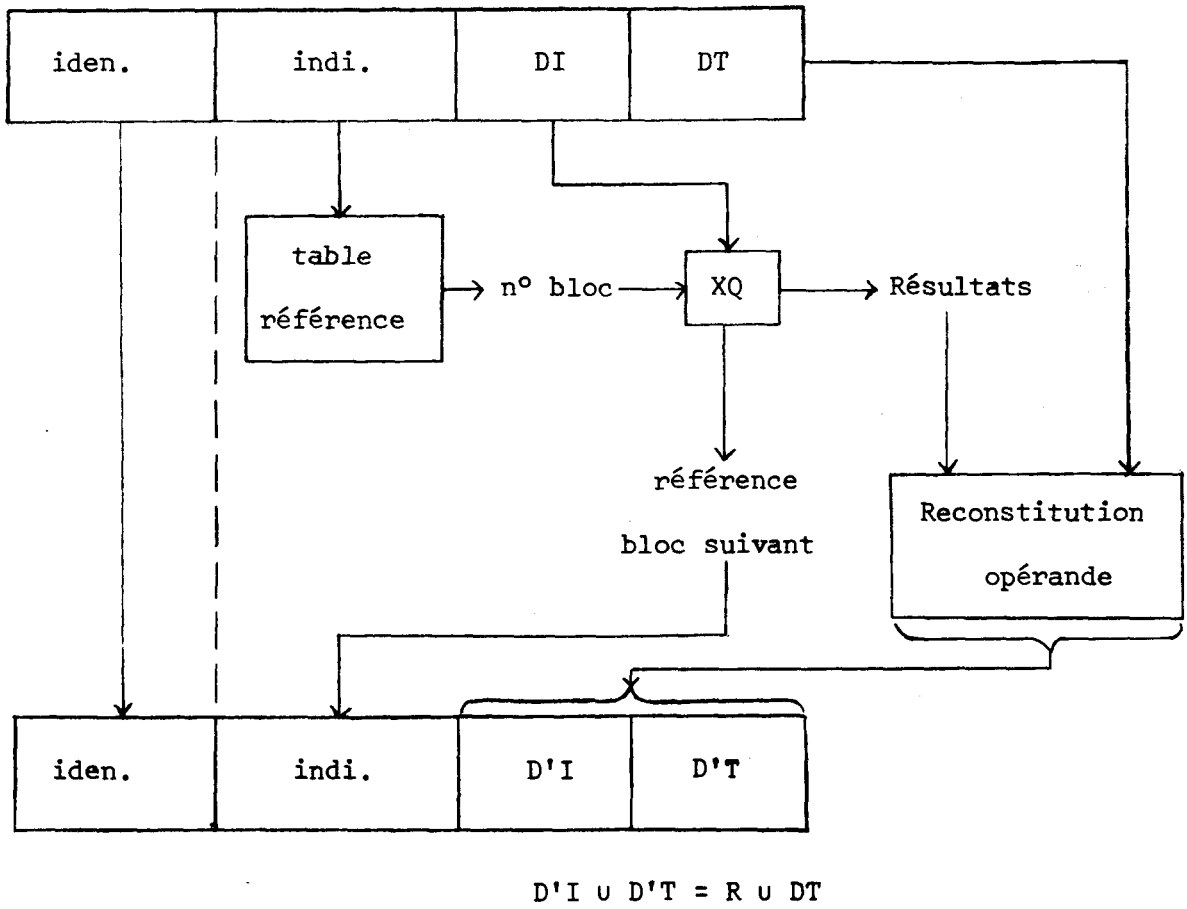


Fig. : 4.2 - Traitement message.

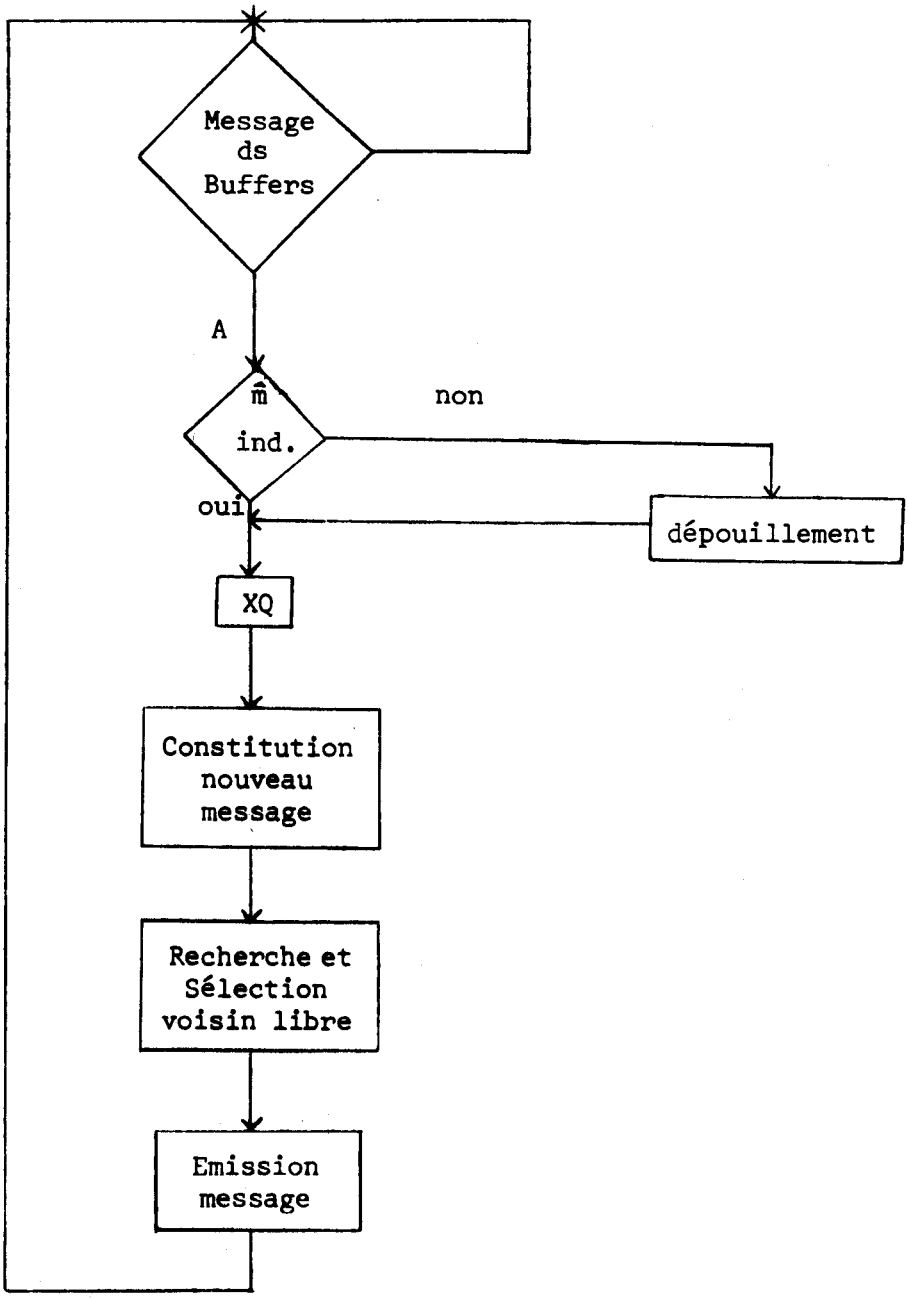
On doit observer aussi qu'un bloc n'est pas seulement un paquet d'instructions de la tâche en cours mais comprend aussi une part de service pour retrouver les données initiales dans l'opérande d'entrée et ensuite reconstituer l'opérande de sortie, deux opérations variant d'un bloc à l'autre et qui ne peuvent donc être réalisées par des routines de service résidentes. Tel qu'il est, l'algorithme présenté est simple mais peut paraître assez peu réaliste pour trois raisons :

7.1 - PREMIERE RAISON

Notre objectif est de construire un macropipe-line aussi performant que possible. Or le fait que la cellule repasse systèmatiquement à l'état libre et donc au redépouillement de l'entête de chaque nouveau message n'est pas

intéressant si, comme il est normal de l'espérer, la probabilité est forte pour que ce soit le même bloc référencé. Autrement dit on devrait essayer de diminuer k. Cet handicap disparait si on introduit une notion d'anticipation : après exécution, la cellule se replace en début d'exécution du bloc précédemment référencé. Le dépouillement est alors remplacé par un simple test d'égalité sur l'indicateur et l'exécution commence aussitôt s'il est positif. Dans le cas inverse seulement on recommence l'opération de dépouillement qui n'est pas négligeable parce que comportant la recherche d'une référence dans une table.

L'organigramme 1 devient celui-ci :

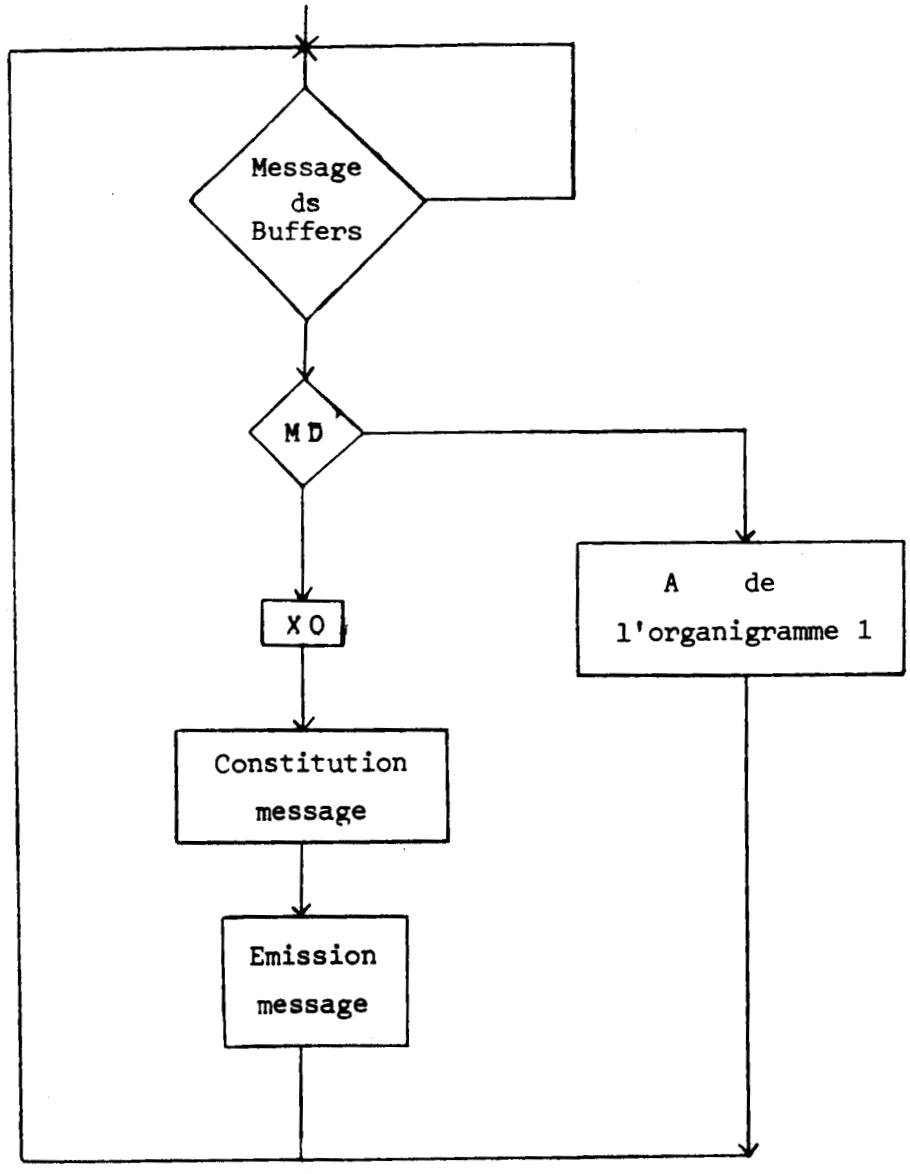


Organigramme 1-a.

On peut aller plus loin dans cette voie en liant successeur et prédécesseur de façon à ne pas refaire systématiquement l'opération de recherche et sélection du successeur libre.

On peut alors utiliser un indicateur booléen positionné par le prédécesseur chez son successeur sélectionné au premier message.

Ceci nous donne l'organigramme 1-b.

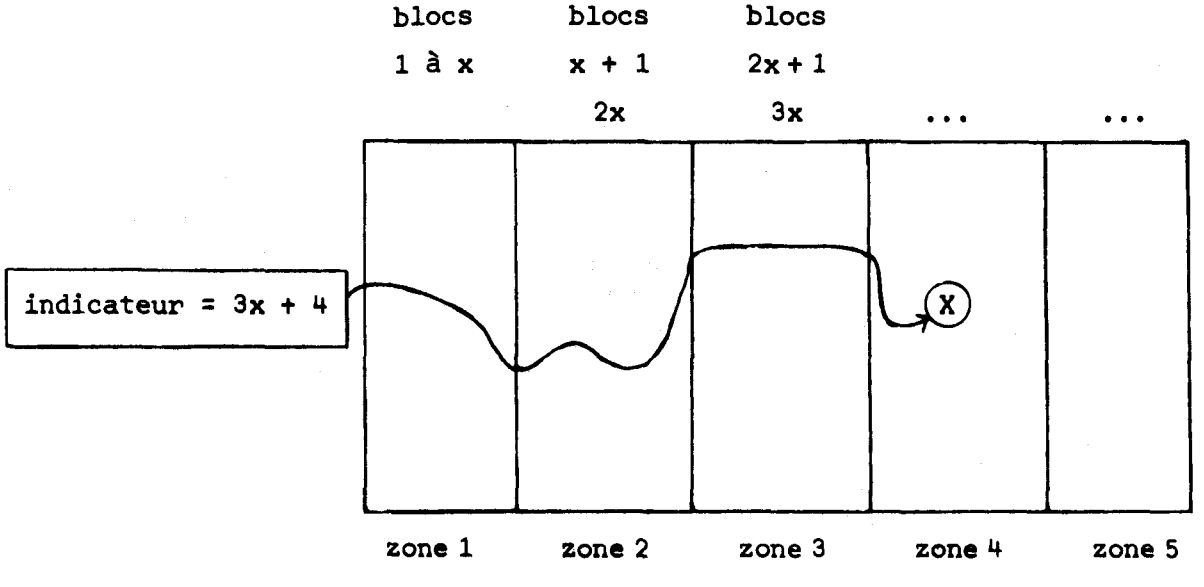


Organigramme 1-b.

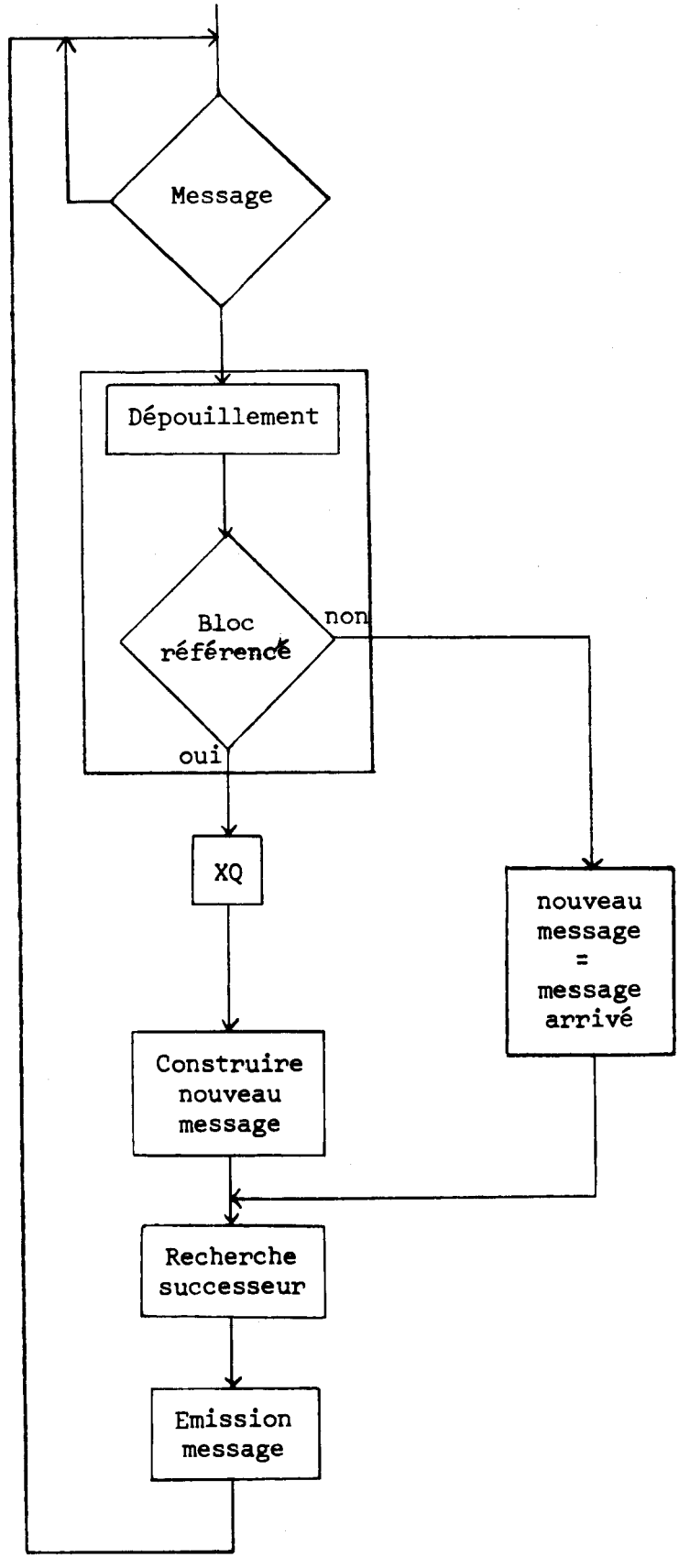
Quelles que soient les modifications apportées à l'organigramme 1 qui reste la référence de base, il est certain que des dispositifs hard permettant une recherche associative et de la table des références de bloc et de l'état des voisins successeurs donneraient des valeurs de k assez élevées pour rendre inutiles les modifications a et b du schéma initial.

7.2 - SECONDE RAISON

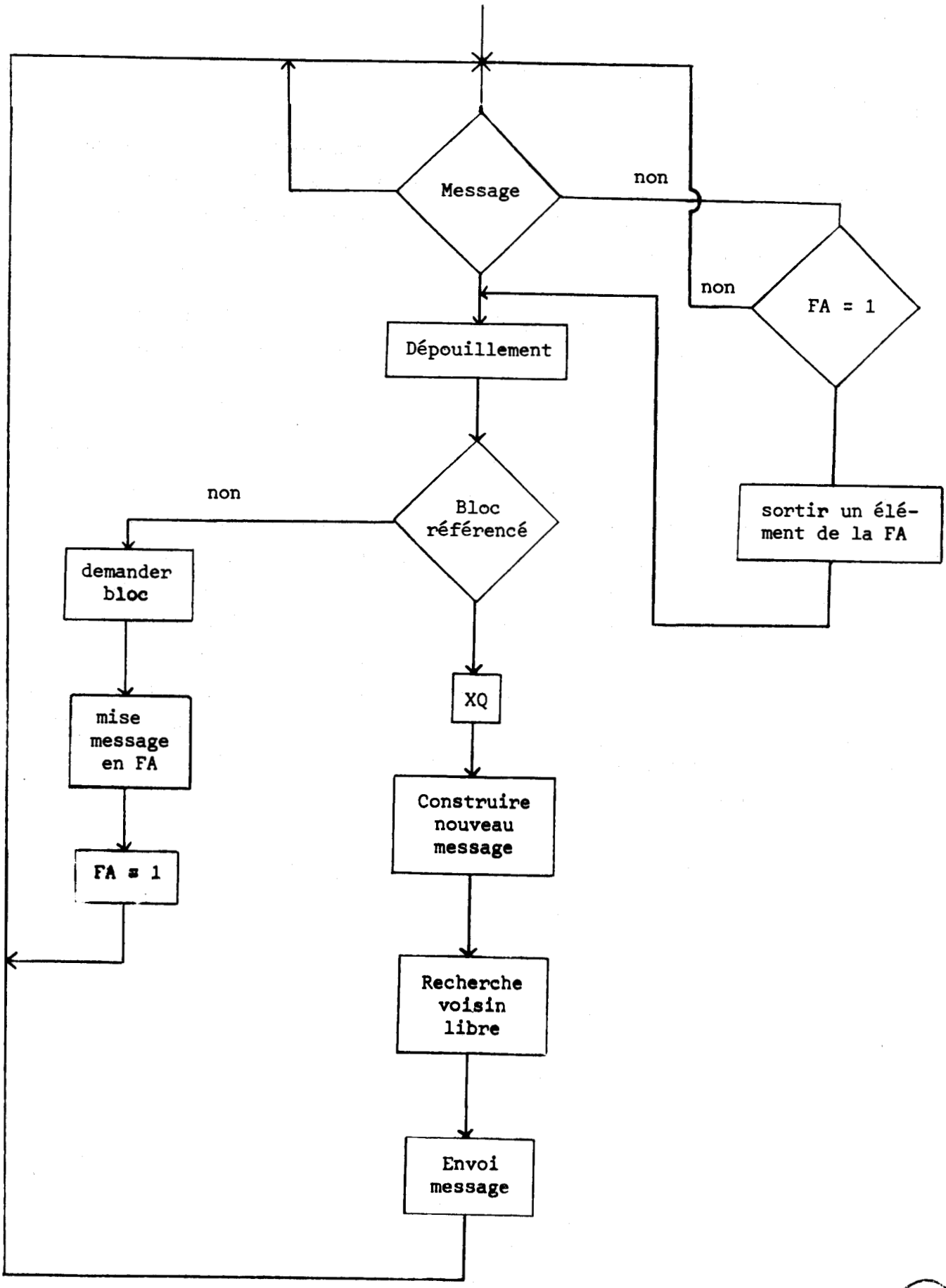
Il n'y a que dans des cas particuliers et dans le cadre d'une machine spécialisée dans un traitement donné, que l'on retrouvera une cellule capable d'exécuter tout bloc référencé par un message quelque soit ce dernier. Même dans le cas où on suppose un préchargement des blocs dans les mémoires cellulaires avant le début d'exécution de la tâche par suite d'un prétraitement sur les tâches en attente, il est prudent de prévoir le défaut de bloc. Deux stratégies sont possibles alors. Dans la première, la cellule laisse passer tout message dont l'indicateur n'est pas reconnu par la table des références des blocs pour un successeur libre. Ceci suppose alors que l'on soit assuré que le trajet quelconque parmi toutes les trajectoires possibles d'un message coupera au moins une cellule possédant la référence du bloc comme illustré sur la figure :



L'autre stratégie, est qu'une cellule ayant détecté un défaut de bloc demande à l'extérieur la transmission de ce bloc, quitte à mettre le message incident dans une file d'attente pour continuer de traiter tout nouveau message incident. Cela correspond à l'algorithme 1-x-a.



Organigramme 1-x-a.



BUS LILLE

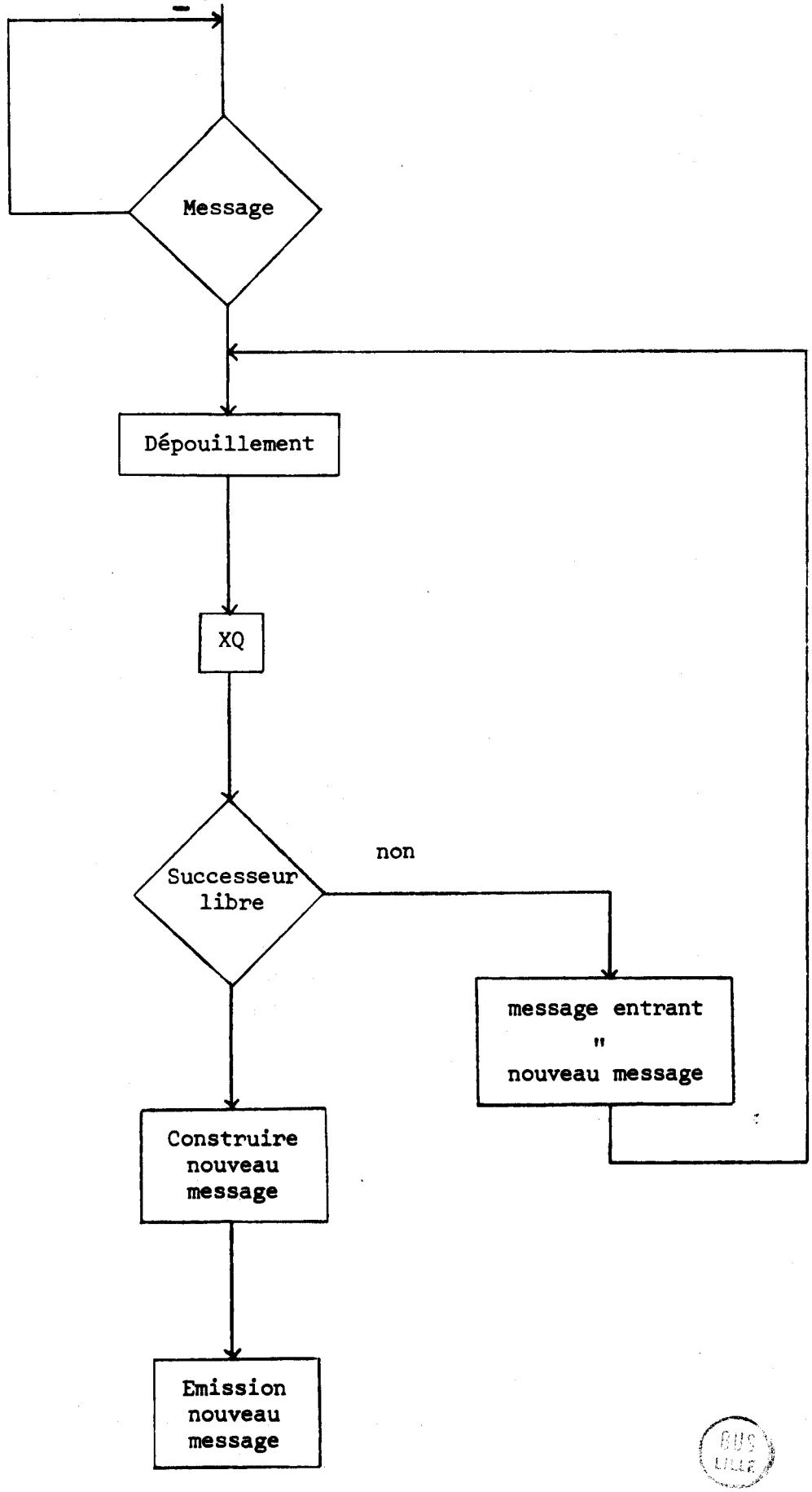
Organigramme 1-x-b.

7.3 - TROISIEME RAISON

La troisième raison est qu'il faut admettre que dans des cas extrêmes, la cellule ne puisse trouver un successeur libre par suite d'une saturation qui ne peut être éphémère si on n'a pas surchargé le système ou vient de la panne d'un certain nombre d'éléments.

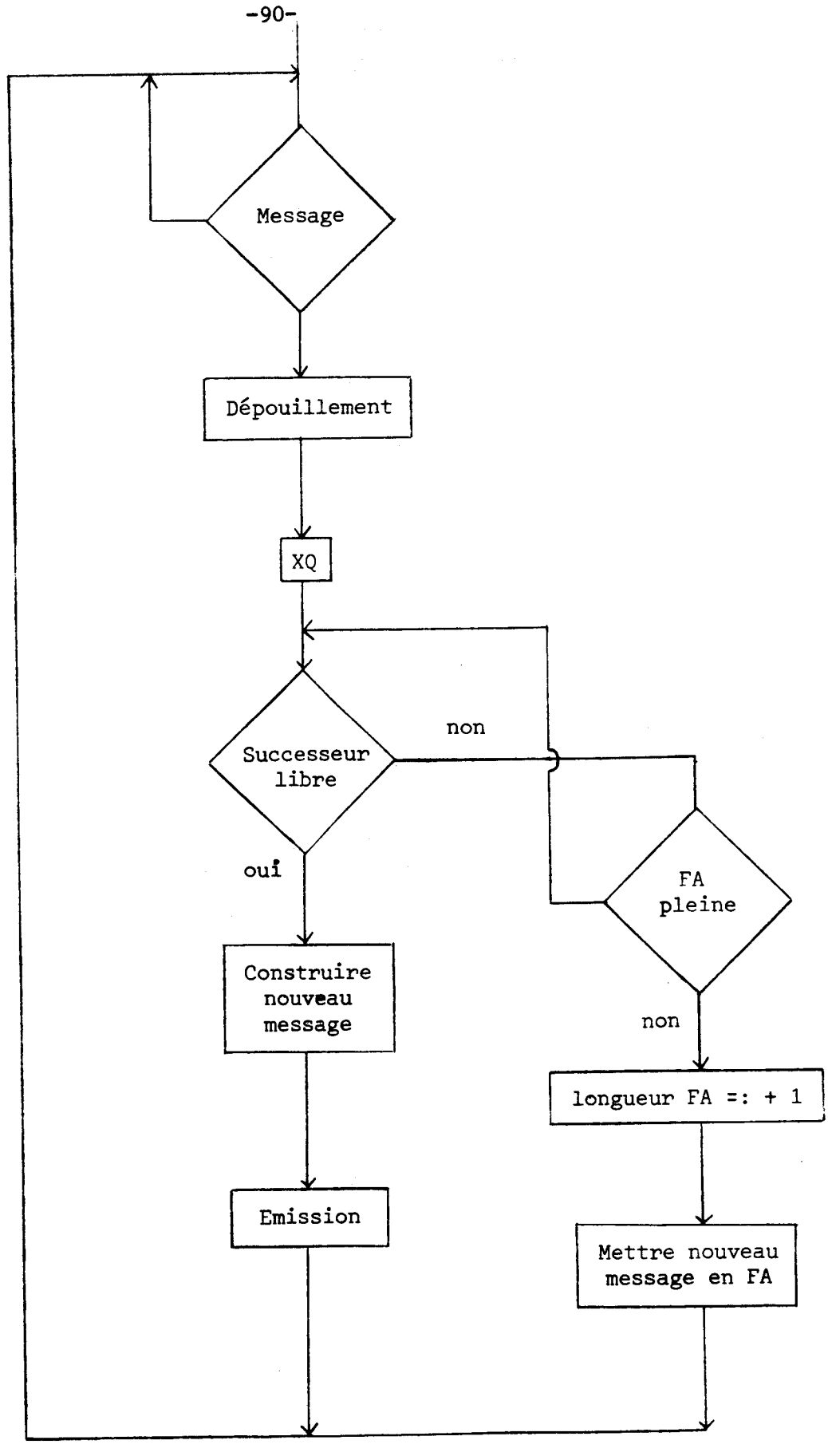
Cet aspect touche donc la dégradation du système. Une première solution consiste à considérer le "nouveau" message comme un message arrivant et de le traiter comme tel.

Il n'y a pas interruption du traitement mais le pipe-line se modifie puisque le prédécesseur de la cellule risque fort d'être placé dans une situation identique et ainsi de suite jusqu'à l'asphyxie s'il n'y a pas eu de libérations nouvelles. Une autre solution, somme toute fort voisine, est de placer le nouveau message en attente jusqu'à la libération d'un voisin successeur. L'asphyxie se produira cette fois quand cette file d'attente interne sera pleine mais tant que cette limite n'est pas atteinte par un grand nombre de cellules, le tableau cellulaire reste opérationnel. Ces deux solutions assurent donc une dégradation progressive et permettent d'encaisser d'éventuelles surcharges sans que le système se bloque ou que soit prise la décision de l'effacement général.



BUS LILLE

Organigramme 1-x-y-a.



BUS LILLE

Organigramme 1-x-y-b.

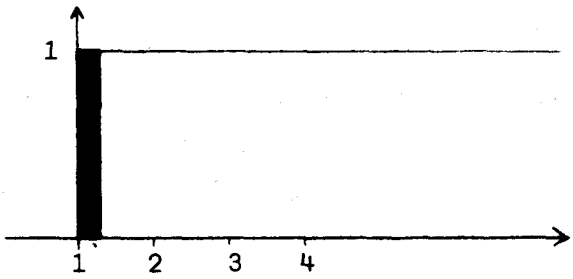
Comme on le voit, à partir du comportement global décrit en 1, un certain nombre d'options peuvent venir se greffer et nous donner des solutions fort diverses suivant la façon dont on décide de régler les trois problèmes que nous avons soulevés, d'autant qu'on peut très bien coupler ces options et associer une file d'attente (b) avec un comportement sans file d'attente (a) lorsque celle-ci est pleine. Il reste néanmoins à bien remarquer que si l'on a enrichi les cellules de façon à repousser les limites ou du volume des traitements ou de la rapidité d'exécution ou de la saturation, une politique saine d'utilisation de l'espace cellulaire consiste à rester nettement en deça de ces limites. Par exemple si on travaille constamment à la limite de la saturation, la panne d'un ou deux éléments risque d'être catastrophique. D'autre part, l'augmentation des temps de service va contrer les gains en volume d'information traité que l'on espérait atteindre.

8 - SCRUTATION DES VOISINS - DÉRIVE

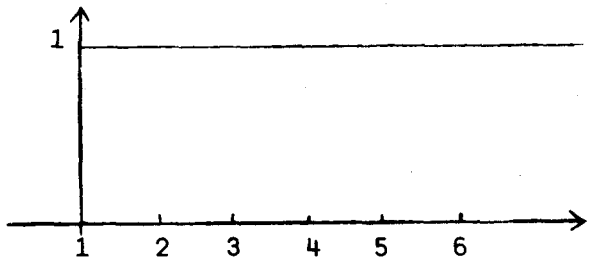
Une routine importante de service est celle qu'on a désigné dans les organigrammes précédents par "recherche et sélection des voisins successeurs libres".

En effet une cellule possède un certain nombre de successeurs possibles (dont elle peut éventuellement faire partie).

Elle choisit parmi eux selon un ordre de scrutation où elle consulte le mot d'état (un seul bit dans le cas présent) de ses différents successeurs. Dans l'hypothèse vraisemblable où on a réparti les entrées dans l'espace cellulaire, un ordre de scrutation immuable pourrait amener une dérive qui la longue déséquilibrerait la charge dans l'espace cellulaire ce qui amènerait des conflits de voisinage dans certaines zones, retardant l'exécution des tâches mises artificiellement en concurrence, du moins si on approche de la saturation. On peut l'illustrer sur les courbes suivantes avec en abscisse le numéro dans la liste de scrutation et en ordonnées la probabilité d'être élu.

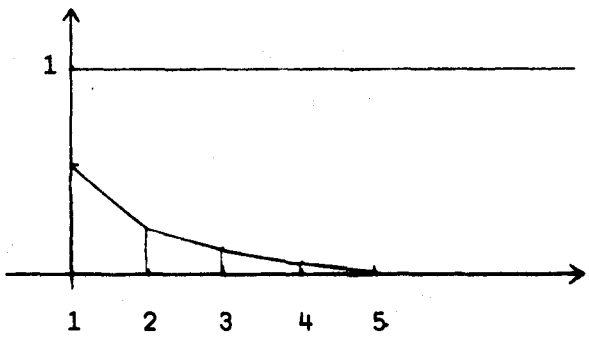


loin de la saturation.



A la saturation : pas de voisin libre.

Près de la saturation, la courbe a une allure intermédiaire



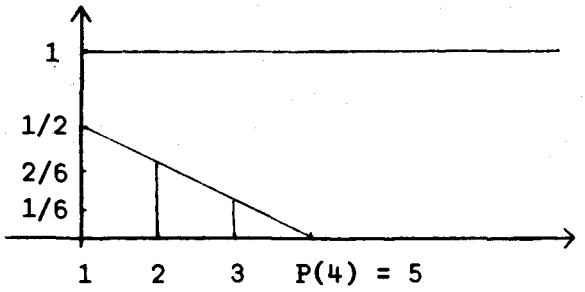
$$\sum_i P(i) = 1.$$

On voit bien que si le numéro 2 occupe toujours la même situation dans l'espace cellulaire par rapport à la cellule initiatrice, le chemin pipe-line va dévier dans cette direction. Ce n'est pas grave si les dérives d'autres tâches ne viennent pas interférer.

BUS LILLE

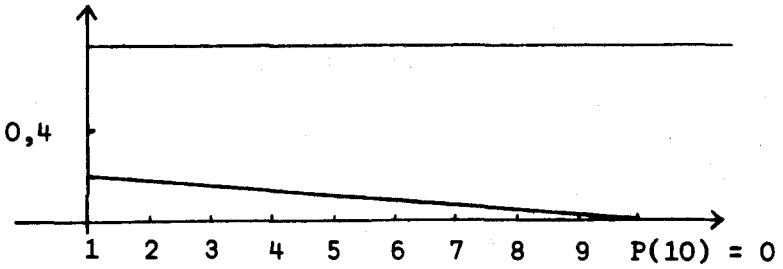
Une approximation de cette courbe est donnée par une fonction monotone décroissante où la probabilité d'être élue est inversement proportionnelle à la position dans la liste de scrutation.

Pour 2D, 3 successeurs.



$$y = -\frac{x}{6} + \frac{2}{3}$$

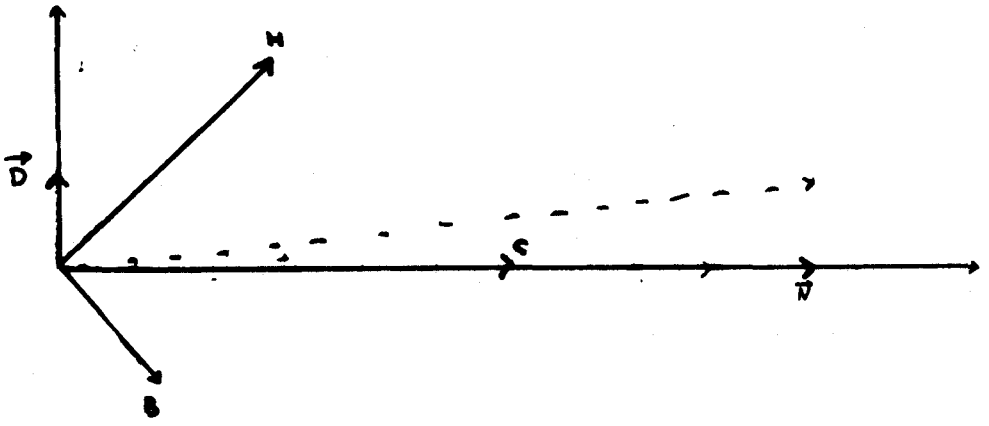
Pour 3D, 9 successeurs.



$$y = -\frac{x}{45} + \frac{10}{45}$$

Ceci nous permet une représentation vectorielle aisée de la dérive apportée par un ordre de scrutation donné.

Ex. : à 2D.



Ordre de scrutation : centre - haut - bas

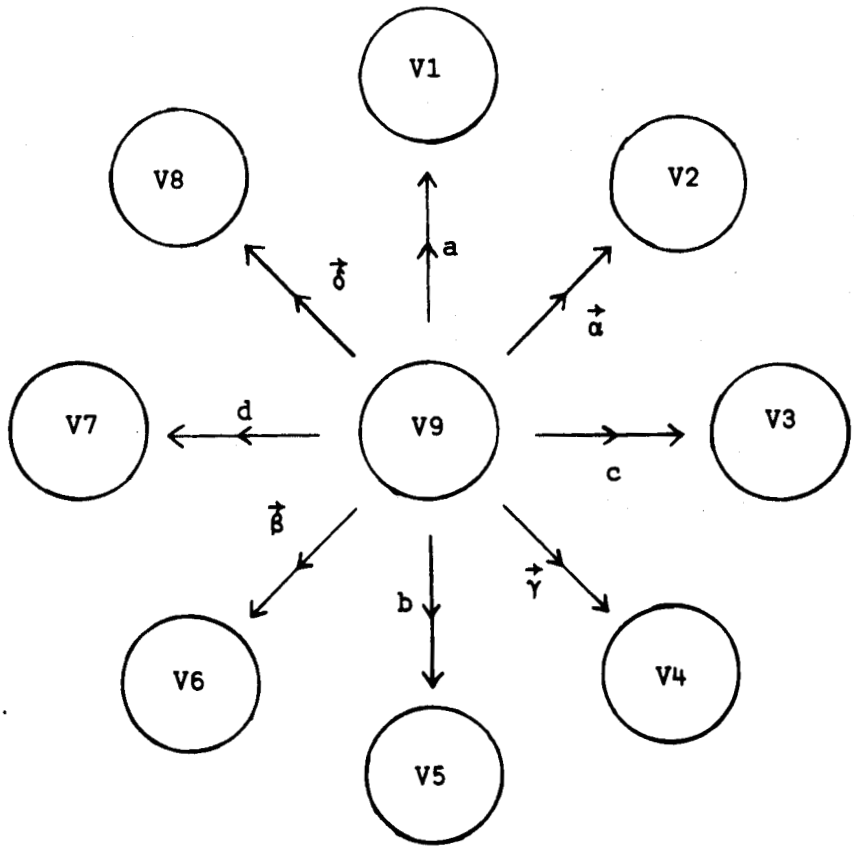
$\vec{V}$: vecteur de tendance de progression,

$\vec{N}$: progression normale,

$\vec{D}$: dérive par rapport à l'axe normal de progression.

Pour les cas de grandes dimensions, on peut donc recourir aux calculs classiques de l'analyse vectorielle pour étudier à priori l'influence d'un ordre de scrutation.

Ex. : à 3D, 9 voisins.



A chaque direction possible de transfert on associe un vecteur unitaire $\vec{a}$, $\vec{b}$, $\vec{c}$. Les coefficients sont inversement proportionnels au rang dans la liste de scrutation.

Par exemple l'ordre V9, V1, V2, V3, V4, V5, V6, V7, V8.

$\vec{D} = 8\vec{a} + 7\vec{\alpha} + 6\vec{c} + 5\vec{\gamma} + 4\vec{b} + 3\vec{\beta} - 2\vec{d} + \vec{\delta}$

$\vec{D} = 4(\vec{a} + \vec{\alpha} + \vec{c} + \vec{\gamma})$

$\vec{a}$ (0,1) $X_D = 9{,}65$

$\vec{b}$ (0, - 1) $Y_D = 4$

$\vec{\alpha}$ $(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2})$

$\vec{\beta}$ $(-\frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2})$ $|\vec{N}| = 9$

$\vec{c}$ (1, 0) $|\vec{D}| = 10{,}4$

$\vec{d}$ (- 1, 0) - vecteur de dérive très fort (vers le coin droit)

$\vec{\gamma}$ $(\frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2})$

$\vec{\delta}$ $(-\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2})$

Autre ordre de scrutation possible.

V9, V8, V3, V4, V7, V5, V2, V1, V6.

$\vec{D} = \vec{\delta} + \vec{c} + \vec{b} + \vec{\alpha}$ $X_D = 1$

$Y_D = 0{,}4$

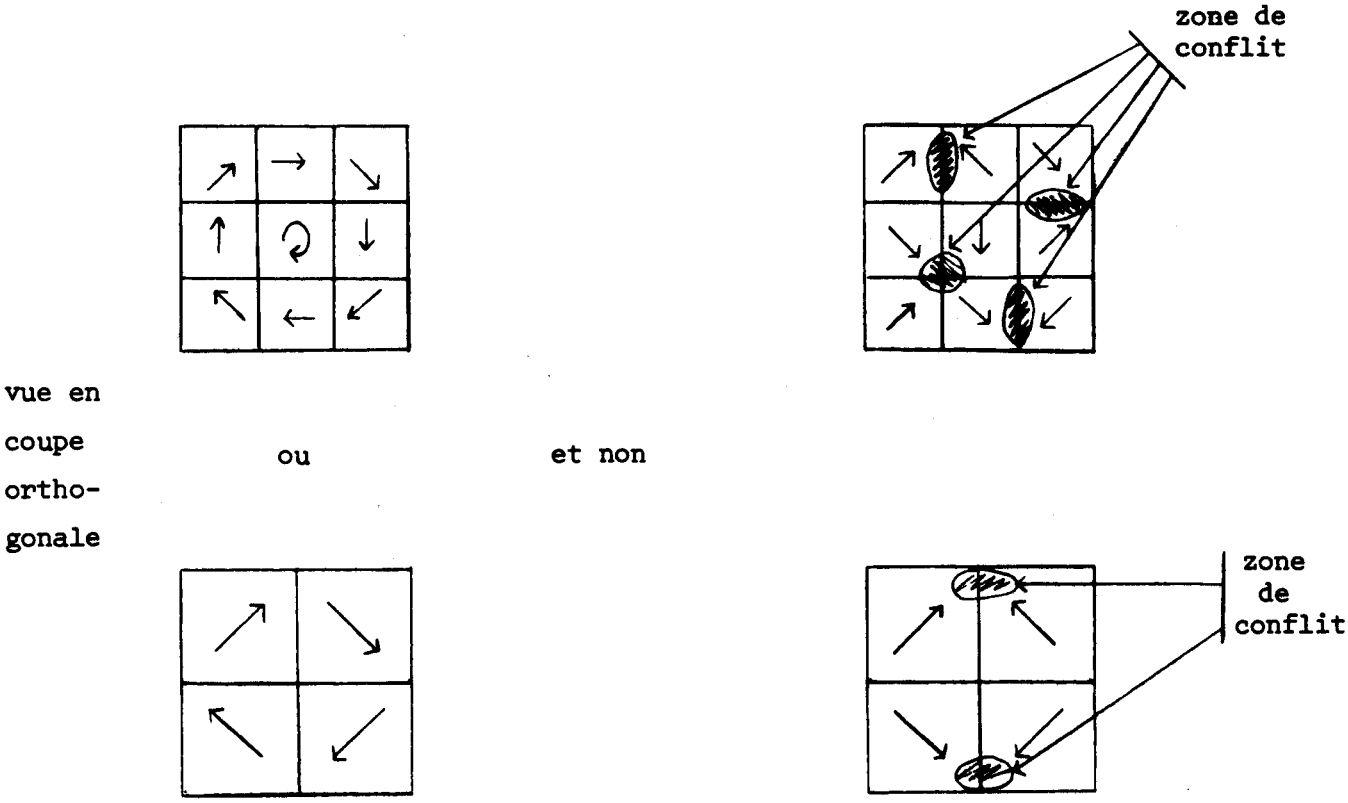
$|D| = 1{,}16.$

$|N| = 9.$

vecteur de dérive faible.

8.1 - STABILISATION STATIQUE

Dans le cas où le travail est réparti uniformément à l'entrée, c'est-à-dire où on choisit les points d'entrée de façon à laisser à chaque tâche le meilleur volume possible, une première politique statique de stabilisation est possible. Soit on se contente de choisir un ordre de scrutation donnant la dérive la plus faible possible soit dans le cas d'un très vaste tableau cellulaire on divise l'espace en zones dont les cellules ont des dérives dans des sens tels qu'il n'y a pas de formation de zones de conflit.



la flèche indique le sens de la dérive.

L'exemple d'ordres de scrutation compatibles dans un partage en quatre zones.

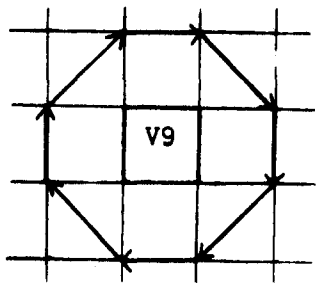
n° apparition dans la liste / n° quadrant	1	2	3	4	5	6	7	8	9
1	V9	V8	V3	V4	V7	V5	V2	V1	V6
2	V9	V2	V5	V6	V1	V7	V4	V3	V8
3	V9	V4	V7	V8	V3	V1	V6	V5	V2
4	V9	V6	V1	V2	V5	V3	V8	V7	V4

Stabilisation dynamique

L'ordre de scrutation n'est pas fixé à l'avance mais se modifie en fonction des dérives précédentes de façon à stabiliser la construction du pipe-line autour d'un axe moyen de progression. Cela suppose en tout état de cause que l'information sur la dérive passe à l'intérieur des messages de données (3 bits supplémentaires à 3D).

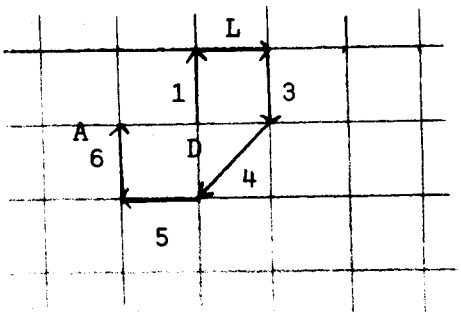
a/ Le tourniquet

Les cellules ont le même ordre circulaire de scrutation mais l'origine (après la position numéro 1 qui est celle de l'axe gauche-droite et qui ne bouge pas) se modifie, en incrémentant la dernière référence autre que V9 (dans un cas à 3D).



Si on a eu une scrutation (V9, V1, V2, V3, V4, V5, V6, V7, V8) ayant eu pour résultat la sélection de V1, la prochaine fois que V9 ne sera pas élu, c'est par V2 que l'on commencera cycliquement la scrutation (on fait + 1 modulo 8 pour trouver la nouvelle origine pour la deuxième position).

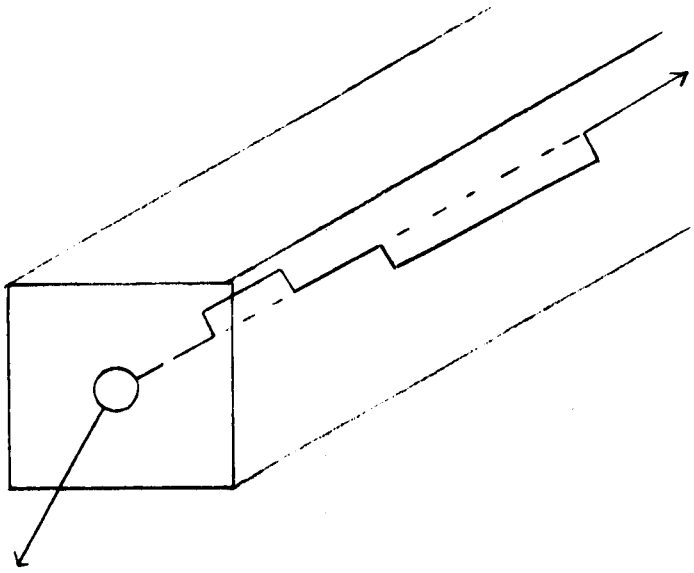
Projection de pipe-line sur la grille d'entrée dans ce cas.

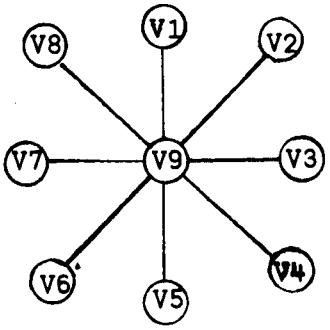


Le hasard a fait ici qu'à deux reprises le 2nd élément de la liste n'a pas été sélectionné (2 et 3).

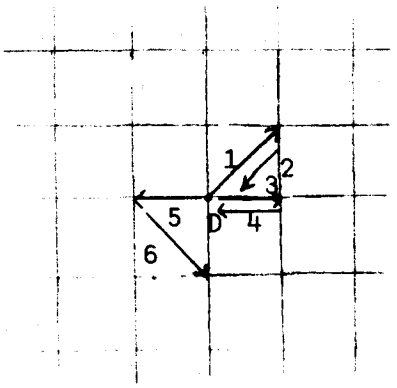
b/ Choix de l'opposé

Cette fois c'est par l'opposé du dernier élu différent de V9 que l'on commence la scrutation (on fait + 4 modulo 8).





Projection du pipe-line sur la grille d'entrée dans le cas b/



A une reprise le second élément de la liste de scrutation n'a pas été sélectionné.
(6)

On peut faire en dernière remarque que l'on peut mélanger politique statique et dynamique de scrutation.

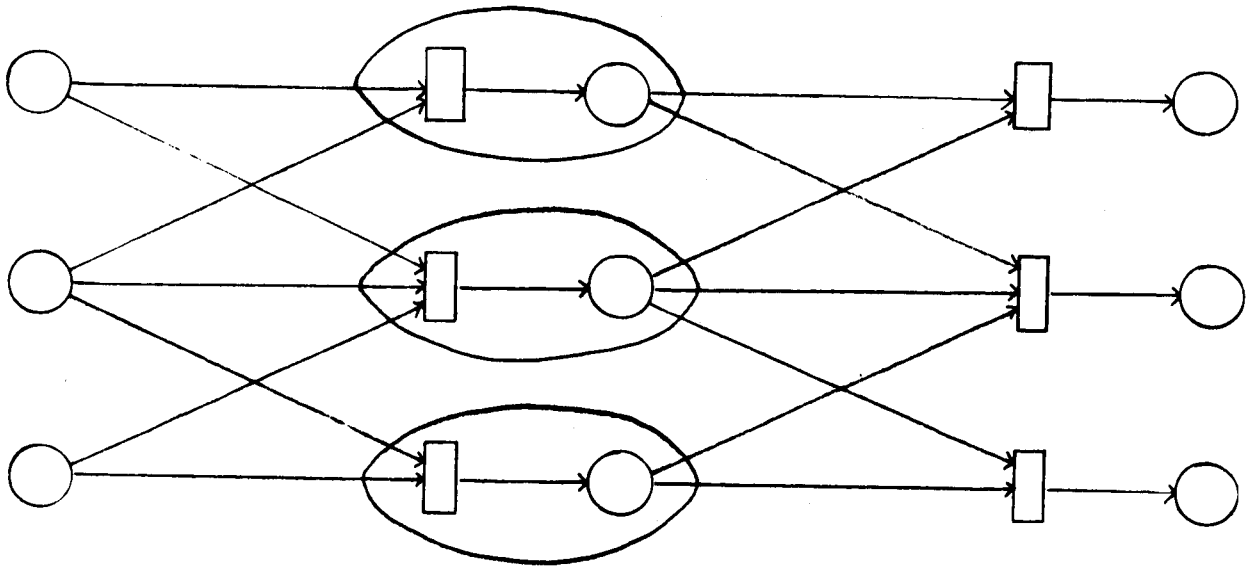
9 - BASE DE L'ARCHITECTURE D'UNE CELLULE

Nous avons réuni maintenant assez d'informations pour proposer une architecture ou du moins un squelette puisque plusieurs types de matériel peuvent réaliser les fonctions demandées. La base du comportement cellulaire tel qu'on l'a défini ici est la possibilité qu'a une cellule de connaître à tout instant l'état de ses voisins successeurs. Ceux-ci ont donc à lui fournir et mettre à jour cette information qui dans le cas présent peut tenir sur un bit. Dans des cas plus complexes ou plusieurs états sont possibles par cellule comme nous le verrons par la suite, cette information sur l'état de chaque voisin serait plus lourde. Autre fonction impérative à assurer : le recueil et l'émission des messages. On va donc supposer l'existence de registres de communications : des registres d'arrivée où les prédécesseurs écrivent et des registres de lecture que lisent les successeurs. Ce mot de registre et cette distinction entre les arrivées et les entrées sont surtout présents pour la clarté de l'explication car dans une architecture réelle, ils peuvent très bien ne pas avoir d'existence physique à condition de créer leurs équivalents logiques par le logiciel interne de la cellule. Par contre, il est impératif d'éviter les conflits d'accès à une cellule convoitée par différents prédécesseurs qui la voient libre. De ce fait, la cellule élisant

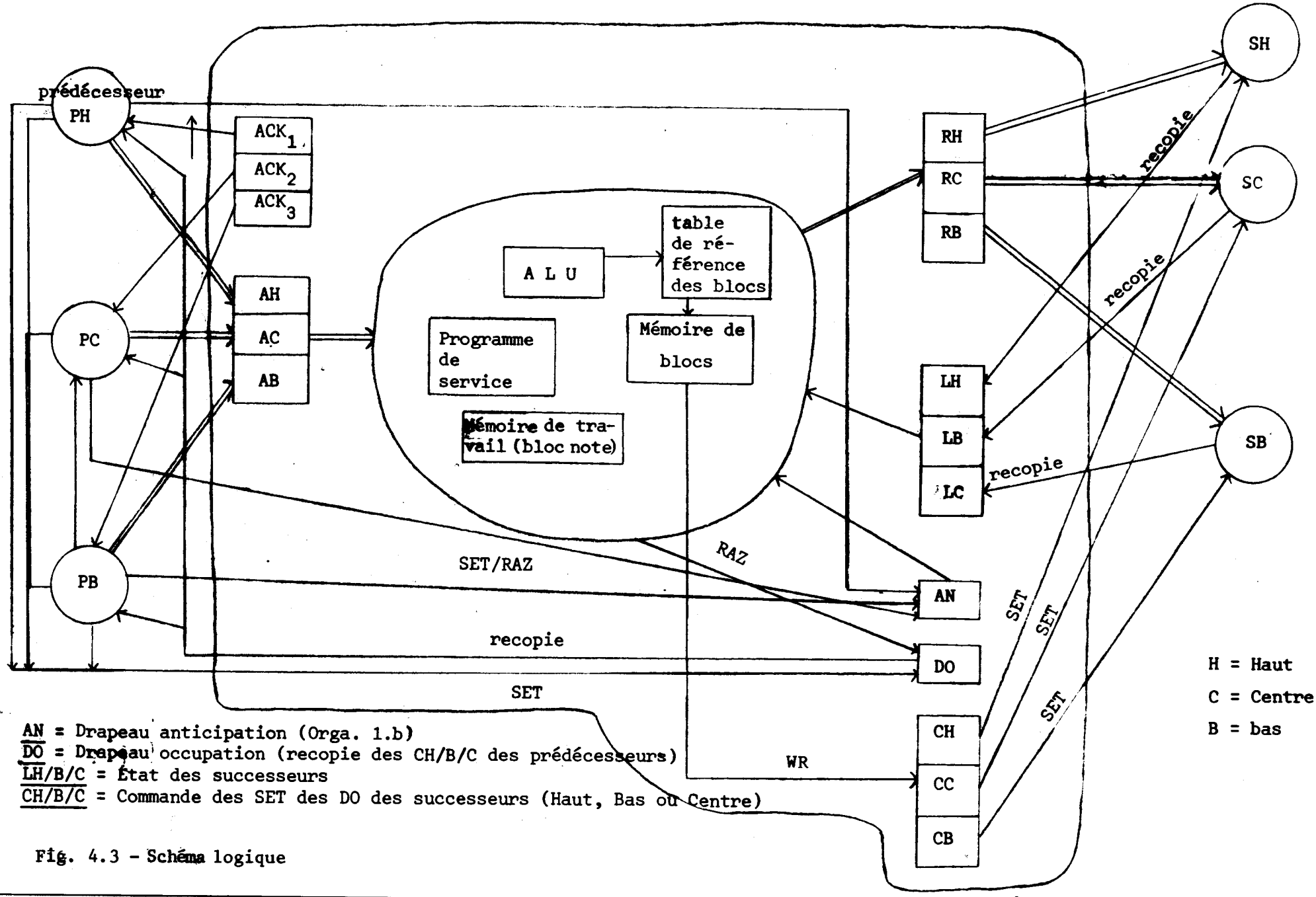
BUS LILLE

un successeur va positionner dans le mot d'état de cette dernière un drapeau indiquant qu'il l'a élu puis attendu l'A.C.K. du successeur. Ceci permet de se prémunir contre les effets d'une lecture quasi-simultanée d'un mot d'état d'une cellule par deux prédécesseurs, chacun voyant la cellule libre. Seul celui qui recevra un A.C.K. après avoir repris le successeur écrira son message, l'autre devant reprendre l'exploration de sa liste de scrutation.

Dans le cas où on choisit de différencier successeurs et prédécesseur, c'est-à-dire que leur intersection est vide, il est normal de concevoir le partage des registres de communication apparaissant sur la figure suivante, un prédécesseur ne pouvant qu'y écrire et un successeur ne pouvant qu'y lire.



BUS LILLE



AN = Drapeau anticipation (Orga. 1.b)
DO = Drapeau occupation (recopie des CH/B/C des prédécesseurs)
LH/B/C = État des successeurs
CH/B/C = Commande des SET des DO des successeurs (Haut, Bas ou Centre)

Fig. 4.3 - Schéma logique

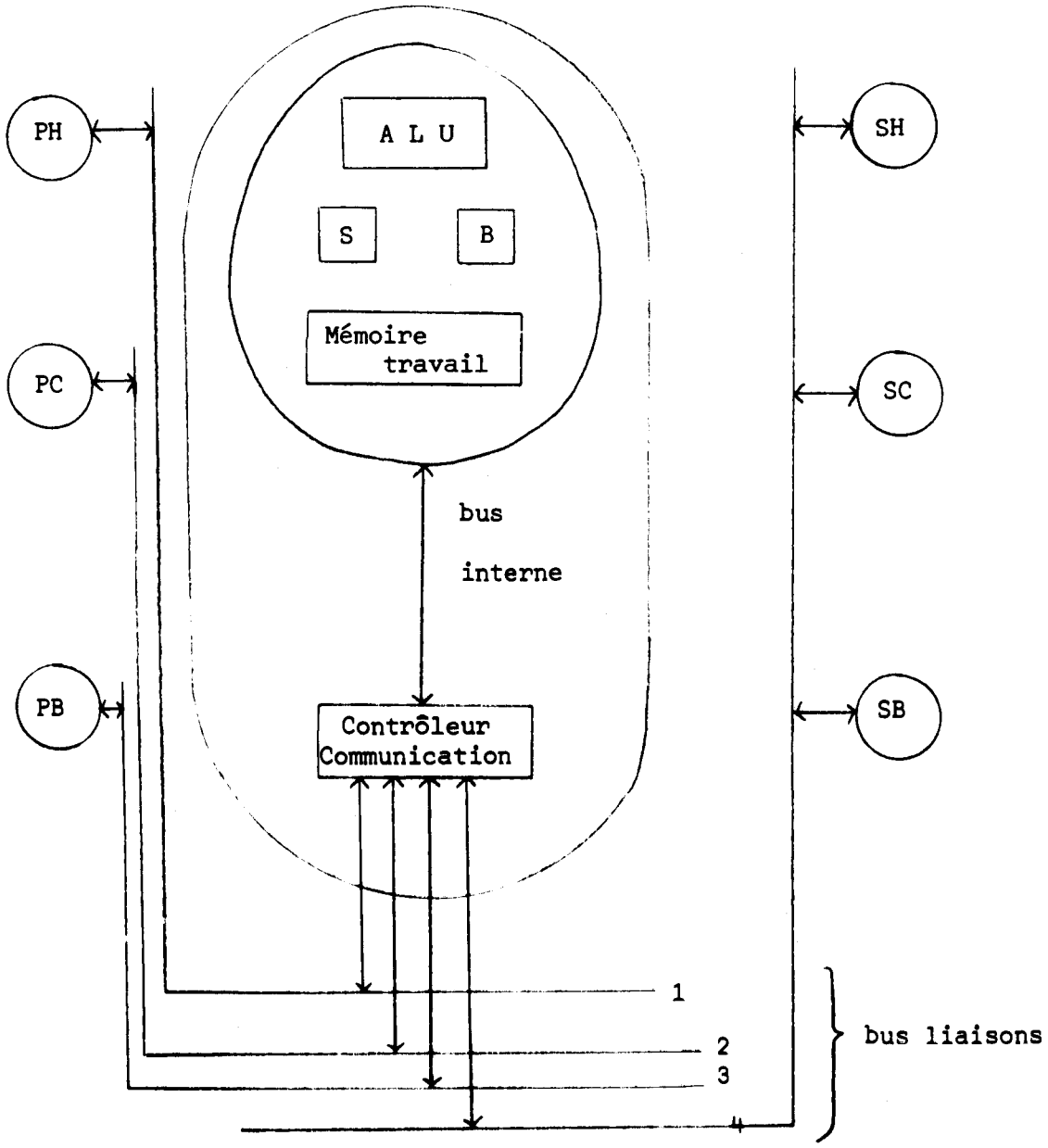
10 - DEUX SOLUTIONS D'IMPLANTATION

10.1 - SOLUTION MULTIBUS

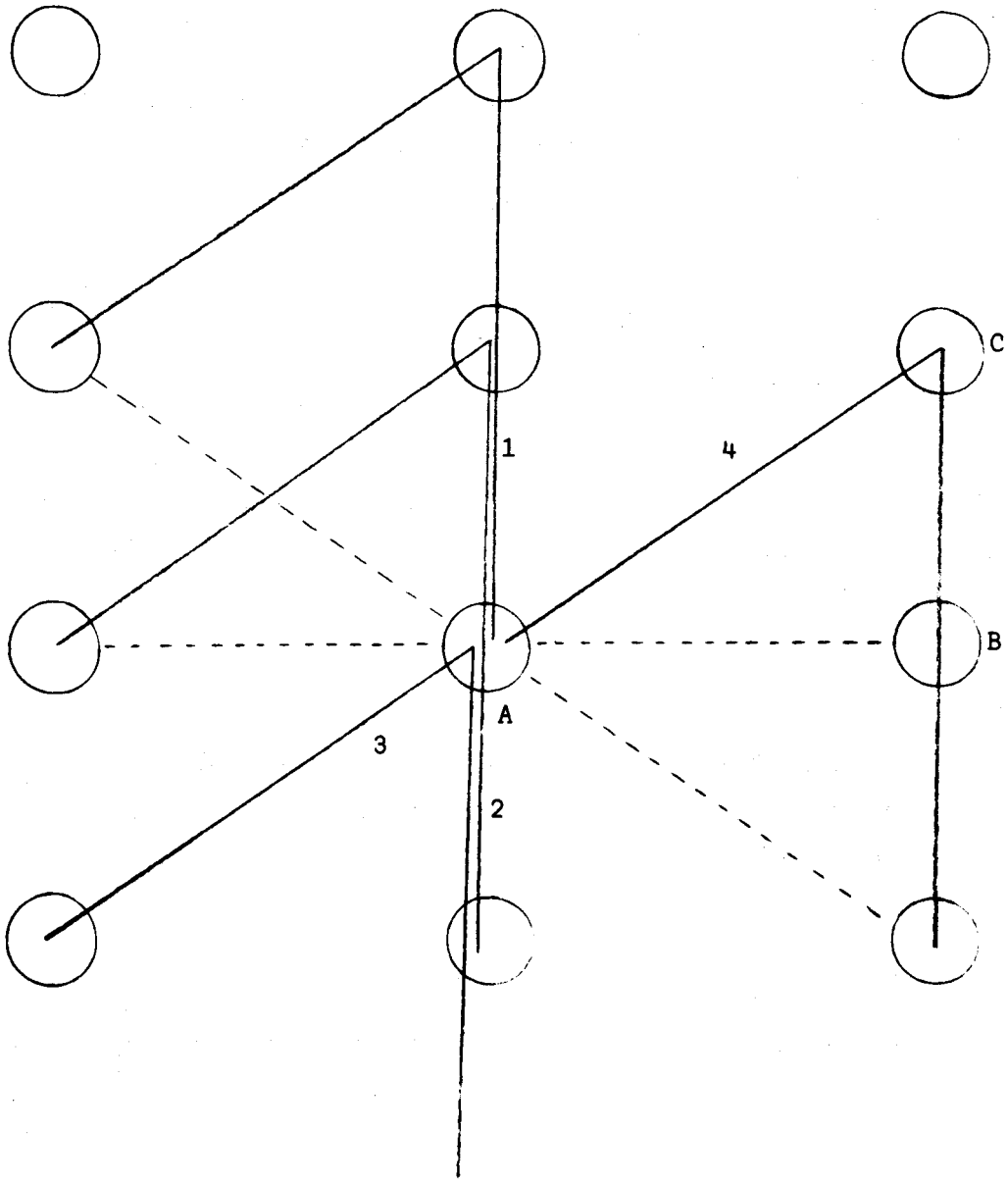
A partir du schéma logique vu précédemment, plusieurs réalisations sont possibles. On peut par exemple alléger le volume du matériel en diminuant les connexions par l'emploi de bus entrelacés, un bus reliant plusieurs cellules et une cellule ayant accès à plusieurs bus. On s'éloigne bien sûr de la cellularité au sens strict car l'extrême de cette politique nous ramènerait à une solution monobus.

Soit par exemple le choix d'un bus pour quatre cellules.

L'architecture de base d'une cellule devient :



La topologie de la figure 4.3. est émulée alors par l'entrelacement des bus de liaisons.



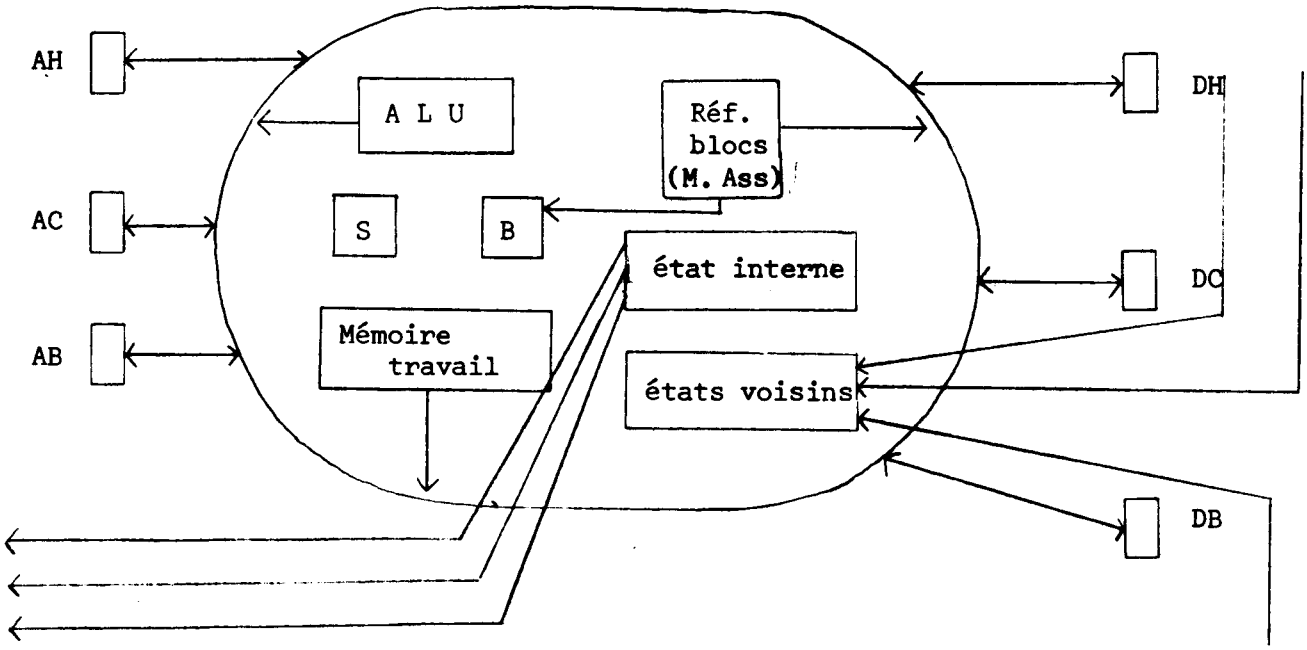
Le bus 4 de la cellule A est le bus 2 de la cellule B, le bus 3 de la cellule C, etc...

BUS LILLE

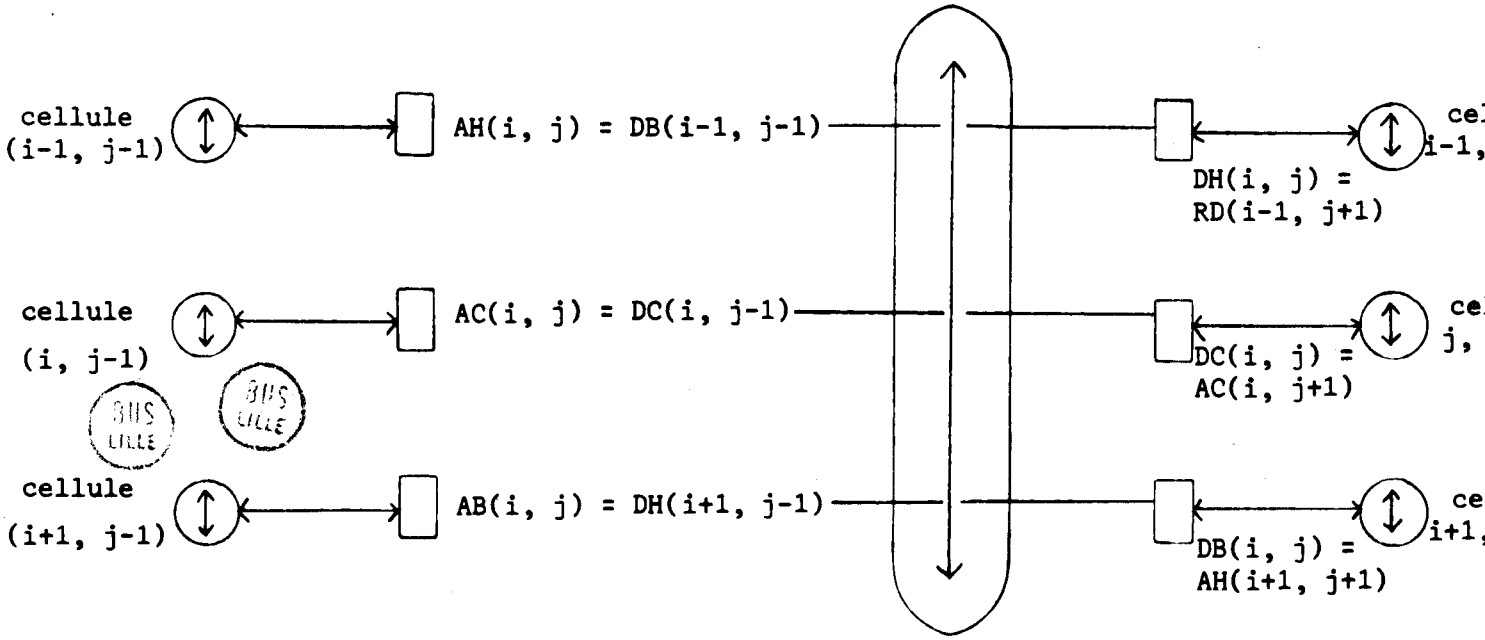
D'autres solutions sont possibles : elles dépendent du débit moyen qu'on veut atteindre dans les échanges inter-cellulaires.

10.2 - SOLUTION CELLULAIRE "STRICTE"

C'est l'approche de l'exemple du D.D.A.



Les registres de communication d'arrivée et de départ (AH, AC, AB, DH, DC, DB) sont directement liés aux bus internes des cellules (de la figure , AH, AC et AB ne font qu'un).



C'est la solution cellulaire type (communication par registres ou mémoire) avec l'emploi d'un microprocesseur comme unité de traitement et de contrôle (service), le schéma ⇕ symbolisant le bus interne du processeur. On suppose implicitement que l'exécution des routines de service et d'E/S était effectué par le processeur de traitement, ce qui est légitime pour des traitements où l'aspect macropipe-line est fortement marqué (courbe

Nous verrons cependant les limitations de cette approche dans le chapitre suivant où on va essayer d'enrichir les cellules et les relations intercellulaires de façon à aborder d'autres formes de traitement sans négliger bien sûr les possibilités du pipe-line.

CHAPITRE 5

CONSTRUCTION DE MACHINES EPHEMERES DANS UN ESPACE CELLULAIRE

5.1 - POSITION DU PROBLEME

Nous allons étudier maintenant la possibilité d'implanter une vaste architecture multi-microordinateur sous forme cellulaire qui ne soit pas fonctionnellement aussi spécialisée que les précédentes. Pour se faire, nous allons donner aux cellules un comportement plus complexe en ce qui concerne les relations de voisinage dans le but d'alléger la tâche du système que nous appelerons moniteur chargé de superviser le comportement de la machine et assurer l'enchaînement des tâches.

Or gérer dynamiquement l'utilisation d'un nombre élevé de processeurs ainsi que l'outil que les relient (tel un réseau de connexion) parait difficile si on n'apporte pas d'importantes restrictions (ex. : les machines MultiSIMD [61]). On va donc montrer qu'en enrichissant le comportement de la cellule, on décharge le moniteur de la tâche d'allocation des "ressources processeurs". Ceci va se faire par la définition d'états, de transitions d'états mais contrairement à l'automate cellulaire où les changements d'états se font sur un front d'horloge, des messages de service valideront ces changements. Il n'est cependant pas de miracle : pour être viable, une telle architecture suppose qu'un pourcentage non négligeable des cellules sont dans un état improductif, c'est-à-dire inactives soit dans un état dit de service sur la définition auquel nous reviendrons. Sous cet éclairage, la principale question subordonnant l'étude puis l'emploi d'une vaste architecture cellulaire et qui ne sera pas résolue ici est la suivante : est-il préférable d'employer un très grand nombre de processeurs "courants" dont une partie seulement sera employée à un instant donné qu'un nombre plus restreint de processeurs très performants reliés par un outil de communication coûteux (5120 points de commutation pour un réseau n 1024 x 1024)

à mettre au point si on le veut réellement souple. A cet égard, pour un système contrôleur, un espace cellulaire apparait comme une boîte noire qui reçoit et fournit des messages "type", contenant des résultats partiels ou définitifs mais aussi des messages "système" pour signaler des erreurs ou des pannes. Le système alimente l'espace cellulaire au fur et à mesure de l'exécution des tâches en fonction du critère de la charge supportée par l'espace cellulaire et qui sera défini en fonction de la topologie, de la puissance des cellules (jeu d'instruction, capacité mémoire) et de leur nombre. Une fois connu ce critère, la gestion externe reviendrait fondamentalement à gérer des files d'attente de tâches prêtes ou en attente, en fonction de leur taille et de leur durée.

5.2 - COMMUTATION DE MESSAGE OU RESERVATION DE CHEMINS

Dans ce qui suit nous présentons une solution avec réservation des chemins. C'est-à-dire que si A et B sont des cellules liées mais non voisines, un chemin (chaîne de cellules contigües) déterminé à l'avance les réunit. Des cellules peuvent appartenir à des chemins différents. Il importe cependant de préciser que les procédures de construction présentées restent valables dans le cas où on choisit d'organiser les communications cellulaires par une méthode de commutation de messages en plaçant un automate de communication interne à chaque cellule et chargé de faire cette commutation suivant un algorithme préétabli et dépendant de la topologie choisie (cf. : par exemple PCC (17 , 18) et X-tree (19)). Il est trop tôt dans l'état actuel de la recherche pour infirmer telle ou telle hypothèse. La réservation des chemins est une solution déterministe car les conflits de routage sont résolus à l'avance et on peut savoir rapidement si le chemin est coupé par exemple mais elle suppose la mise à jour de tables de réservations (tel numéro de message ou tel prédécesseur —> tel successeur). la solution par commutation de message est indépendante d'un chemin donné mais garde-t-elle son intérêt à un niveau aussi microscopique ? En effet s'il existe un chemin unique entre A et B et que ce dernier est détruit, il peut se passer un temps très long avant que A ou B en aient conscience et préviennent le moniteur.

C'est bien sur un cas extrême qui rappelle cependant qu'on doit se prémunir contre un certain nombre d'aléas (message perdu ou message bouclant dans le réseau par exemple) d'autant que l'espace cellulaire apparait comme une boîte noire inviolable vue de l'extérieur (sauf bien sûr l'espionnage de l'évaluateur de performances qu'on aura introduit).

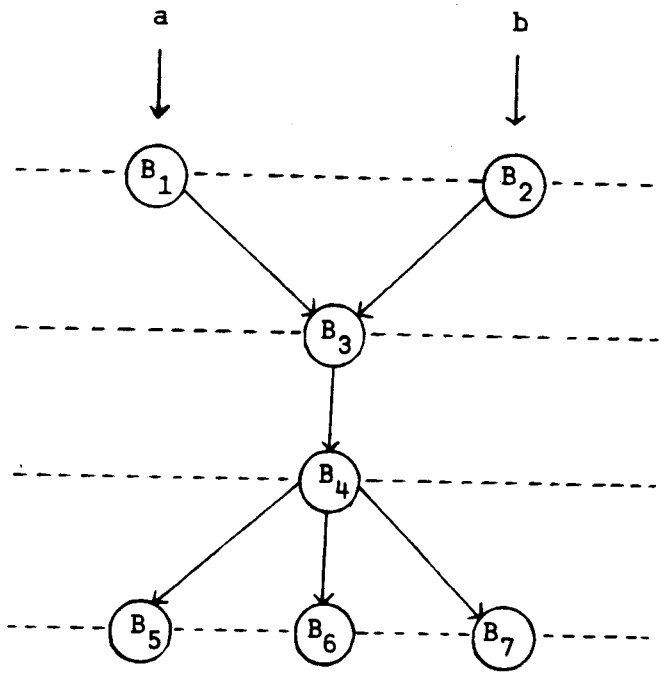
5.3 - PARALLELISME DE BLOC OU MACHINE EPHEMERE

Le macropipe-line portait déjà en soi la notion de parallélisme de bloc puisque le pipe-line est souvent présenté comme un parallélisme implicite. Cette fois nous allons compléter la cellule et les relations intercellulaires de façon à effectuer des traitements où apparait explicitement un parallélisme de bloc. On ne cherchera pas à exploiter le parallélisme optimal d'un programme donné mais seulement celui exploitable dans l'espace cellulaire que nous aurons défini. Cette notion "d'exploitabilité" admet pour paramètre la taille des blocs, les servitudes internes des cellules, celles liées aux relations de voisinage, la topologie (connexité), les performances intrinsèques d'une cellule (temps de cycle, largeur des mots). Il serait donc hasardeux de définir les terrains d'action futurs d'une telle machine sans une étude préalable des divers points cités.

Le concept de machine éphémère est une autre approche (18) plus générale. L'espace cellulaire a été défini comme composé de cellules indépendantes, interchangeables où se mêlent fonctions d'exécution et fonctions de routage mais on peut avoir besoin de construire pour une durée déterminée une structure éphémère réunissant un certain nombre de cellules dans la réalisation d'une fonction commune (ex. : trieur de Batcher, FFT, traitement d'image du type CLIP 3, tableaux cellulaires spécialisés définis dans le chapitre deux etc...). A une cellule sera donc associée soit un bloc soit une "fonction", ce dernier terme recouvrant le précédent puisqu'on peut admettre que plusieurs blocs (au sens du chapitre précédent soient nécessaires pour réaliser une fonction.

Quand des blocs différents peuvent s'exécuter indépendamment sans aucune relation d'antécédence entre eux (sous réserve de la complétude de leurs entrées respectives), on les place sur un même niveau.

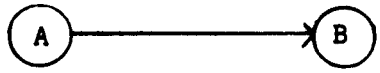
$B_1 : \alpha = f_1(a)$
$B_2 : \beta = f_2(b)$
$B_3 : \gamma = f_3(\alpha, \beta)$
$B_4 : \delta = h(\gamma)$
$B_5 : \omega = g_1(\gamma)$
$B_6 : \theta = g_2(a, b, \gamma)$
$B_7 : \eta = g_3(\gamma)$



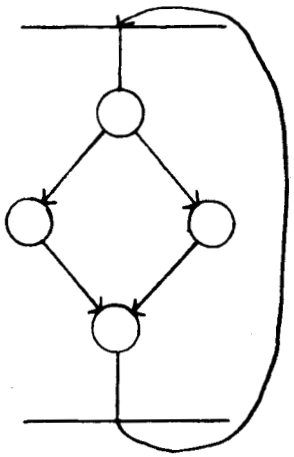
Nous ne considérerons que des graphes multipartis (56), c'est-à-dire, pour ce qui nous intéresse, dépourvus de circuit. Cette restriction vient du fait que nous allons implanter physiquement dans l'espace cellulaire les graphes en affectant de façon biunivoque un bloc ou une fonction à une cellule par une procédure qui ne manipule que des graphes bipartis.

5.4 - MACHINE GRAPHE

Une façon commode et largement employée (5,30,45,53,55) d'exprimer le parallélisme d'un programme est de le faire apparaître sous la forme d'un graphe orienté. Nous nous contentons ici d'une représentation simple où à un noeud correspond un traitement élémentaire effectué par un bloc, à un ou orienté une marque d'antériorité, c'est-à-dire qu'ici B est activable quand



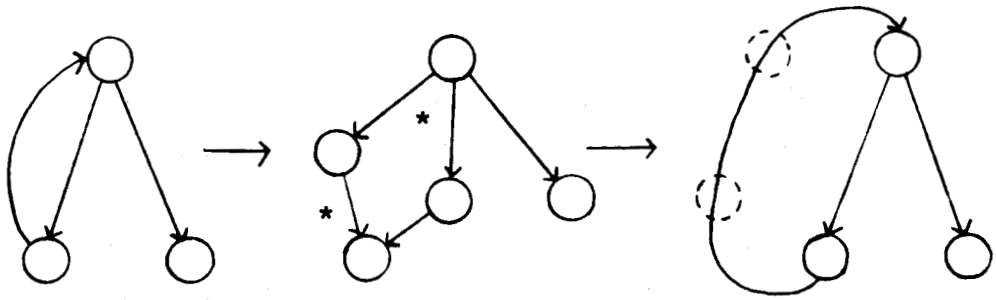
A est terminé. En général, cette antécédence vient du fait que B récupère des résultats calculés par A. Cela n'exclut pas la présence de boucles de programmes à condition qu'elles soient :



a) locales : c'est-à-dire internes à un bloc. On ne le fait pas apparaître sur le graphe.

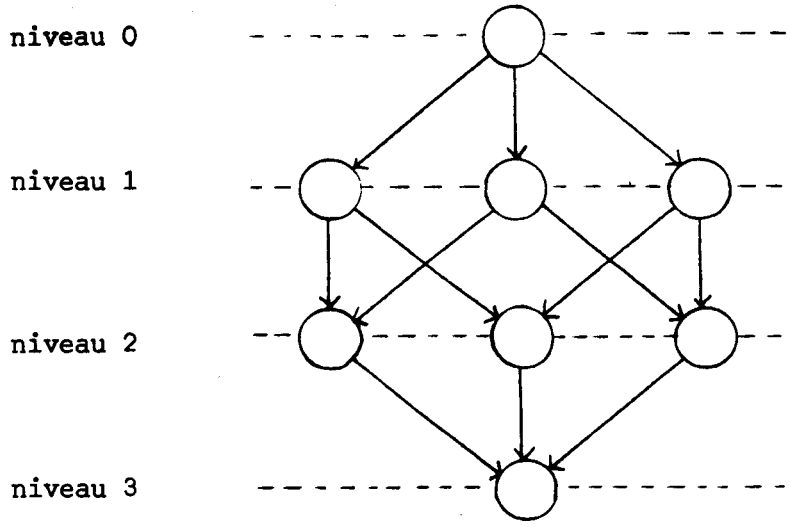
b) globales : le bouclage est fait à l'extérieur de l'espace cellulaire sous contrôle du calculateur hôte supportant le moniteur.

c) Si l'espace est isotrope, on utilise des noeuds (et blocs) fantômes pour rendre le graphe multiparti lors de sa construction tout en permettant l'existence de cycle à l'exécution (message "étoile").



graphe théorique construction exécution

Ce procédé peut s'étendre à la construction de machines éphémères, par exemple, on veut construire une structure multiprocesseur où les processeurs sont les sommets d'un cube : le graphe multiparti sera :



Nous appelerons dans ce qui suit le graphe décrivant l'enchaînement des blocs d'instructions ou symbolisant la structure d'une machine éphémère "le graphe théorique d'exécution". La machine-graphe sera l'ensemble des cellules liées temporairement pour l'exécution du programme. Nous verrons plus loin comment sont traités les cas de figure où un bloc se termine par une proposition conditionnelle.

Cela dépend essentiellement du choix fait pour construire dans l'espace cellulaire le graphe théorique d'exécution :

- S'il se construit au fur et à mesure de l'exécution des blocs, le graphe théorique d'exécution se construit (et se détruit) par parties et n'est jamais contenu en entier dans l'espace cellulaire.

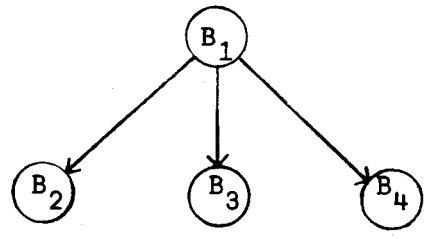
- Si on construit la "machine-graphe" avant l'exécution, le graphe inscrit doit contenir tous les chemins possibles, quitte à ne pas en emprunter certains à l'exécution. On peut penser que ces deux comportements s'appliqueront à des domaines d'application différents.

5.5 - OPERATIONS DE CONSTRUCTION DE LA MACHINE GRAPHE

5.5.1 - Eclatement

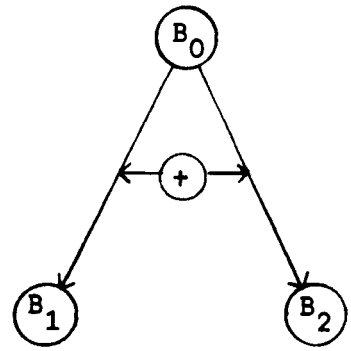
a) Eclatement simple

C'est le cas où plusieurs blocs ont besoin pour leur exécution de résultats d'un même bloc.



b) Conditionnel

B_0 se termine par une proposition **conditionnelle P telle que si** P = 1 on lance B_1, si P = 0 on lance B_2.



Deux cas de figure sont d'ailleurs possibles :

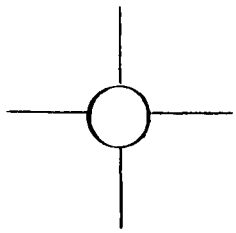
- B_1 ou B_2 exploite les résultats du B_0,

- B_1 ou B_2 n'ont aucunement besoin de résultats acquis par B_0 (sauf l'ordre d'exécution).

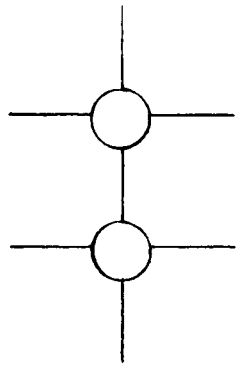
Un éclatement va être essentiellement caractérisé par son degré qui correspond au nombre de blocs initialisés.

Si la connexité en sortie de la cellule employée est inférieure à ce degré, on envisage d'accoler plusieurs cellules de façon à atteindre le degré d'éclatement voulu.

Exemple : à 2 D, **maillage carré.**



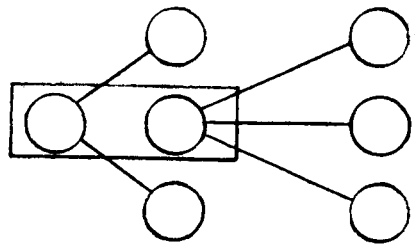
1 cellule : degré = 4



2 cellules : degré = 6

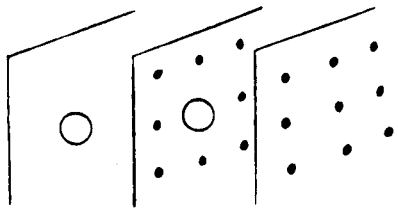
à 2 D, anisotrope.

1 cellule : degré = 3



2 cellules : degré = 5

à 3 D, anisotrope.

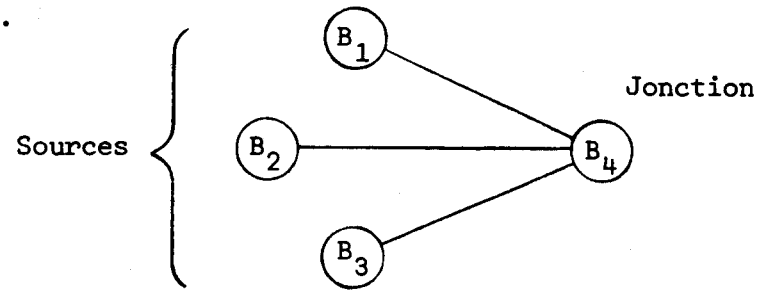


1 cellule : degré = 9.

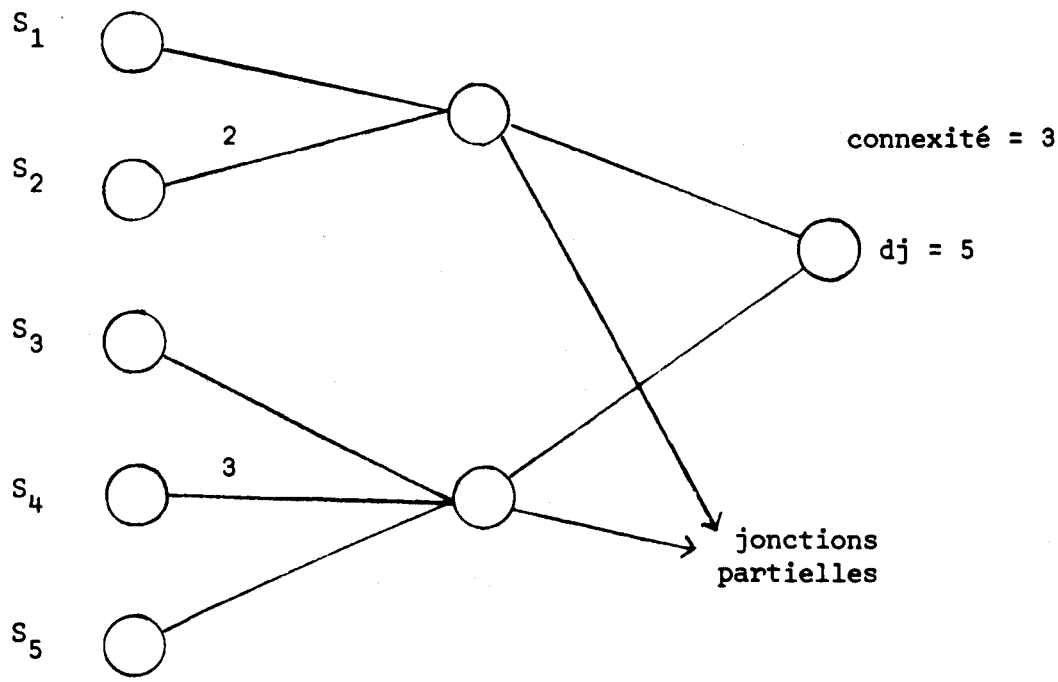
2 cellules : degré = 17.

5.5.2 - Jonction

Ce cas se présente lorsqu'un bloc a besoin pour être exécuté de recevoir des résultats faisant suite à l'exécution de plusieurs autres blocs. Dans un espace cellulaire, cela se traduit par le fait qu'une cellule doit recevoir des messages émis par d'autres que nous appelerons les "sources" dans ce qui suit.



Une jonction aussi peut être caractérisée par son degré, i.e. le nombre de sources. Là encore, si le degré de la jonction est supérieur à la connexité de la cellule, on peut effectuer des jonctions partielles dans un certain nombre de cellules "relais".



5.6 - IMPLANTATION DES OPERATIONS DE CONSTRUCTION

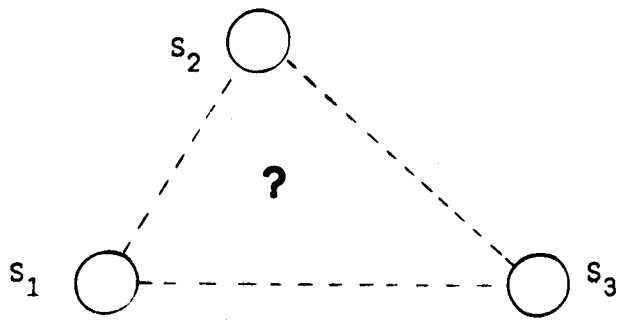
Rappelons bien que nos cellules sont des entités indépendantes et que leur connaissance du monde extérieur se limite à ce que leurs voisines leur communiquent. Dans le cas le plus simple du chapitre précédent, une cellule ne connaissait ses voisines que sous l'information libre/occupé. Dans ce qui suit dans le paragraphe, l'espace cellulaire est considéré comme défini et n'exécutant qu'une tâche, il n'y a pas de conflit entre construction et exécution.

5.6.1 - Eclatement

Suivant le degré de l'éclatement demandé et le nombre de successeurs libres disponibles, une cellule va réaliser immédiatement l'éclatement ou initialiser un certain nombre de cellules dans un état subalterne de relais de façon telle que le degré cherché soit atteint. L'opération est simple et ne demande pas qu'une information complexe soit échangée entre voisines. Ceci se comprend car une cellule unique est chargée du lancement de l'opération.

5.6.2 - Jonction

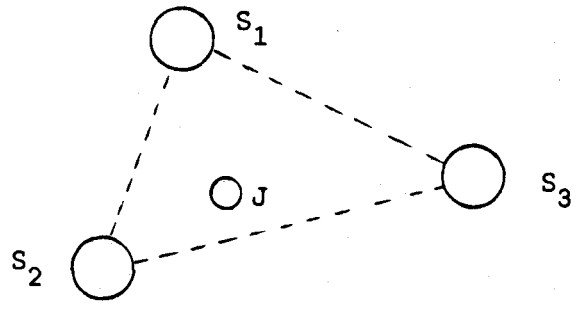
On doit envisager le cas où il existe un certain nombre de sources non directement voisines.



Il faut donc découvrir une jonction possible et une seule de préférence à partir des sources.

5.6.2.a - Une solution centralisée

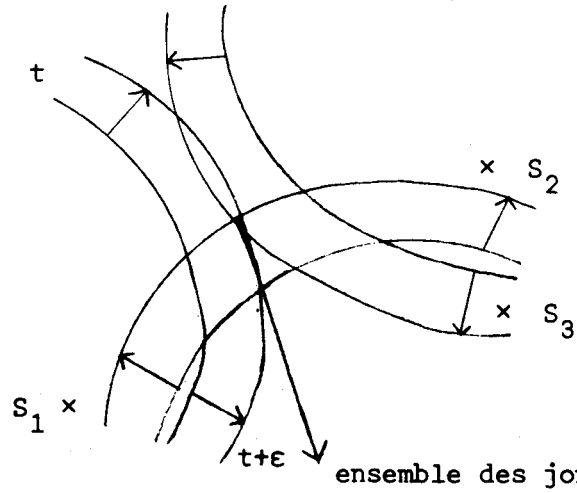
Comment procéderait un contrôleur central ? A partir des positions de S_1, S_2,..., S_m il déterminerait quelle est la meilleure position pour la jonction et désignerait la cellule l'occupant ou en étant le plus près possible. Il calculerait ensuite les chemins joignant chaque S_i à J.



Le critère de choix pour la meilleure position de J pourrait être de prendre une source, ou mieux de minimiser la longueur des chemins de données (ici J serait le centre de gravité de S_1, S_2, S_3). D'autres considérations, plus générales, pourrait guider ce choix. Comme on a vu qu'il était hors de question d'employer un contrôleur central, vouloir plaquer cette solution dans l'espace cellulaire pourrait paraître attrayant. Il faudrait désigner une cellule qui soit l'arbitre et exécute l'algorithme centralisé de désignation de la jonction. Ceci implique que cet arbitre (qui pourrait être une source désignée arbitrairement comme celle ayant traité le bloc x par exemple) possède les positions des différentes sources. Celles-ci n'étant pas connues à l'avance et ignorant de leurs côtés la position de l'arbitre, elles doivent diffuser ces informations de cellules voisines en cellules voisines jusqu'à ce que l'arbitre recueille ces positions, exécute l'algorithme donnant la position optimale de la jonction, prévienne ensuite les autres sources, la cellule choisie pour la jonction, détermine les chemins des sources à celle-ci. Cette procédure est de ce fait en contradiction avec nos objectifs car il suffit de quelques cellules en panne pour rendre cette procédure très lente. Imaginons par exemple que la jonction choisie soit en panne ou qu'un chemin arbitrairement calculé pour y accéder soit coupé !

5.6.2.b - Procédure de recherche et validation de la validation de la jonction basée sur la notion d'état et de basculement d'état sur message de service

La procédure fondamentalement la plus simple pour trouver une jonction est que les sources émettent dans toutes les directions des messages (opérande partiel, degré de la jonction, numéro de la jonction) que stockeraient automatiquement les cellules touchées par ce message, comparant le nombre d'opérandes partiels différents stockés et le degré de la jonction. Une cellule s'apercevant qu'elle possède le nombre d'opérandes partiels requis passerait en exécution. L'inconvénient majeur de cette simplicité est qu'à priori de nombreuses jonctions vont se déclarer **candidates**. On trouverait donc un grand nombre de cellules exécutant à divers stade un même traitement puis l'émission de messages identiques.



ensemble des jonctions possibles à $t + \varepsilon$ = cet ensemble grossit tandis que ε grandit.

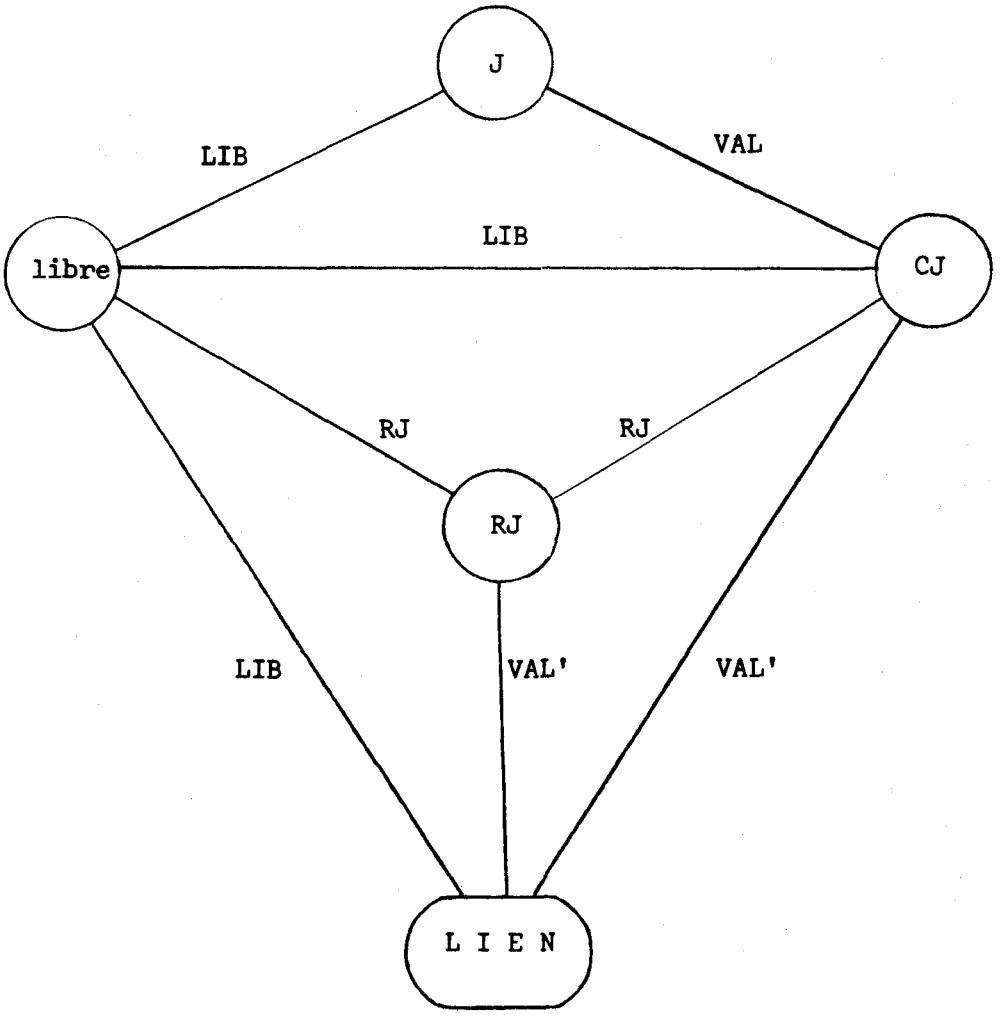
On va donc **compléter ce modèle** tout en gardant l'idée de base afin de ne valider qu'une seule jonction et arrêter la phase de recherche de la jonction. C'est ce que va permettre la "procédure simple de validation de la jonction" (le mot simple signifie qu'on peut en trouver d'autres optimisant certains points précis mais plus complexes à implanter).

Cette procédure permet :

a/ De trouver une jonction, c'est-à-dire de trouver une cellule qui appartienne à l'intersection des descendances des diverses sources.

b/ De valider une jonction, c'est-à-dire désigner une jonction et une seule pour : <u>premièrement</u> = recevoir les divers opérandes partiels ==> conversation des chemins entre source et jonction.
<u>deuxièmement</u> = exécuter le bloc prévu.

Le diagramme d'état suivant va nous permettre de visualiser simplement la procédure.



On y voit les différents états pris par la cellule au cours d'une phase de construction de la machine graphe.

Une cellule source va emettre dans toutes les directions un message dit "Recherche jonction" (abréviation RJ) qui **contient** :

1° Une référence de l'opérande partiel (simple numéro) détenu par la source ($1 < r < d$).

2° Degré de la jonction.

3° Numéro de la jonction (en cas de plusieurs recherches jonctions à l'intérieur d'une même tâche) : peut être la référence du noeud correspondant dans le graphe théorique d'exécution.

Une cellule qui reçoit un message RJ va le ranger dans une table de liaison TL1 où, pour chaque numéro de jonction, on fait correspondre la référence d'opérande partiel et le code de la *cellule prédécesseur* l'ayant envoyé.

Ceci permettra de remonter ensuite les chemins en sens inverse.

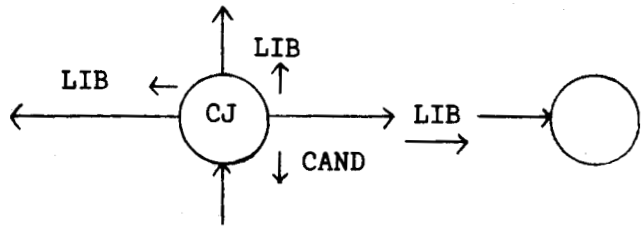
Jonction x degré = d_1	Réf. O P	Emetteur
	1	EST
	2	NORD
		.
		.
Jonction y degré = d_2	1	SUD
	2	EST
		.
		.

On assure l'unicité des chemins en ne retenant qu'un seul emetteur pour une même référence : le premier l'ayant émis.

A chaque nouvelle réception d'un message RJ, la cellule consulte sa table de liaison et sait si le degré de la jonction est atteint. Si oui, elle devient alors une candidate à la jonction (état CJ).

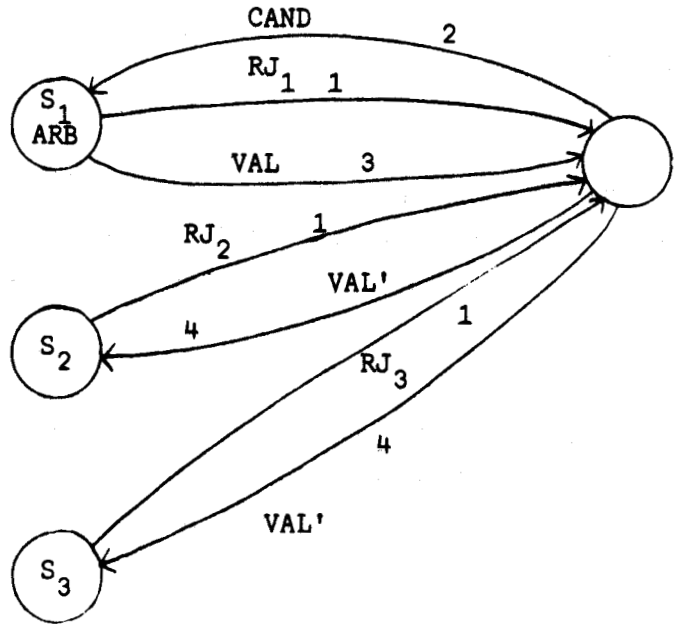
Validation de la jonction

Dès l'apparition d'une cellule candidate à la jonction, la phase de recherche jonction devrait s'interrompre. En fait, comme on va le voir cette interruption n'est pas immédiate. La cellule CJ va emettre un message de libération LIB ayant pour paramètre le numéro de la Jonction vers d'éventuels successeurs initialisés dans cette recherche de la jonction. En un premier temps, seuls les successeurs de CJ vont se libérer de la recherche de la jonction.



Parallèlement la cellule va emettre un message marquant sa candidature à la jonction en direction d'une source, désignée comme arbitre suivant une convention prise avant l'exécution, à la compilation. C'est en effet la solution la plus pratique pour assurer l'unicité de la décision.

Exemple : Jonction de degré 3.



Ce "message" (qui peut se traduire par la mise à jour de drapeaux dans une cellule par une autre) sera appellé CAND. Il remonte jusqu'à la source ARBITRE par les cellules RJ grâce à leur table de liaison TL1 interne à chaque cellule. L'emetteur du CAND est repéré et placé dans une seconde table de liaison TL2. On dira alors que la cellule est placée dans un état LIEN pour la jonction référencée, caractérisée par la mise à jour de deux tables de liaison :

TL1 (réf. jonction) donne le prédécesseur
TL2 (réf. jonction) donne le successeur
} pour un couple (réf. jonction, réf. opérande) donné.

Le signal CAND (réf. jonction) arrive donc ainsi jusqu'à l'arbitre qui le retient s'il est le premier (on applique encore la règle du 1er demandeur - 1er servi). L'arbitre renvoie alors un message de validation (noté VAL) vers le candidat élu et un message LIB (réf. jonction) qui va parcourir sa descendance en RJ pour la jonction référencée et la libérer (sauf la branche composée de cellules dans un état LIEN). La candidate qui reçoit le VAL le répercute vers les autres sources. Cette fois c'est ce dernier signal (toujours référencé par le numéro de la jonction) qui provoque le passage de l'état RJ à l'état LIEN pour la référence donnée (constitution de la 2ème table de liaison). Les sources recevant le VAL libèrent les autres descendants de la RJ (réf. jonction).

Notons que c'est pour éviter le cas, où la construction d'une jonction empêcherait celle d'une ou plusieurs autres qu'une cellule peut simultanément être RJ pour différentes jonctions et même LIEN pour certaines et RJ pour d'autres.

Nous verrons plus loin que si un automate interne à chaque cellule est chargé du contrôle des messages "courants", on peut permettre de surcroît à une cellule d'être dans un état actif pour une branche du graphe

et dans un état de service (LIEN) pour une autre. Cela impliquera en contre-partie certaines contraintes au niveau de l'écriture des messages, plus ou moins compliquées selon qu'on permet des recoupements simples (en un point) ou le partage de chemins.

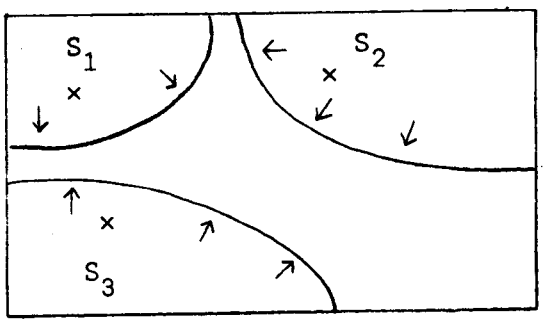
La gestion des communications intercellulaires par une politique de commutation de messages est l'extrême de cette dernière proposition. La cellule est dotée d'un automate chargé de gérer la commutation des messages. Les messages RJ contiennent les adresses physiques des cellules sources (ce qui n'était pas obligatoire précédemment). Les cellules acquérant les messages RJ les traitent de la même façon. Quand une cellule se déclare CJ, elle renvoie à l'arbitre un message CAND avec l'adresse de ce dernier et la sienne propre. L'arbitre élit la première cellule ayant répondu et lui renvoie le message VAL. Celui-ci, une fois acquis par la CJ élue est répercuté vers les autres sources par un message à leurs adresses et contenant le VAL et l'adresse de la cellule réalisant le jonction. Pendant l'exécution, les sources adresseront dans cette cellule. La différence essentielle entre ces deux comportements reste que dans le premier cas les chemins sont connus (pas d'adressage physique) et inconnus dans le second (besoin de l'adresse réelle).

5.7 - CONTRAINTES PHYSIQUES

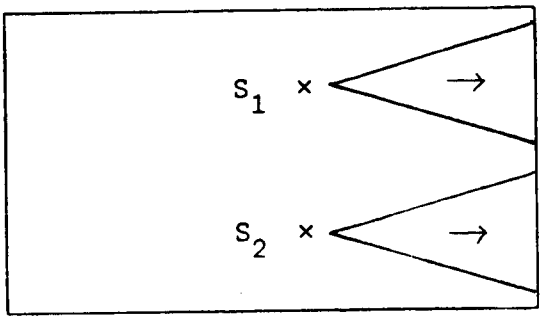
La procédure précédente est valable à deux conditions placées en préambule (5.5). Voyons comment les réaliser dans une implantation réelle.

5.7.1 - Espace cellulaire illimité

Dans un espace isotrope, c'était une condition superflue, les descendances de cellules quelconques étant non vides.

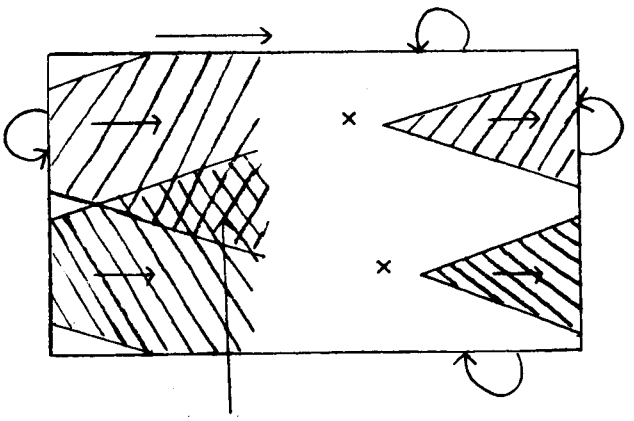


Dans un espace anisotrope, il n'en irait pas forcément de même :



$D(S_1) \cap D(S_2) = \emptyset$

Pour assurer cette condition, nous nous placerons donc dorénavant dans des structures torroïdales pour réaliser cette condition.



Notons que de tels cas de figure doivent être considérés comme limites et rarissimes dans un contexte normal d'utilisation.

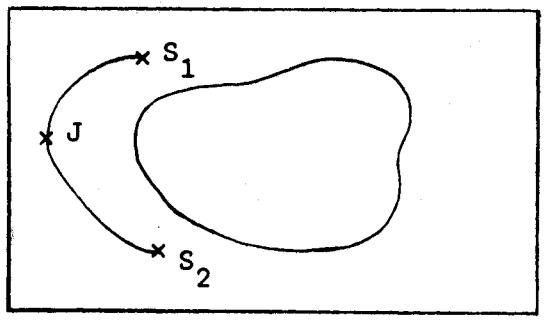
BUS LILLE

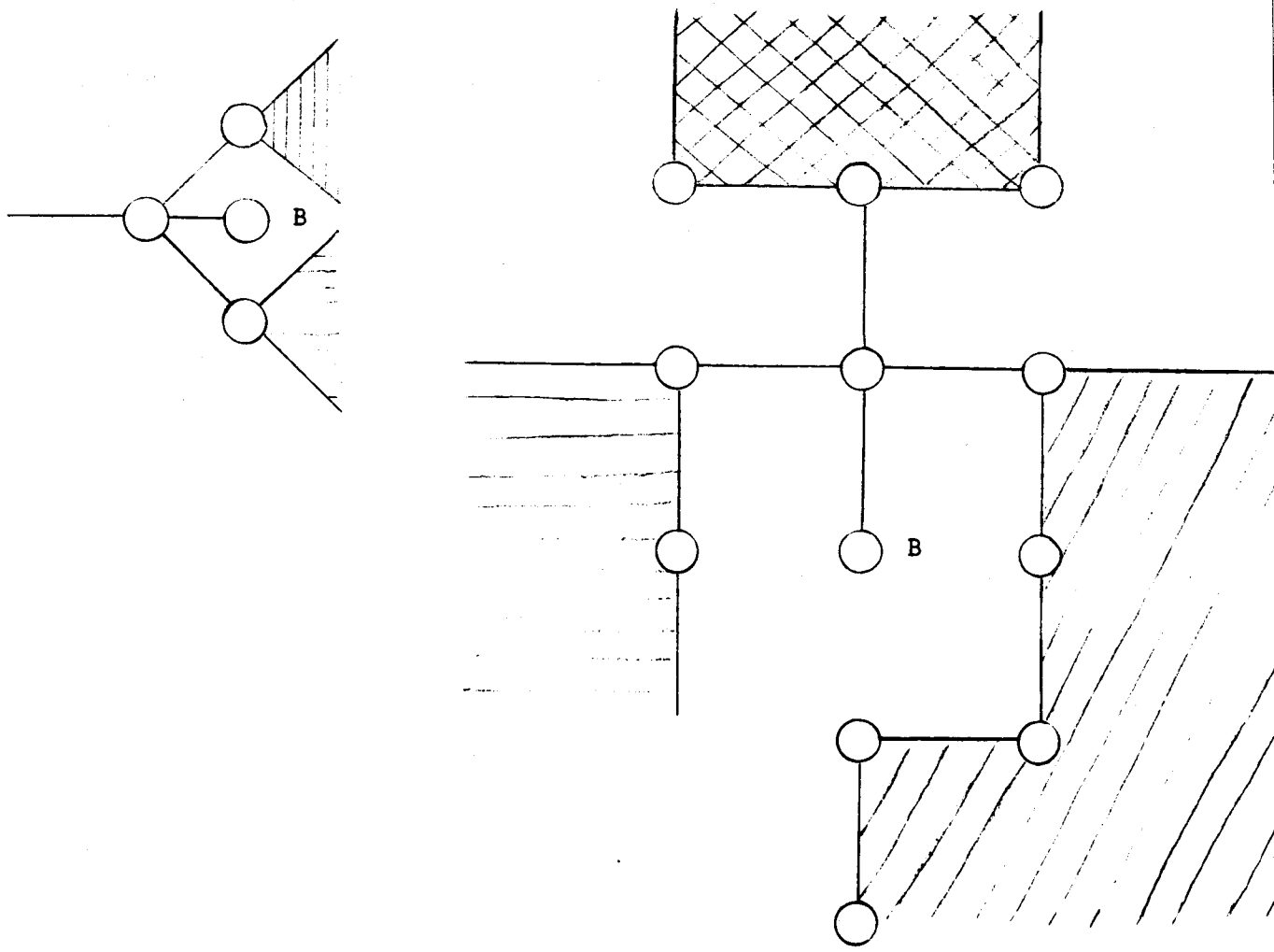
5.7.2 - Absence de conflit entre exécution et construction

Nous avons implicitement supposé dans le paragraphe précédent que toutes les cellules étaient disponibles pour la construction du graphe corvéables à merci en quelque sorte.

Ce n'était qu'une hypothèse simplificatrice destinée à faciliter l'exposé de la procédure de recherche et validation de la jonction. Abordons maintenant la cohabitation des phases de construction et d'exécution.

Bien que nous n'avons fait aucune hypothèse sur l'implantation physique, des études ont montré dans le cas du P.C.C. (18) l'importance de la minimisation de la longueur des chemins de données dans un système multiprocesseur utilisant un réseau cellulaire. Cette étude se référait à un maillage carré mais il y a de fortes présomptions pour qu'on trouve des résultats analogues dans diverses topologies. Or si lors de la construction d'une machine graphe, on considère comme occupé et inaccessible les cellules en exécution de la tâche précédente ou déjà requise par la tâche en construction, il est certain que dans de nombreux cas on va augmenter dans d'énormes proportions la longueur des chemins de données. si ce n'est aboutir sur des situations de blocage.



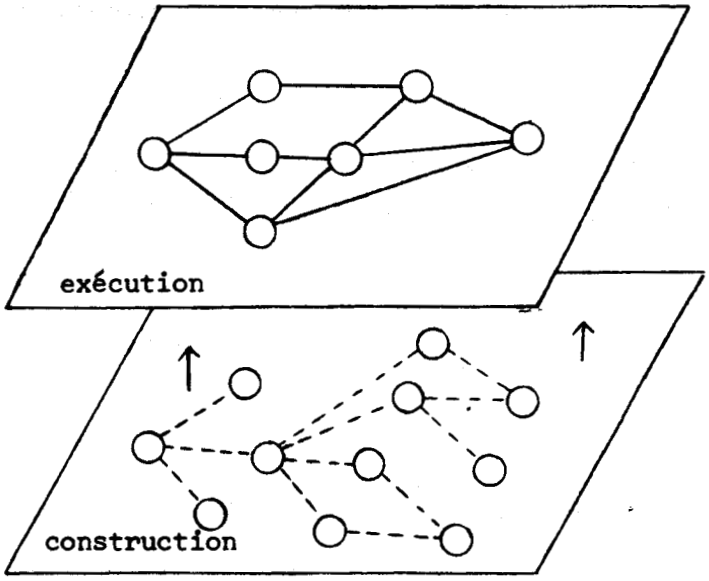


Nous proposons de permettre la cohabitation des actions de construction et d'exécution et éliminer les risques d'autoblocage par blocus involontaire d'une cellule par ses voisines.

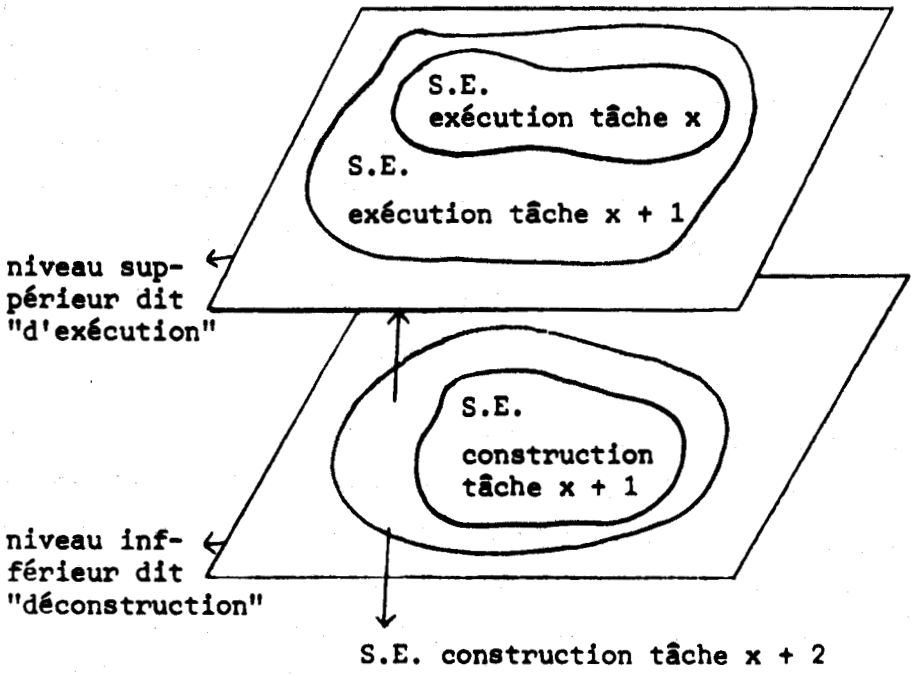
On considère l'espace cellulaire comme pouvant se disséquer en deux sous-espaces superposés :

- Un sous-espace d'exécution de la tâche x.
- Un sous-espace de construction de la tâche x + 1.

BUS LILLE



Sous espace construction (tâche x + 1) = sous espace exécution (tâche x). Tel que présenté là, ce partage apparait statique. Or en fait la tâche x + 1 peut commencer à s'exécuter avant la fin effective de la tâche x tandis que commence la construction d'une tâche x + 2 et ainsi de suite. La figure suivante est donc beaucoup plus réaliste.

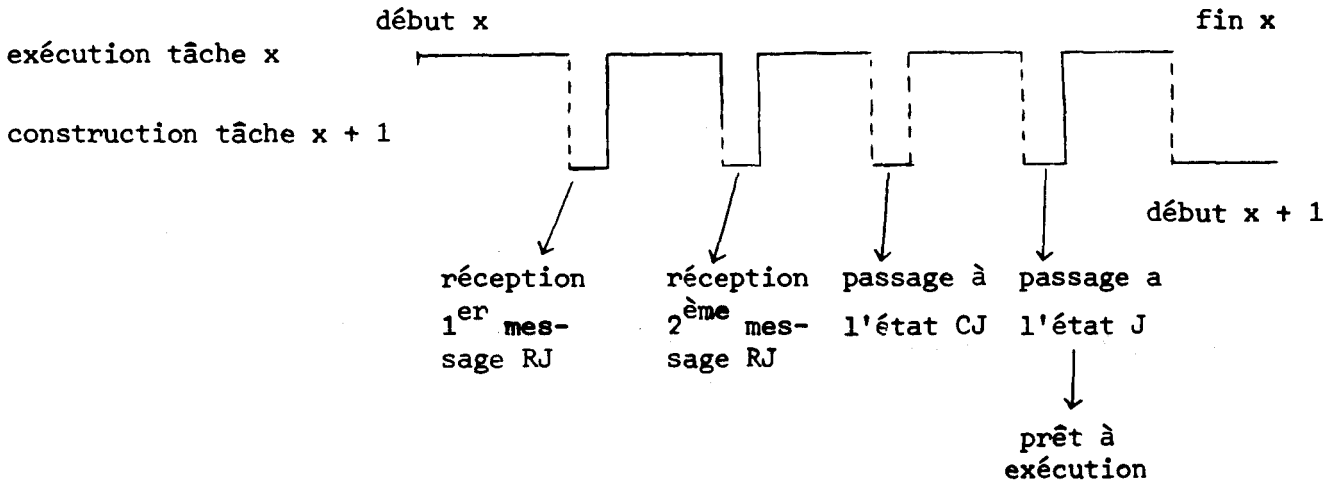


BUS LILLE

Cette superposition peut se réaliser physiquement de diverses façons.

1° Une procédure de construction peut interrompre toute cellule en exécution afin de "réserver" un graphe optimal d'exécution pour la tâche suivante.

Exemple : au niveau d'une cellule, cela peut se traduire par :



2° La notion d'interruption apparaît trop gênante, on va implanter simultanément les deux niveaux pour assurer la simultanéité effective des deux actions. On superpose un processeur d'exécution puissant (type VLSI) et un processeur de service, construit en technologie rapide (composants FAST, TTL, ECL, I_2 L et FPLA) qui assure le double rôle de constructeur de la machine graphe et de gestion des échanges intercellulaires, avec l'extérieur. C'est la solution retenue.

On associe au processeur un automate chargé du traitement des messages constructeurs "courants", c'est-à-dire de ceux qui du point de vue opérations ne demandent pas plus que des comparaisons, incrémentations, décalages. Les opérations plus complexes seraient exécutées par le processeur VLSI. L'automate de routage des messages est bâti autour d'un séquenceur (FPLAS), additionneur, comparateur, registres et petite mémoire rapide. Un des objectifs

visés est que le temps de construction d'un niveau du graphe soit de l'ordre de la µs dans les cas les plus défavorables alors qu'on situe le temps d'exécution d'un bloc dans l'ordre de 0,1 à 1 ms, temps sur lequel on aligne les accès mémoires dans le cas des "défauts de bloc". Ainsi, dans le cas d'une connaissance anticipée du graphe d'exécution (toujours s'il s'agit d'une machine éphémère), le temps global d'exécution ne sera qu'à peine supérieur au temps théorique d'exécution compte tenu du temps nécessaire à la construction des premiers niveaux. L'intégration dans un même boîtier des deux niveaux serait bien sûr un objectif à plus long terme

5.8 - CONSTRUCTION INDEPENDANTE DE L'EXECUTION

Cette indépendance n'a un sens que si cette construction se fait avant l'exécution (à l'intérieur d'une même tâche). "L'état" d'une cellule est caractérisé par un comportement et donc par un enchainement d'opérations de "service" déterminé. Les basculements d'état sont commandés par des messages adhoc, les transitions permises étaient explicitées par un diagramme (Fig. 5.1)

Parmi ces états on va distinguer des états transitoires et des états terminaux, c'est-à-dire des états où doit se retrouver une cellule une fois la phase de construction terminée.

5.8.1 - Etats transitoires

Eclatement (symbole E) : la cellule recherche d voisines successeurs, d étant le degré de l'éclatement.

Relais (symbole RLS) : cellule accessoire à la réalisation d'un éclatement.

Source (symbole S) : la cellule doit trouver une correspondante commune à un certain nombre (degré de la jonction) de ses semblables.

Recherche - Jonction (symbole RJ) : la cellule participe à la propagation des messages RJ et mémorise les chemins de façon partielle (un opérande partiel = un prédécesseur dans la table de liaison).

Candidate - Jonction (symbole CJ) : la cellule se reconnait comme la correspondante possible des diverses sources et pose sa candidature à la jonction (émission CAND).

Jonction (symbole J) : la cellule dans l'état CJ a reçu un message de validation VAL de l'arbitre. Elle le répercute vers les sources non arbitre et se place dans un état terminal d'exécution.

Normal (symbole N) : la cellule n'a qu'un prédécesseur et un successeur dans le graphe (précède un état d'exécution terminal).

5.8.2 - Etats terminaux

Libre (symbole L) : la cellule n'est chargée d'aucune tâche d'exécution ou de relais.

Machine graphe (symbole MG) : c'est un état qui correspond à l'exécution d'un bloc (unité de traitement). Il suit un état E, J ou N).

LIEN (symbole LN) : c'est un état qui ne participe pas à l'exécution proprement dite mais qui permet de lier entre elles des cellules (MG). Il suit un état RELAIS, Recherche jonction, candidate jonction.

Fig. : 5.1 - Diagramme des transitions de la phase construction

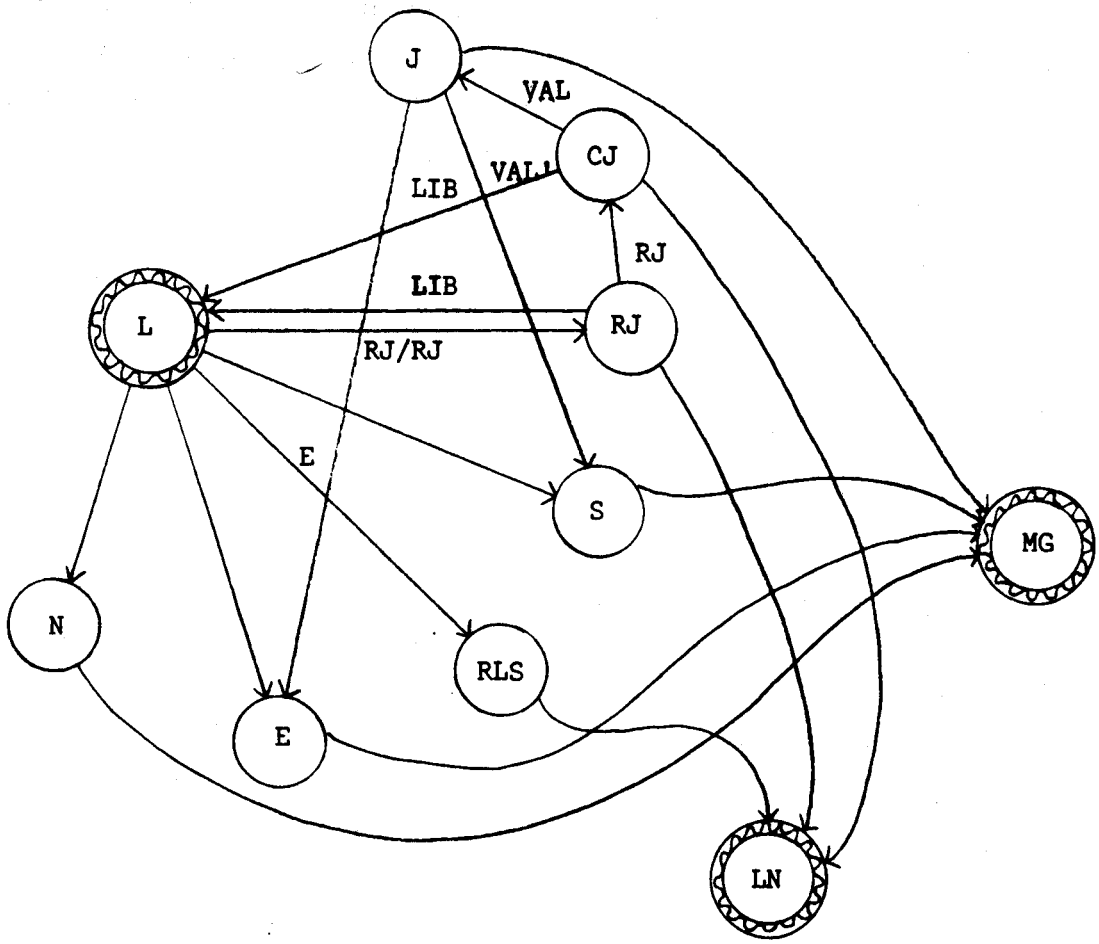
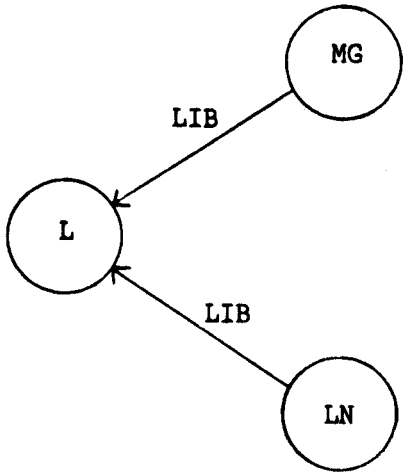


Fig. : 5.2 - Diagramme des transitions de la phase exécution



Il y n'y a qu'un état terminal : L

BUS LILLE

Les transitions entre états transitoires et états terminaux peuvent être considérés comme automatique lors de la prise en compte de la tâche correspondante au niveau d'exécution. La figure 5.1 laisse en blanc quelques transitions autres que celles relatives à l'éclatement ou à la jonction. C'est que nous n'avons pas encore explicité la façon dont on va passer d'un graphe théorique d'exécution à la machine graphe.

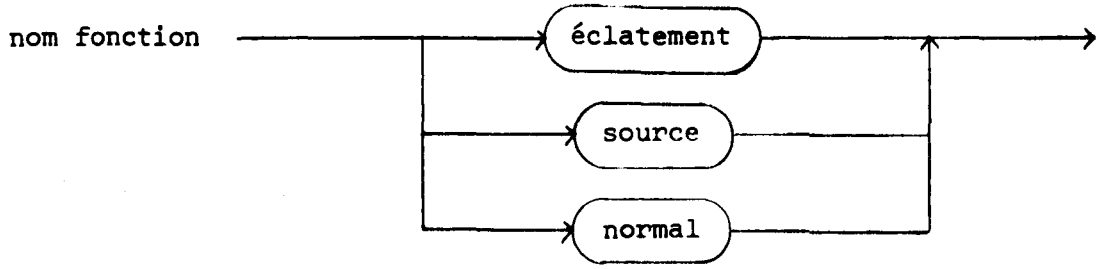
5.8.3 - Message constructeur

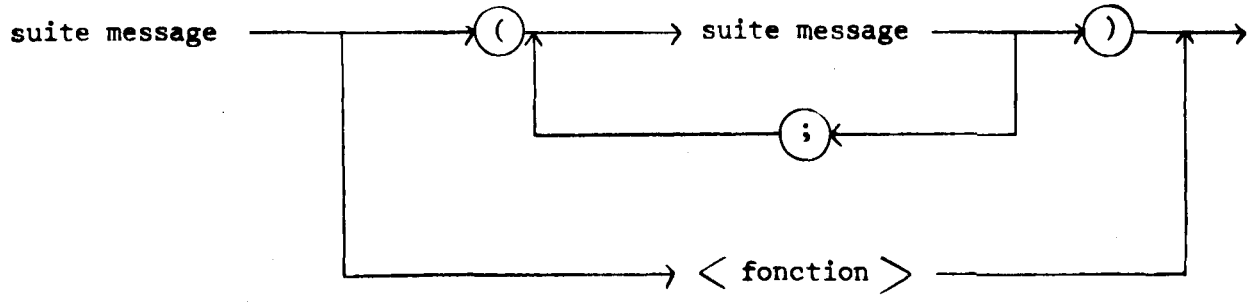
La construction d'une machine graphe va être réalisé grâce à un ou plusieurs messages constructeurs, émis par le système hôte, portant ce qu'on pourrait appeller le *code génétique* du graphe. Celui-ci comportera assez d'information pour permettre aux cellules de reconstituer leur graphe grâce aux diverses routines de service correspondant aux états précédemment passée en revue. Ce message va être bâti autour d'un alphabet limité à quelques symboles et trois séparateurs qu'on va symboliser dans ce qui **suit** par les *parenthèses* ( ), le *point virgule* ; et la *virgule* ,.

On peut condenser l'écriture du message sous forme syntaxique :

message initial ——> < fonction > ——> (,) ——> < suite message >

fonction ——> < nom-fonction > (,) < entier > ——>





Au niveau d'une cellule, l'information utile se compose de deux éléments :

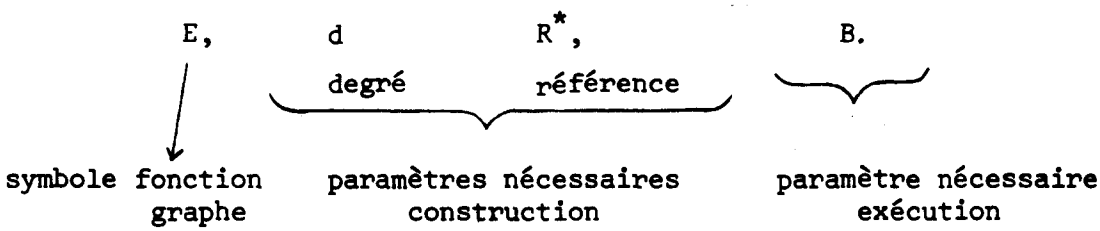
premièrement : La fonction graphe à exécuter avec les paramètres nécessaires.

deuxièmement : La référence du bloc à exécuter (instructions + constantes initiales).

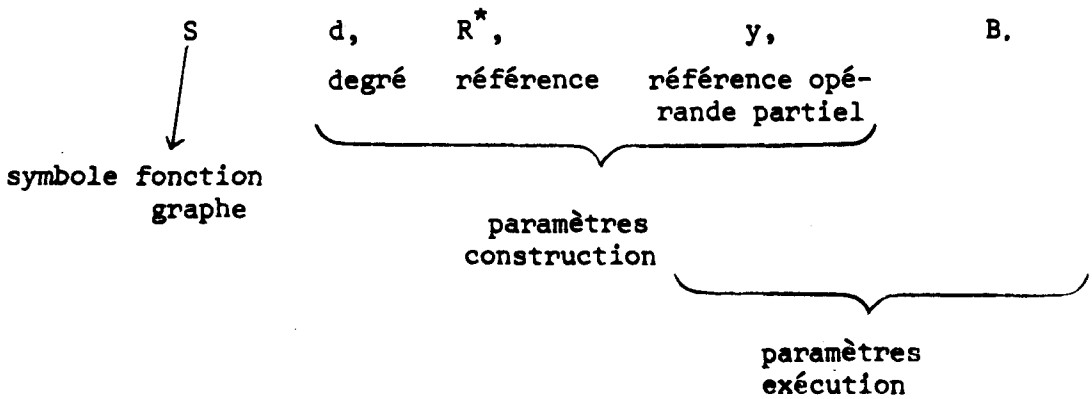
La référence du bloc peut être obtenue par la concaténation de la référence du noeud (R) et de l'occurence de tâche en cours ou par un nom spécifique (cas de blocs standards présents dans les cellules).

On écrira donc :

Eclatement



Jonction (initialisée par les cellules *sources*)

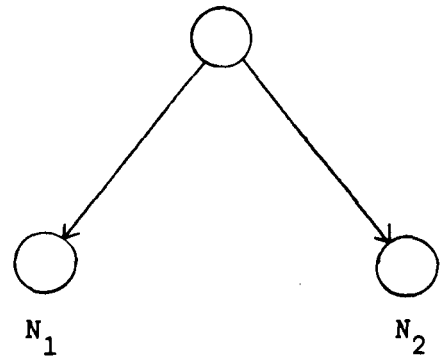


* Référence du noeud correspondant dans le graphe théorique d'exécution.

NORMAL : Correspond au cas d'un noeud ayant un seul successeur et un seul prédécesseur. On le considèrera fonctionnellement comme un éclatement de degré 1 ce qui uniformise l'écriture des messages.

- Les virgules séparent les informations nécessaires à reconstituer un noeud donné.

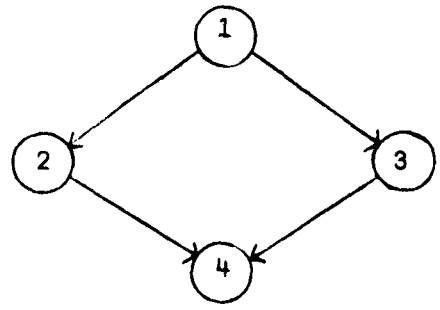
- Les points virgules séparent les informations destinées à des noeuds du même niveau (après un éclatement).



$M[N_1]$; $M[N_2]$

- Les parenthèses correspondent directement aux divers niveaux du graphe. Au fur et à mesure que le graphe se construit, le message initial est amputé des niveaux remplis.

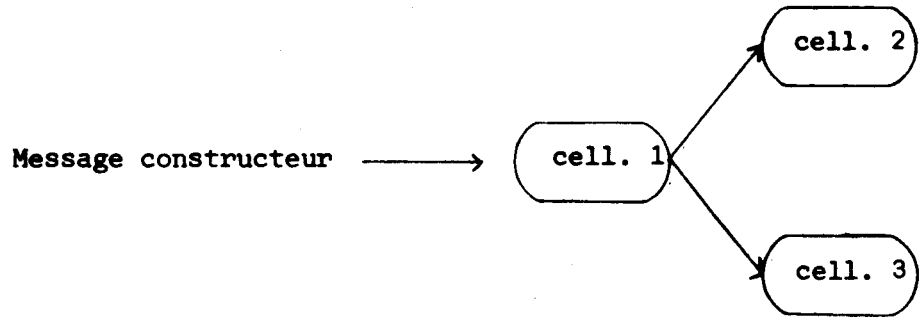
<u>Exemple</u> : On omet les références blocs.



Le message va s'écrire :

E, 2, 1 (S, 2, 4 1 (E, 1, 4) ; S, 2, 4, 2)

Une cellule considère d'abord l'entête qui lui donne les fonctions de construction et d'exécution.



La cellule a dépouillé l'entête E, 2, 1 et elle retransmet deux messages différents à deux successeurs 2 et 3 qui sont respectivement

M_2 = S, 2, 4, 1 (E, 1, 4)

M_3 = S, 2, 4, 2

La cellule redistribue les messages sous la forme nouvelle entête (nouvelle suite message), déduit du précédent par l'élimination du niveau supérieur de parenthèse.

Seule M_2 possède une "suite message" car dans le cas d'une jonction, on n'a besoin de donner la suite message qu'à une seule source pour toucher l'ensemble des successeurs de la jonction.

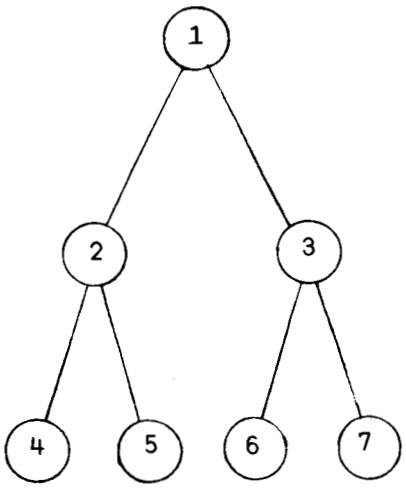
Cell. 2 et Cell. 3 vont donc chercher à établir une jonction.

Celle-ci une fois validée selon la procédure vue en 5.6 Cell. 2 transmet la suite du message à la cellule élue.

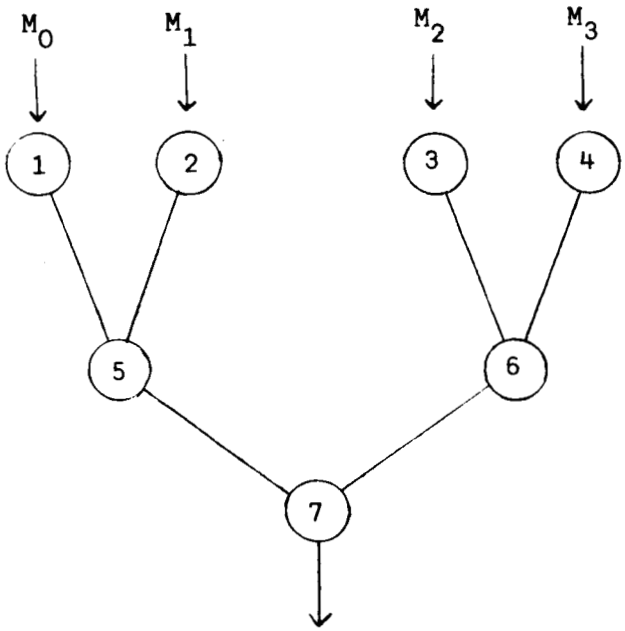
<u>Exemple</u> : Arborescences.

a/ <u>En éclatement</u>.

E, 2, 1 (E, 2, 2 (E, 1, 4 ; E, 1, 5) ; E, 2, 3 (E, 1, 6 ; E, 1, 7)).



b/ En jonction



M_0 = S, 2, 5, 1, 1 (S, 2, 7, 1, 5 (E, 1, 7))

M_1 = S, 2, 5, 2, 2

M_2 = S, 2, 6, 1, 3 (S, 2, 7, 2, 6)

M_3 = S, 2, 6, 2, 4

BUS LILLE

C'est le système hôte qui gère globalement la machine et les E/S qui emet les messages constructeurs initiaux.

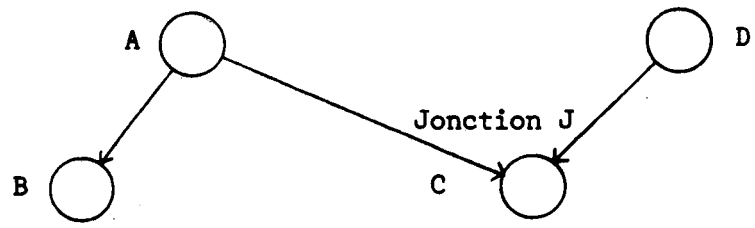
- Noeuds "Source - Eclatement".

Dans le cas où dans le graphe théorique d'exécution, il existe un noeud qui serait à la fois éclatement et source, deux options sont possibles.

a/ On introduit une nouvelle fonction graphe "éclatement/source".

b/ On garde les précédentes mais on introduit des fonctions fictives ou fantômes.

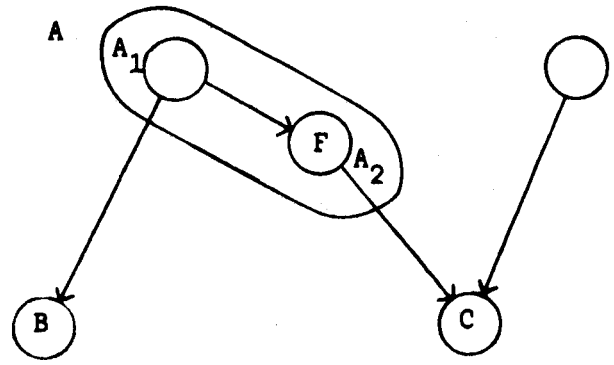
<u>Par exemple</u> :



A est à la fois un éclatement et une source pour la jonction J.

La solution (a) se traduit au niveau d'une cellule par l'exécution d'un éclatement en deux temps : on élit le correspondant B puis on lance une recherche jonction dont le résultat donne C.

La solution (b) nous fait décomposer A en deux noeuds A_1 et A_2.



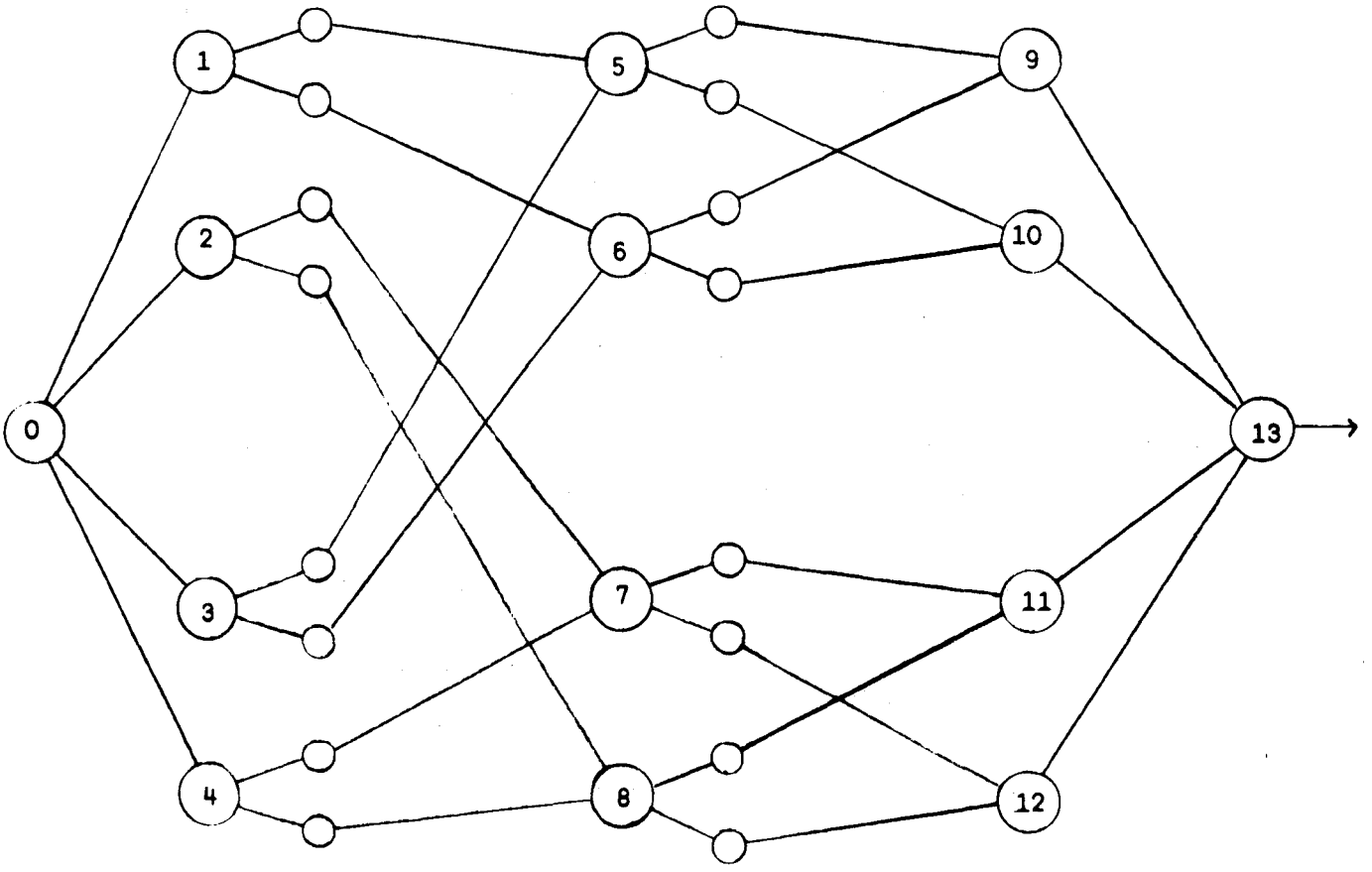
A_2 est censé réaliser une fonction fantôme, reconnue comme telle par l'automate de niveau inférieur : il n'y a pas de réservation du niveau supérieur.

La solution (a) permet une plus grande compacité du graphe dans l'espace cellulaire alors que (b) accélère la procédure de construction (cas des cellules éclatement - multisources).

Nous envisageons d'abord d'implanter (b).

Cette notion de tâche fantôme et qui ne débouche donc sur aucune réservation du niveau supérieur d'une cellule peut être utile pour restituer des boucles dans un contexte isotrope ou construire des machines éphémères où l'on rencontre fréquemment des noeuds "éclatement - source".

<u>Exemple</u> : Machine FFT 8 facteurs.

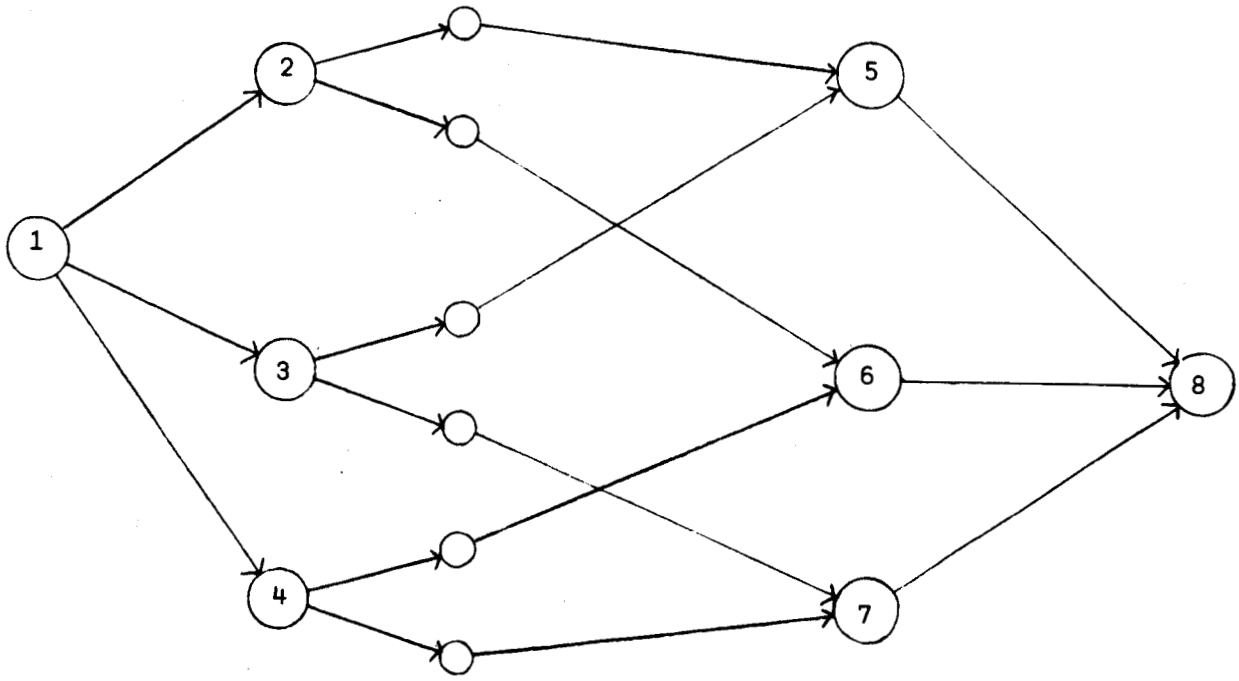


Le message constructeur, sans tenir compte de la déclaration des fonctions d'exécution s'écrira :

M = E, 4, 0 (E, 2, 1 (S, 2, 5, 1 (E, 2, 5 (S, 2, 9, 1 (S, 4, 9, 1) ; S, 2, 10, 2)) ; S, 2, 6, 2) ; E, 2, 2 (S, 2, 7, 1 (E, 2, 7 (S, 2, 11, 1 (S, 4, 13, 0)) ; S, 2, 8, 2) ; E, 2, 3 (S, 2, 6, 1 (E, 2, 6, 1 (E, 2, 6 (S, 2, 10, 1 (S, 4, 13, 3)) ; S, 2, 5, 2) ; E, 2, 4 (S, 2, 8, 1 (E, 2, 8(S, 2, 11, 1 (S, 4, 13, 4)) ; S, 2, 11, 2).

On a numéroté que les seuls noeuds correspondants au graphe théorique d'exécution.

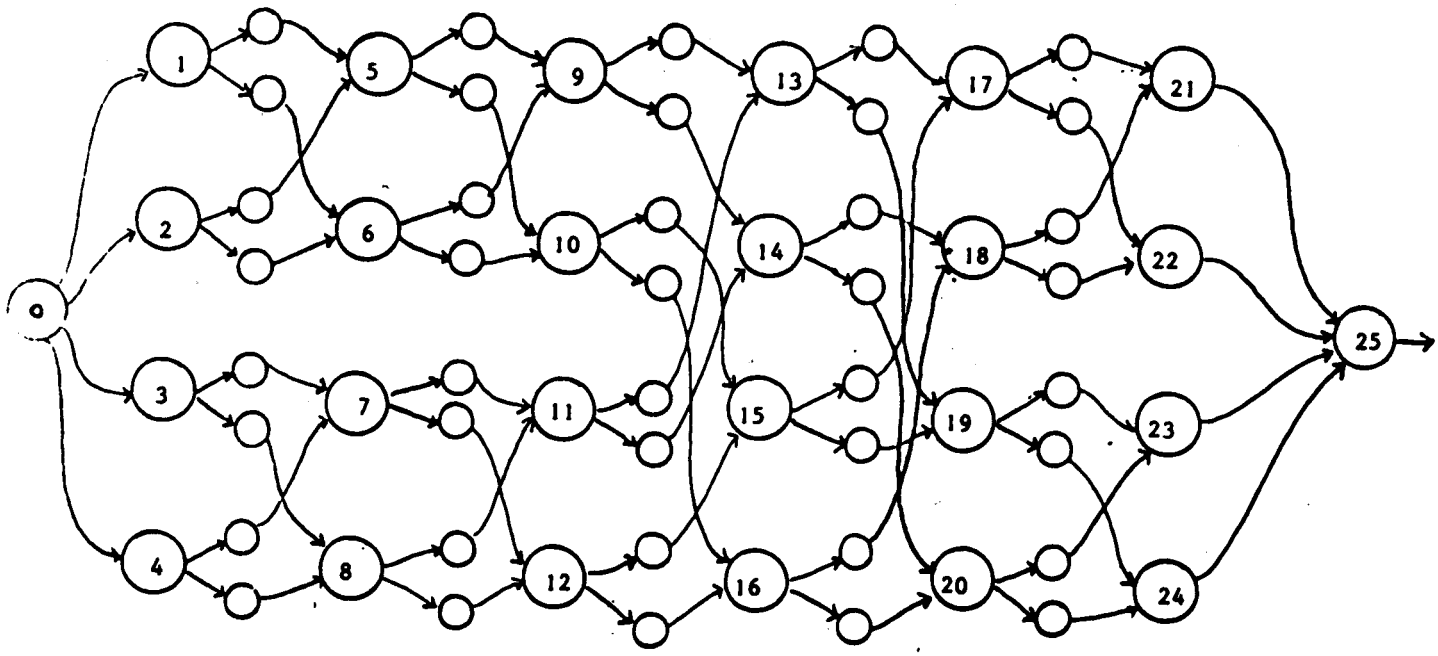
Exemple 2 : Construction d'une structure cube.



Le message constructeur initial sera :

Message initial = E, 3, 1 (E, 2, 2 (S, 2, 5, 1 (S, 3, 8, 1 (E, 1, 8)) ; S, 2, 6, 2) ; E, 2, 3 (S, 2, 5, 2 ; S, 2, 7, 1 (S, 3, 8, 2)) ; E, 2, 4 (S, 2, 6, 1 (S, 3, 8, 3) ; S, 2, 7, 2)).

Exemple 3 : Trieur de Batcher.(7)



Le message constructeur initial sera M_0 = E, 4, 0 (M_1, M_2, M_3, M_4)
M_1 = E, 2, 1 (S, 2, 1, S (E, 2, 5 (S, 2, 1, 9 (E, 2, 9 (S, 2, 1, 13 (E, 2, 13 (S, 2, 1, 17 (E, 2, 17 (S, 2, 1, 21 (S, 1, 8, 25) ; S, 2, 2, 22)) ; S, 2, 2, 19)) ; S, 2, 2, 14)) ; S, 2, 2, 10) ; S, 2, 2, 6).

5.8.4 - Contrôle de la construction

Celui-ci ne peut être qu'un contrôle passif fait à postériori. Les cellules ayant acquis une référence précise (numéro de bloc) correspondant à un noeud dans le graphe d'exécution renvoie un message portant cette référence vers le système-hôte. Celui-ci peut donc suivre la construction et détecter certaines anomalies en comparant cette dernière au modèle qu'il possède.

BUS LILLE

5.9 - CONSTRUCTION DEPENDANTE DE L'EXECUTION

Cette dépendance peut s'exprimer de deux manières :

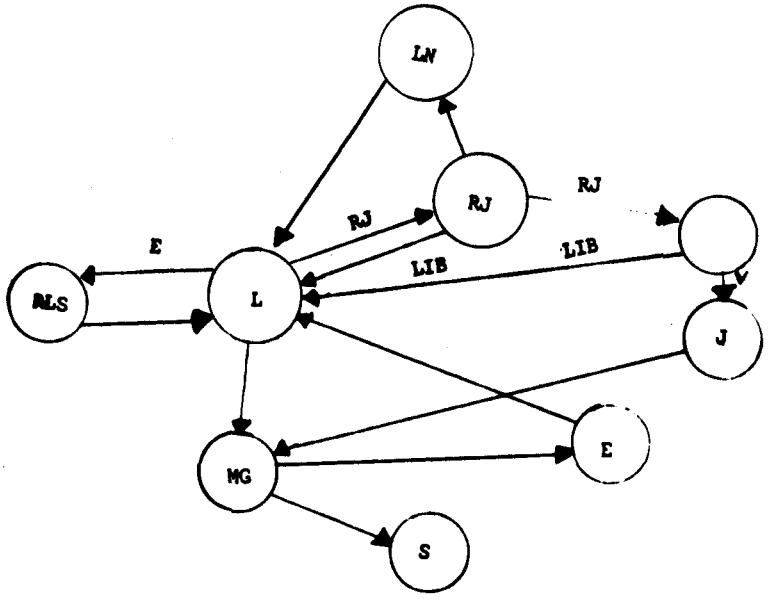
Le système hôte exécute une tâche en utilisant l'espace cellulaire comme un périphérique en lui attribuant des traitements partiels dont la forme peut varier selon des résultats précédemment acquis.

Les cellules exécutent un bloc avant de passer à la recherche du ou des successeurs, Elle n'est pas très séduisante car il n'y a pas comme précédemment recouvrement entre exécution et construction alors que celui-ci existait à deux niveaux :

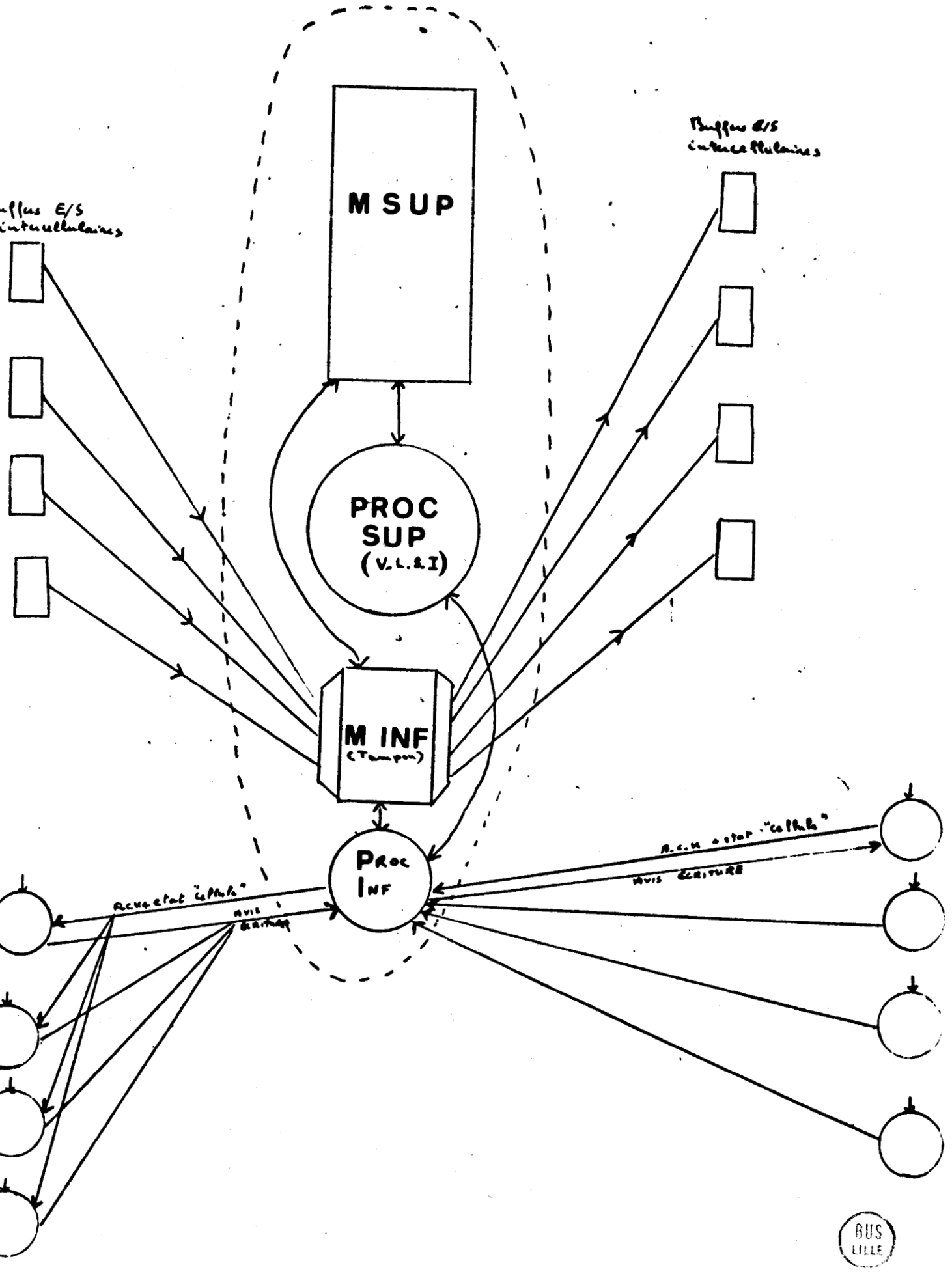
- entre tâches différentes,
- à l'intérieur d'une même tâche, on peut commencer l'exécution du niveau inférieur alors que les autres se construisent.

Si t est le temps moyen d'exécution d'un bloc et θ le temps moyen de construction du graphe pour passer d'un niveau au suivant, le temps total de passage sera de $N(t + \theta)$ dans ce cas contre $nt + \theta$ ou nt dans le cas précédent. Le temps θ dépendant fortement de la nature du graphe, cette solution peut rester intéressante dans les cas où l'écart moyen du temps d'exécution local est faible et les fonctions graphes locales de degré 2 par exemple, (type Fork-join).

Fig.: 5.3-Diagramme des transitions d'états permises



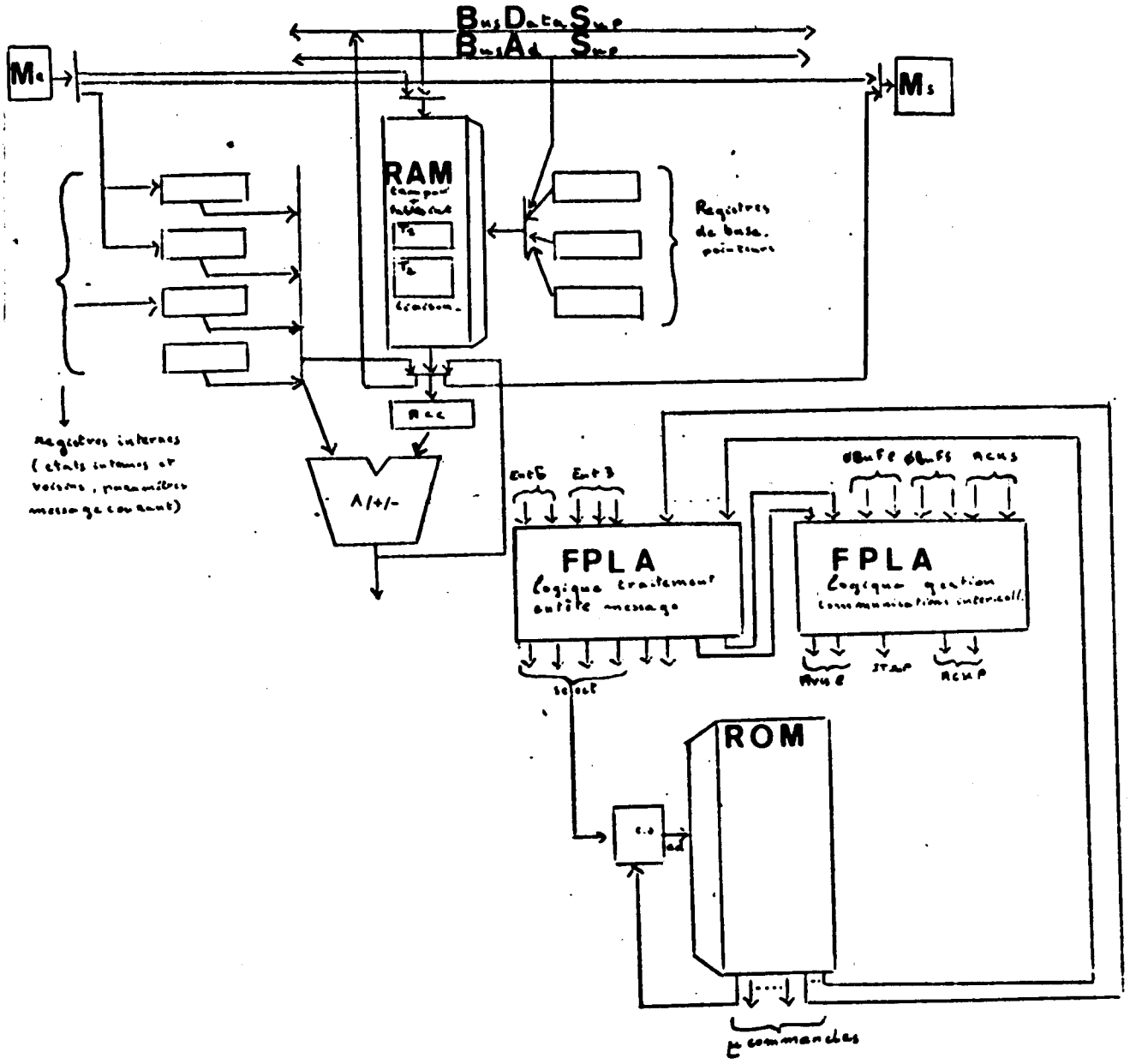
On a montré ici le cas où il n'y a pas d'états terminaux : libérations et constructions s'enchaînent au fur et à mesure des exécutions. On pourrait se ramener au cas précédent si la première exécution servait à la construction du graphe, récupéré ensuite pour d'autres occurences de tâche par le système, la libération s'effectuant sur son ordre. Les procédures de construction restent identiques.



BUS LILLE

Schéma symbolique de la cellule de base.

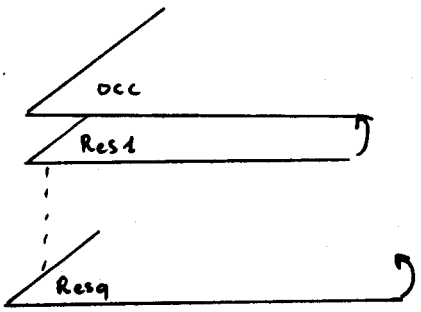
5.10 - TRAITEMENT DES MESSAGES PAR L'AUTOMATE DE NIVEAU INFERIEUR



On montre ici une réalisation possible de l'automate de niveau inférieur . L'unité de contrôle est réalisé à partir de FPLAs et ROM .Les tables de liaison sont placées dans une zone de la mémoire RAM statique qui sert en même temps de tampon avec le niveau supérieur .Il faut noter que le gros du travail réalisé au niveau inférieur est la gestion des tables de liaison ,comportant notamment des recherches pseudo-associatives .Une solution empoyant des mémoires associatives "vraies" pour cette recherche ainsi que la scrutation des voisins libres (cf chap 4) diminuerait d'un ordre de grandeur au moins les temps de service tout en permettant l'emploi des seuls Fplas pour réaliser l'unité de commande. Le prix resterait bien sur l'obstacle majeur d'une telle solution . Elle est néammoins la plus intéressante à long terme si on veut intégrer le niveau inférieur . Au stade actuel de l'étude ,l'émulation de cet automate par un processeur rapide du type 8X3oo (voire AMD 2900) serait par contre envisageable dans l'avenir immédiat .
Le paragraphe qui suit décrit les taches du niveau inférieur sur un plan strictemen fonctionnel ,sans préjuger de la réalisation .

5.10.1 - Degré de liberté - Ordre de scrutation des voisins libres

La "liberté" d'une cellule ne sera plus représentée comme dans le chapitre précédent par un simple booléen. On a permis en effet la cohabitation de deux niveaux : un niveau d'exécution dit supérieur et un niveau de construction dit inférieur. Ce découpage peut correspondre à l'exécution d'une occurrence de tâche x tandis que x + 1 se construit, se réservant les cellules nécessaires à son exécution future. Dynamiquement ou globalement, alors que x s'éxécute, x + 1 peut fort bien commencer sur exécution et ainsi de suite. Au niveau d'une cellule, on conçoit fort bien que lorsqu'on exécute Tx, on construit Tx + 1 et si on a le temps Tx + 2..Tx + q : il y a donc alors q tâches en attente d'exécution. Dans ce cas, le degré de liberté attaché à une cellule serait inversement proportionnel à q, une cellule étant considérée d'autant moins libre que son plan de charge futur est engagé.



C'est à rapprocher de la stratégie de diffusion dans le scheduling spatial du MU.NET (25).

Pour l'instant nous nous contentons de deux degré de liberté :

1er degré = cellule ni en exécution ni réservée.
2ème degré = cellule en exécution mais non réservée.

Une cellule "occupée" étant à priori en exécution et réservée. Le choix d'une politique de réservation de chemins nous amène à introduire un troisième degré de liberté. En effet pour éviter tout risque de blocage dans la construction d'une machine graphe, il est nécessaire de laisser passante une cellule occupée, mais en l'ocurrence ce ne sera possible tant que les tables de liaison de la cellule ne sont pas saturées.

On obtient un mot d'état sur deux bits :

degré de liberté	1°	00	=	niveaux SuP et INF libres
	2°	01	=	niveau SuP occupé, INF libre.
	3°	10	=	niveau SuP occupé, INF réservé, cellule passante
		11	=	niveau SuP occupé, INF réservé, cellule non passante

La scrutation des voisins libres (cf-chapitre IV) se fera d'abord au 1° degré (0,0) puis au 2° degré (0,1) puis au 3° degré (1,0).

5.10.2 - Les messages

On distingue quatre types de message :

1. Messages terminaux ou de service (destinés à la sortie). Entrent notamment dans cette catégorie les messages de compte rendu des diagnostiques effectuées par les cellules "inactives" lors des procédures de test qui s'effectuent chaque fois qu'une cellule est momentanément inactive état (0,0).

2. Messages "système". Pour l'instant on n'a envisagé que le message SLIB (ref tâche) qui provoque la RAZ immédiate des processeurs de niveau supérieur exécutant la tâche référencée.

3. Messages constructeurs (E,S) ou constructeurs intermédiaires (RJ, Relais, VAL,CAND,LIB) concernent le niveau inférieur.

4. Messages LIEN (véhiculent les données, les résultats). Ce sont les liaisons utiles au traitement. Ils sont cependant traités au niveau inférieur pendant leur transit .

On considère ici les messages 3 et 4.

5.10.3 - Message type 3

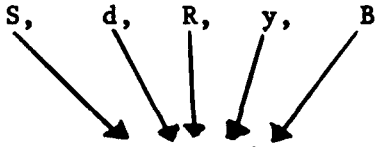
On va distinguer :

- la manipulation du message
- la procédure correspondant à l'état de la cellule consécutif à la lecture du message.

Dans les cas particuliers où une cellule dans un état (1,0) reçoit un message source ou éclatement avec une référence bloc qui ne soit pas le fantôme, la cellule repasse le message tel quel à un successeur libre non sans avoir marqué la référence de la source ou l'éclatement dans la table de liaison.

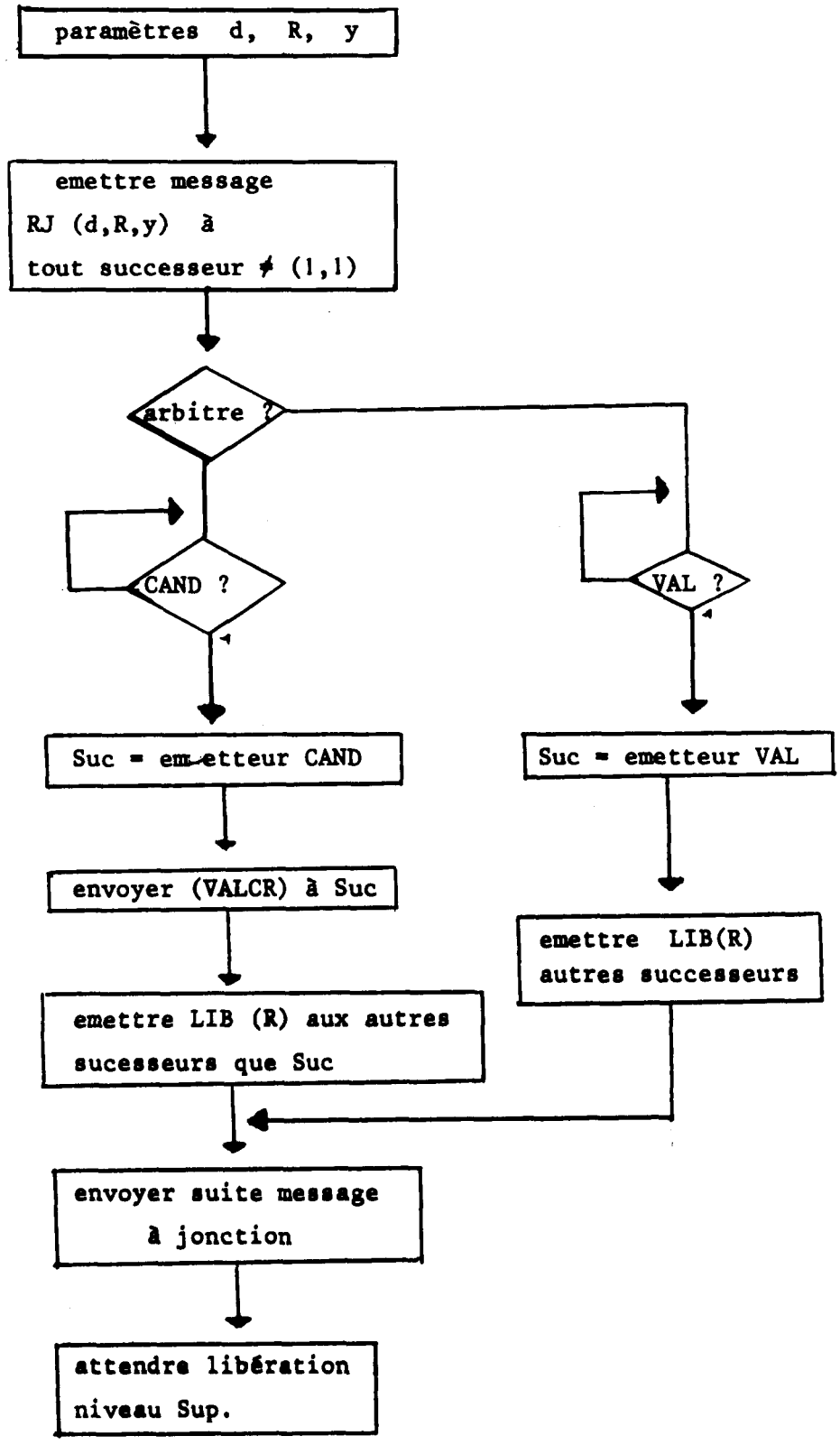
5.10.4 - Entête message = source

Symbole S, d, R, y, B (suite message)



formats fixes.

La manipulation du message proprement dite est simple puisqu'une fois la jonction établie, la "suite message" est transmise en bloc par le chemin sélectionné. La procèdure correspondant à l'état "source" est décrite sous la forme d'un organigramme pour alléger la représentation tout en la gardant facilement compréhensible. Cela ne préjuge pas de la réalisation effective du séquenceur.

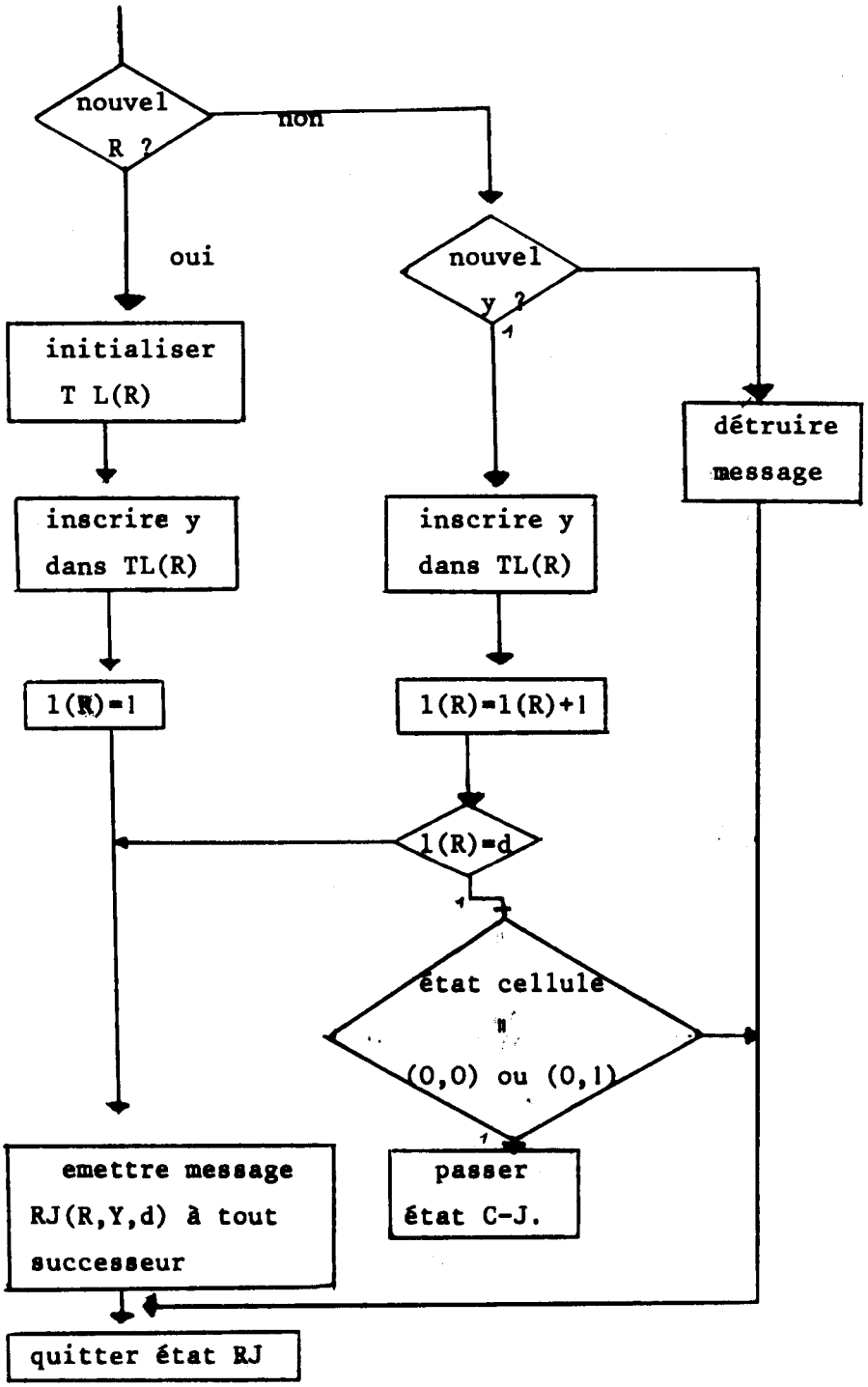


BUS LILLE

La désignation de l'arbître peut être le résultat d'un calcul ou d'un choix arbitraire (par exemple la source possédant la référence opérande partiel égale à 1 est l'arbitre).

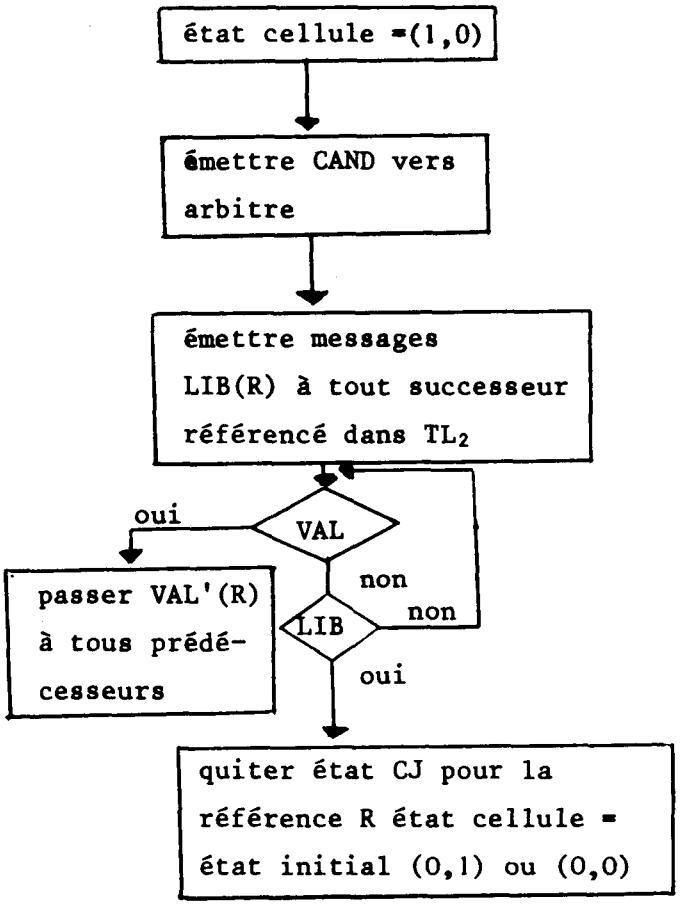
5.10.5 Message R J (message constructeur intermédiaire)

La manipulation du message est simple = on recopie les paramètres avant d'envoyer tel quel le message à tout successeur libre (état (0,0), (1,0), (0,1)) ou on le détruit.



BUS LILLE

5.10.6 état CJ



BUS LILLE

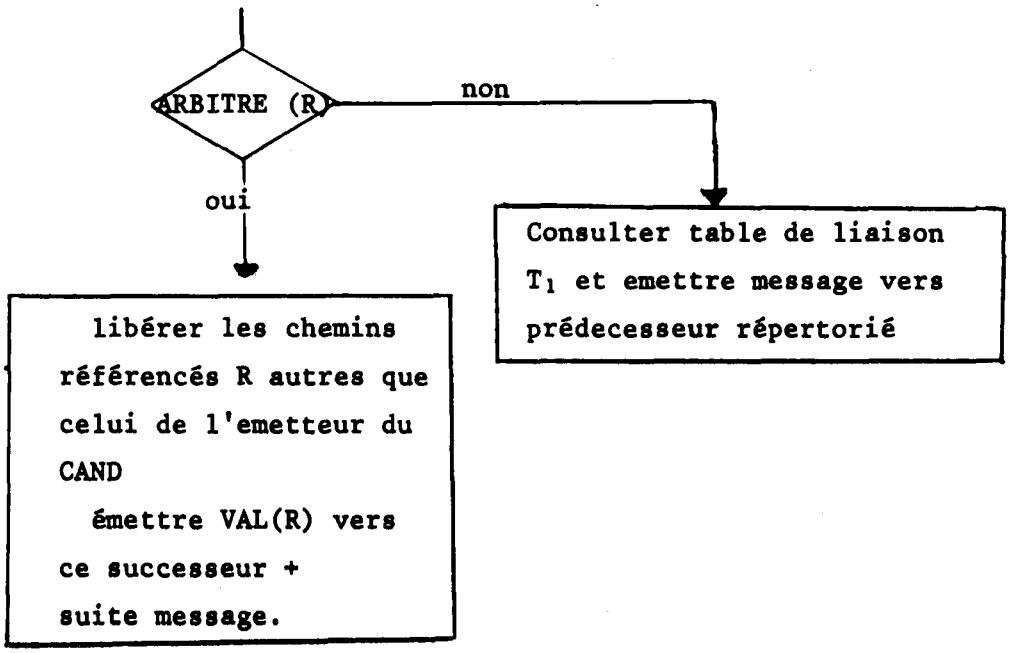
5.10.7 Message CAND

Rappellons qu'un message CAND portant référence d'une jonction (noeud correspondant du graphe théorique d'exécution) est émis par une cellule Candidate Jonction vers l'arbitre (cf 5.62).

Le message se présente sous la forme =

$$\underbrace{C\ A\ N\ D,}_{\text{entête}}\ \underbrace{R}_{\text{référence}}$$

La cellule qui reçoit un CAND compare la référence incluse avec sa référence propre si elle est arbître sinon elle consulte la table de liaison T_1 pour retrouver le chemin menant à ce dernier.



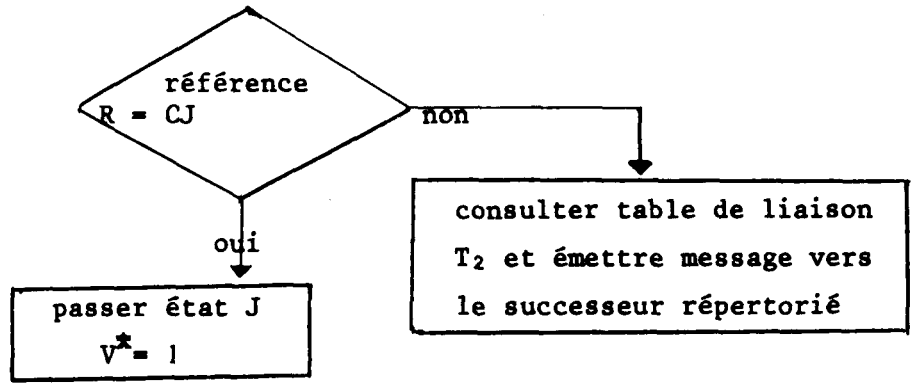
5.10.8 Message VAL ( )

De même, rappellons qu'un message VAL portant référence de la jonction (noeud équivalent du graphe théorique d'exécution) est émis par l'arbître vers la candidate jonction qu'il sélectionne (cf 5.62).

Le message VAL se présente donc :

$\underbrace{V\ A\ L}_{\text{entête}}$, $\underbrace{R}_{\text{référence jonction courante.}}$

La manipulation du message est encore une fois très simple. La cellule qui reçoit un VAL vérifie si la référence est équivalente à celle qu'il a mémorisé dans le cas où cette cellule s'est déclarée CJ. Sinon elle consulte sa table de liaison pour représenter le VAL vers le bon successeur.



<u>état J</u> : La cellule repercute le VAL vers les sources non arbîtres.

5.10.9 - Message LIB (R)

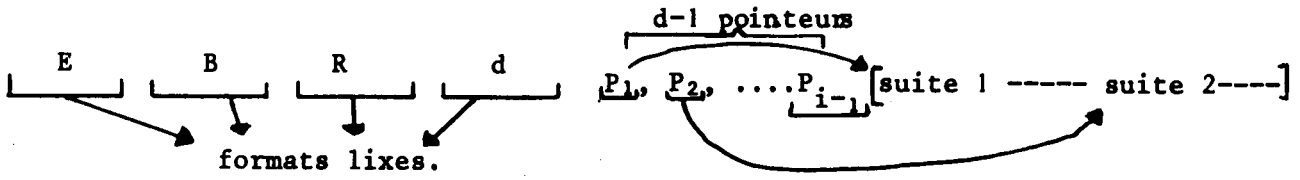
Le message correspond à la fin d'une phase de construction et n'est pas assimilable à un SLIB lancé par le système. Il porte la référence de la jonction sur laquelle la cellule avait été précédemment préssentie. L'opération consiste à libérer les emplacements référencés par R dans la table de liaison TL_2 (cf 5.62). Une cellule dans l'état (1,1) repasse à l'état (1,0) ou (0,1) ou (0,0) selon l'occupation des deux niveaux de la cellule.

* cf le RdP décrêvant globalement l'activité du niveau inférieur.

5.10.10.- Message éclatement

Symbole E, d, R, B (———— ;———— ;————)

on l'a codé sous la forme :



Une fois qu'on a reconnu d, les différents "suite messages" sont rangés dans la mémoire auxiliaire du niveau inférieurs.

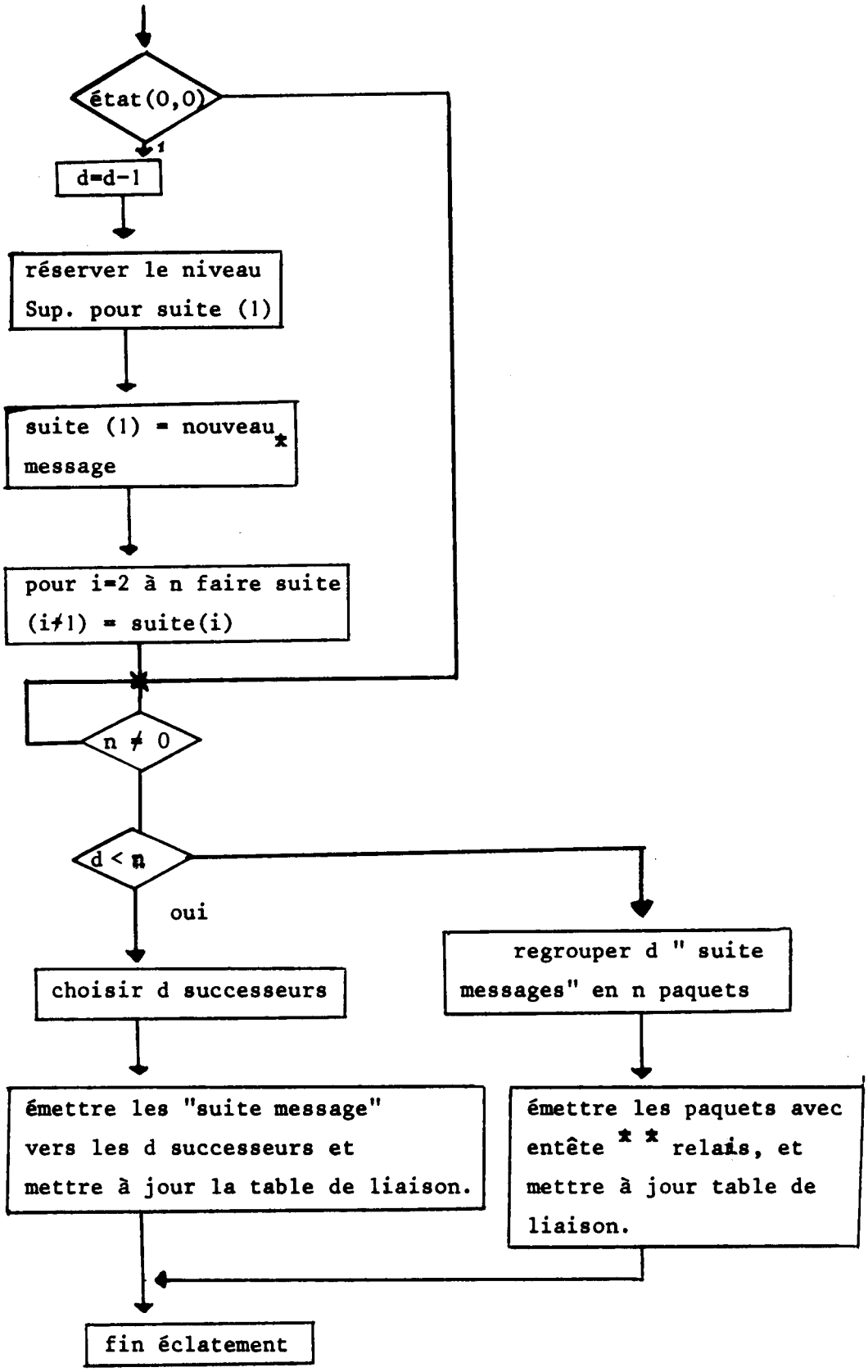
L'état transitoire "éclatement" va se concrétiser par les opérations suivantes :

paramètres d, R(suite$_1$), R(suite$_2$).....R(suite d)

d = degré de l'éclatement

n = nombre de voisins libre

R (suite i) = référence du message suite à emettre = nécessaire pour la mise à jour de TL.



* le nouveau message sera traité dès la fin de l'éclatement.

** Il faut rajouter une entête E, nouveau degré, fait pour émettre un message, conforme à la syntaxe défini.

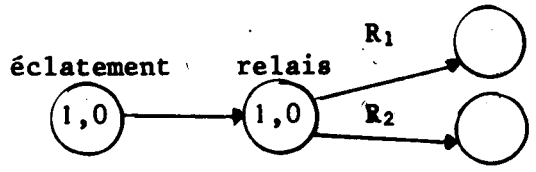
On a ramené le cas du "RELAIS" à celui de l'Eclatement en ajoutant que si la référence B est le fantôme, la cellule dans un état (0,0) peut se réserver pour exécuter un des "suite messages".

Si d < n, membre de voisin dans un état (0,0) ou (0,1) ou (1,0), chaque message suite est envoyé sur un successeur différent.

Dans le cas contraire, (n-1) successeurs recevront des groupements de Entier ($\frac{d}{n}$) messages "suite" et le dernier E ($\frac{d}{n}$) + reste ($\frac{d}{n}$).

De la même façon, un message N va être codé comme un message éclatement dont le degré d = 1. Ceci fait que dans tous les cas on range dans la table de liaison le code successeur aux adresses des références R de l'entête des messages suites.

<u>exemple</u> :



$$\underbrace{\text{Eclatement, Fant,}}_{\text{relais}} \text{ —— } \underbrace{\left(E,\ R_1,\ (\text{———})\ ;\ E,\ R_2(\text{———})\right)}_{\text{TL}}$$

va faire que dans la table de liaison TL le code du successeur haut va être rangé à R_1 et le code du successeur bas à R_2. Par la suite, tout message LIEN référençant R_1 ou R_2 sera aiguillé dans la bonne direction grâce à TL_1.

5.10.11 - Messages de type 4 ou LIEN

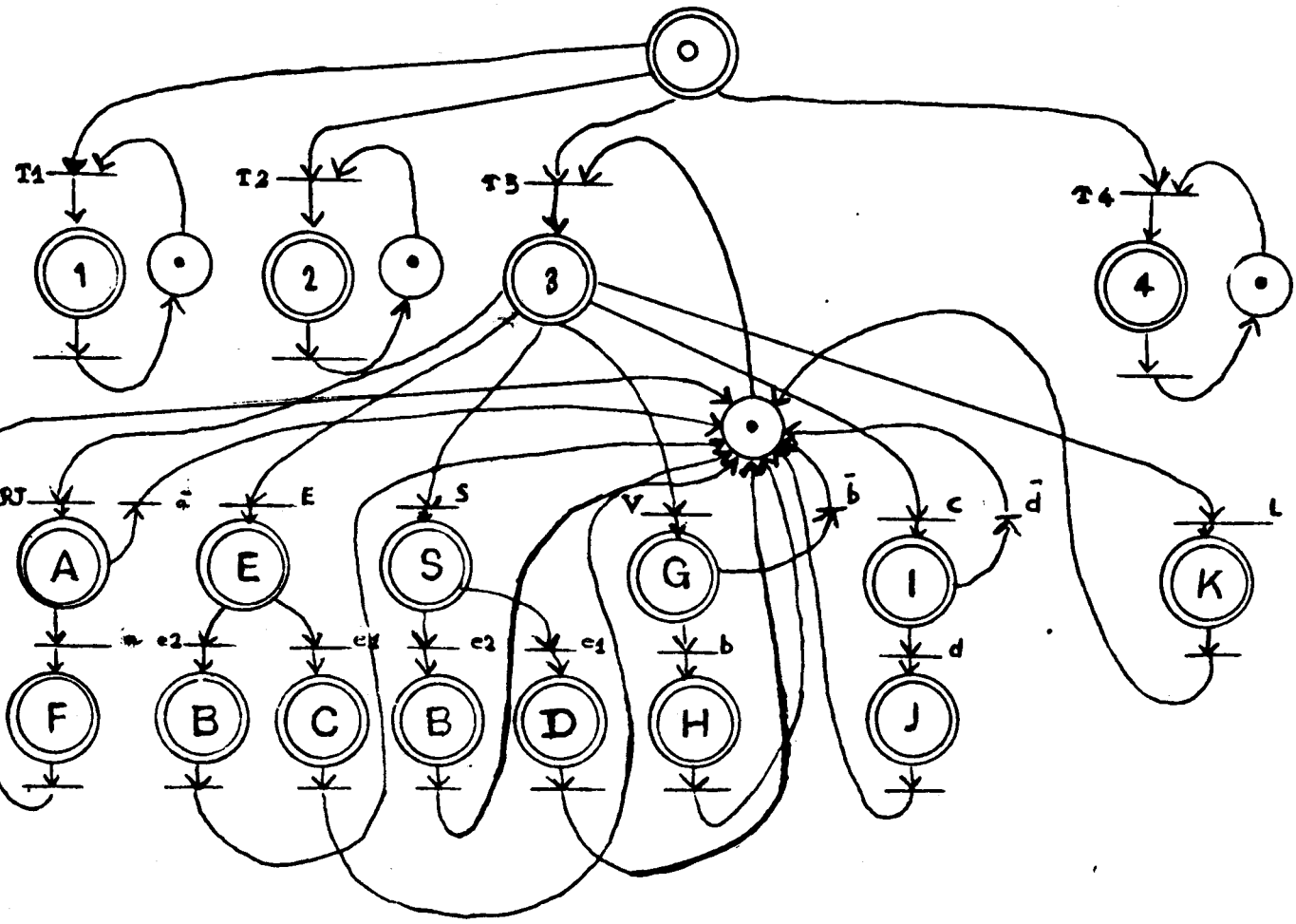
Ils correspondent dans le diagramme à un état terminal LIEN

Un message LIEN se décompose en trois parties.

entête	destinataire	données
LIEN	référence R	————————

L'entête est reconnu par l'automate de niveau inférieur qui consulte ensuite la destination R dont on a vu qu'on pouvait l'identifier au numéro d'un noeud dans le graphe théorique d'exécution précompilé. Si R est égal à la référence du niveau Supérieur, la partie data du message est placé dans la mémoire tampon et le niveau supérieur est averti. Dans le cas contraire, la consultation de la table de liaison fournit le successeur à qui on passe le message dans son intégralité.

Le réseau de Pétri interpétré suivant décrit globalement, le comportement du niveau inférieur quant à la manipulation des messages. Précisons que l'interprétation du réseau de Pétri se traduit par accoler un prédicat à certaines transitions.



Places cerclées en double : places d'action.

Places cerclées simples = synchronisations.

T_1, T_2, T_3, T_4 = égal à 1 selon le type du message 1, 2, 3, 4

RJ, E, S, V, C, L = égal à 1 selon le message de type 3 reçu = recherche jonction, éclatement, Source, VAL, CAND, LIB .

a = 1 quand la cellule est candidate jonction pour la référence donnée.

b = 1 si le message VAL reçu est destiné à la cellule et non en simple transit.

d = 1 quand la cellule se reconnait être l'arbître pour la référence incluse dans le message CAND reçu.

e_1= 1 quand la cellule est dans l'état (0,0) ou (0,1)

e_2= 2 quand la cellule est dans l'état (1,0)

Actions représentées

0 = dépouillement 1° entête message

1 = cf 5.10.2 traitement message type 1

2 = cf 5.10.2 traitement message type 2

3 = cf 5.10.3 traitement message type 3

4 = cf 5.10.11 traitement message type 4

A = cf 5.10.5 traitement message RJ

B = cf 5.10.3 cas ou état cellule = (1,0)

C = cf 5.10.7 traitement message éclatement 2° phase

D = cf 5.10.4 traitement message source 2° phase

E = cf 5.10.10 traitement message éclatement 1° phase

F = cf 5.10.6 traitement correspondant à un état CJ

G = cf 5.10.8 traitement réception message VAL (R)

H = cf 5.10.8 traitement message VAL=passage état J

I = cf 5.10.7 traitement réception message CAND(R)

J = cf 5.10.7 suite traitement CAND si la cellule est arbître

K = cf 5.10.9 traitement libération = mise à jour tables de liaison

S = cf 5.10.4 traitement réception message source

5.11.- PASSAGE DU CODE AUX CELLULLES.

On n'a pas fait jusqu'à présent d'hypothèse sur la façon dont au niveau d'une cellule on acquérait le code. Si on se place hors du cas où une cellule posséde à l'avance toutes les références possibles, diverses méthodes sont envisageables :

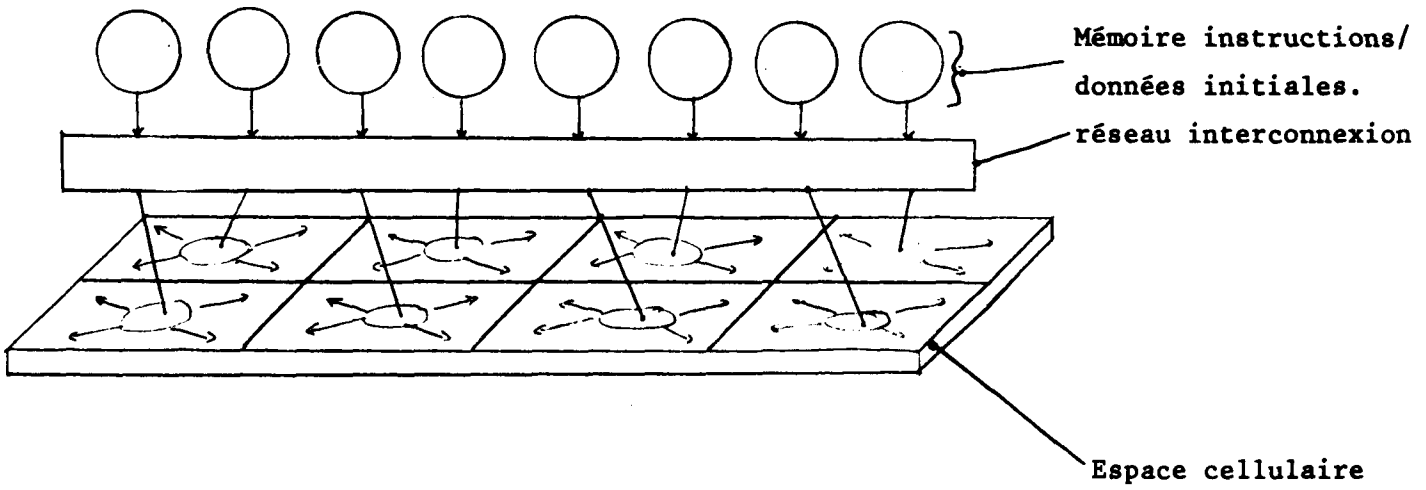
Le code est passé avec les messages constructeurs
avantage : on réutilise les chemins de données et une cellule élue devient immédiatement opérationnelle.
inconvénient : le message est d'une taille respectable.
Les temps de transmission et les tailles des tampons entre niveaux inférieurs vont devenir très grands.

déjà cité dans le chapître précédent (1.72) : toute trajectoire de message coupe au moins une zone dont les cellules possédent les codes référencées.
inconvénient : longueur des chemins, pas de souplesse car c'est une variante au cas où toutes les cellules possédent toutes les références.

Les cellules gérent un ensemble de blocs selon des stratégies classiques de pagination : (RU, MFU etc) avec deux alternatives :

a) les codes sont passées par les chemins de données en cas de défaut de bloc (procédure : détection d'un défaut de bloc = émission d'un message système référent le bloc vers le gérant mémoire + adresse physique de la cellule ou adresse logique (noeud du graphe théorique dans le cas de machine éphémère maintenue).

b) les codes sont passés par des chemins spécifiques. Par exemple l'espace cellulaire est divisé en un certain nombre de zones où existe un point d'Entrée/sortie spécifique par qui vont transiter les demandes de bloc et les codes



Le temps d'acquisition d'un bloc est borné par :

- temps d'accés mémoire secondaire
- temps passage dans réseau d'interconnexion
- temps de transit dans un chemin de cellules égale au diamètre max des zones découpées dans l'espace cellulaire autour des points E/S.

On parait ainsi violer l'integrité de notre espace cellulaire si on tient compte de la définition donnée au chapître 1. Il faut néanmoins noter cette solution comme une alternative possible.

5.12. Un exemple

On a utilisé le graphe de la figure 5 VI pour illustrer en quelques petites expériences le comportement d'une telle architecture, limitée seulement à une quarantaine de cellules.

5.12.1. Fonctionnement statique

On construit deux machines éphéméres "maintenues" déduites du graphe (5-VI) dans diverses topologies (cf figure 5-VII VIII IX X XI). On a répertorié dans le tableau I-5 les longueurs moyennes (l) des chemins, le chemin entre deux cellules représentant deux noeuds successifs dans le graphe d'exécution se définissant comme le nombre d'intervalles les séparant l'un de l'autre, c'est-à-dire le nombre de cellules intermédiaires plus un. Comme les deux graphes ont été construits l'un après l'autre, les valeurs l_1 et l_2 représentent les longueurs moyennes des chemins pour chaque construction. NTL représente la valeur moyenne du nombre de chemins passant par des niveaux inférieurs des cellules (ie taux moyen occupation des tables de liaison) et TOL le taux de cellules étant utilisé pour des tâches de liaison (assurant au moins une liaison, une émission ou une réception).

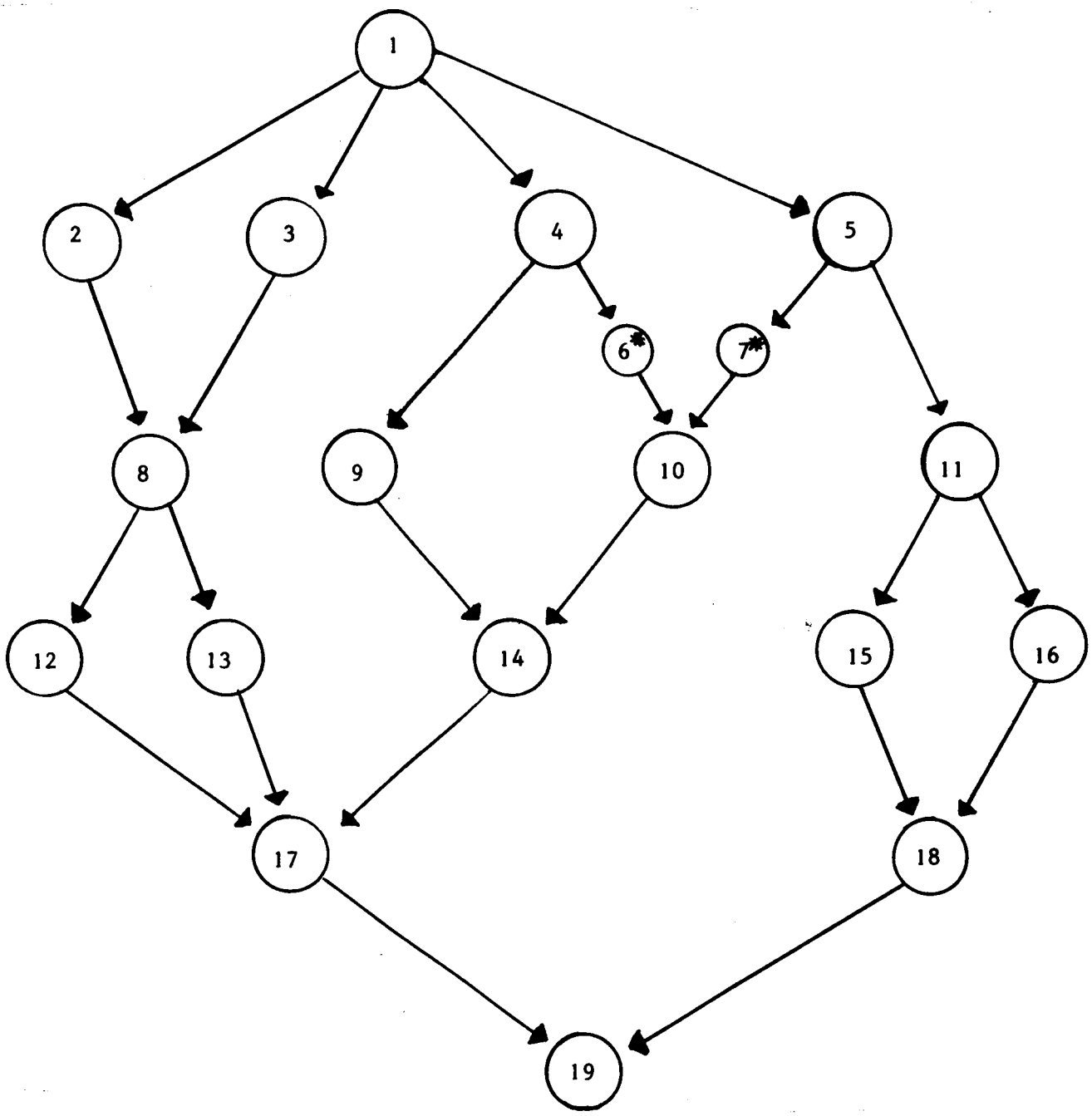


Fig 5-VI Graphe théorique d'exécution

Les noeuds 6 et 7 ne participent pas à l'exécution mais seulement à la construction.

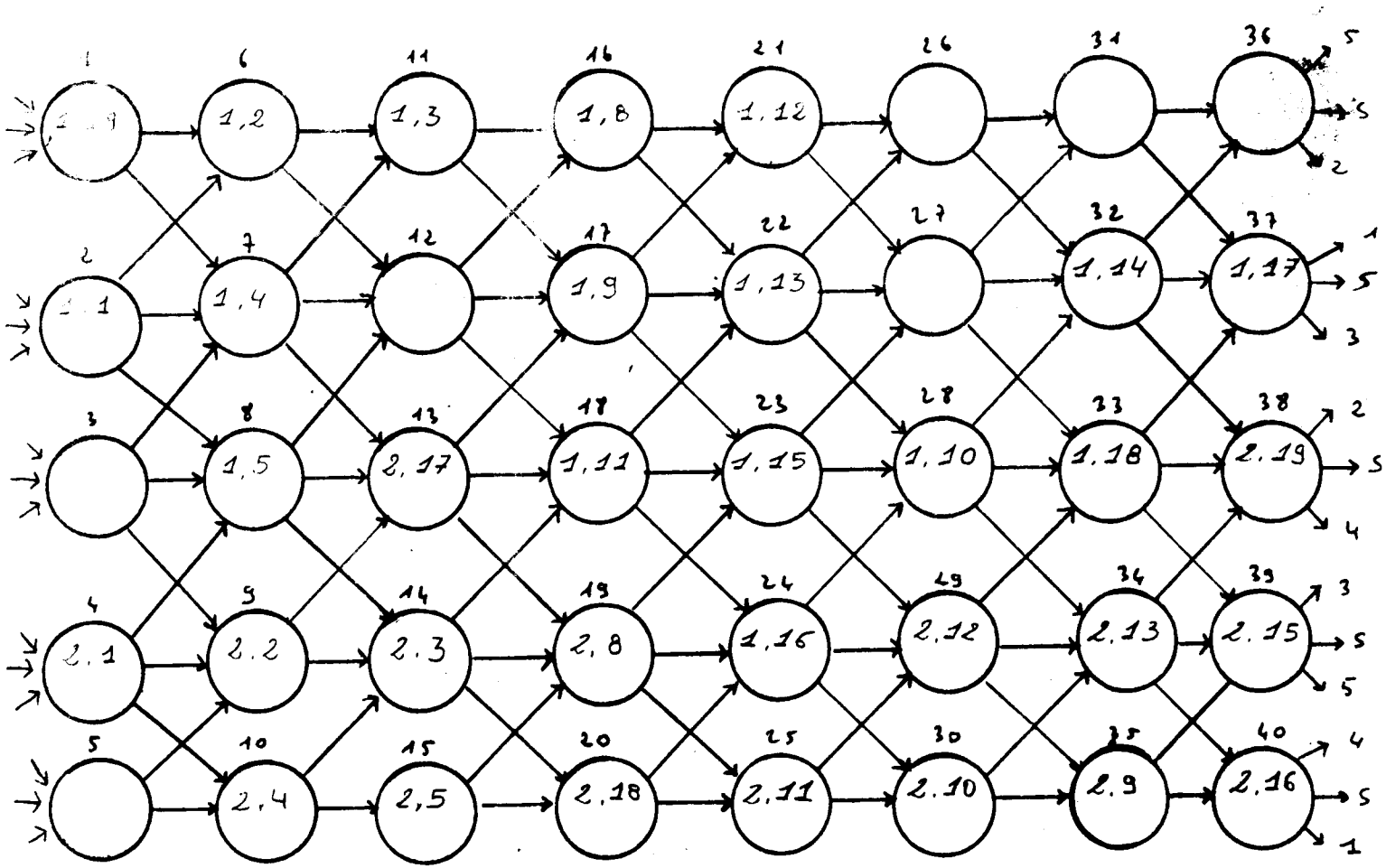
SUS LILLE

N° essai \ moyennes	l	l_1	l_2	T.O.G. (%)	N.T.L.	T.O.L. (%)
essai 1	2,39	1,87	2,91	85 %	3,7	97,5 %
essai 2	1,95	1,48	2,42	85	3,2	85
essai 3	1,60	1,48	1,74	85	2,92	85
essai 4	1,63	1,18	2,08	87	3,05	90
essai 5	1,45	1,22	1,68	87	2,82	87
essai 6	1,28	1,26	1,30	75,5	1,1	89
essai 7	1,52	1,26	1,78	75,5	2,5	77
essai 8	1,3	1,29	1,31	75,5	2,28	84

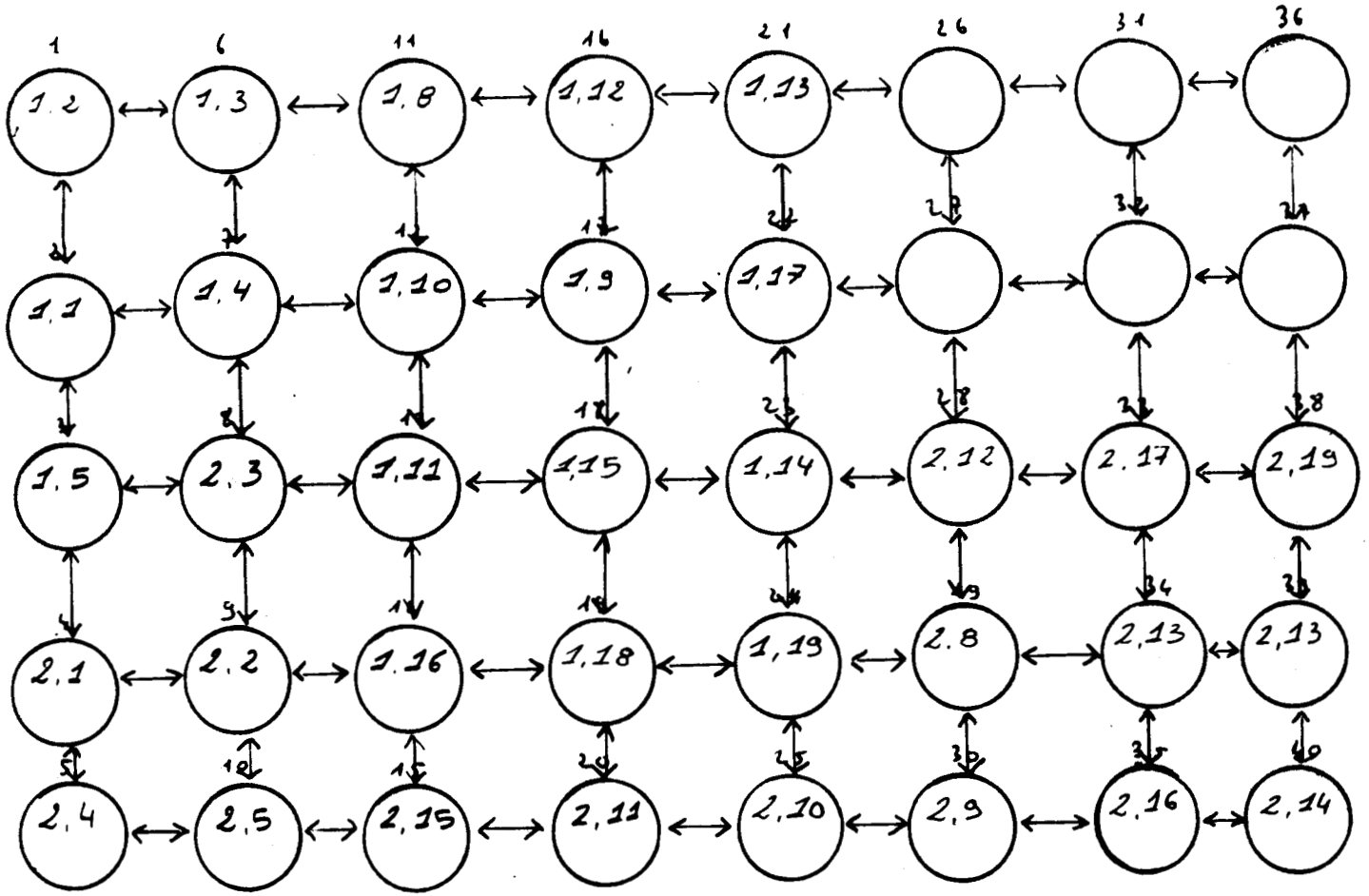
TABLEAU I-5 Moyennes obtenues lors de différents essais en utilisation statique.

BUS LILLE

BUS LILLE

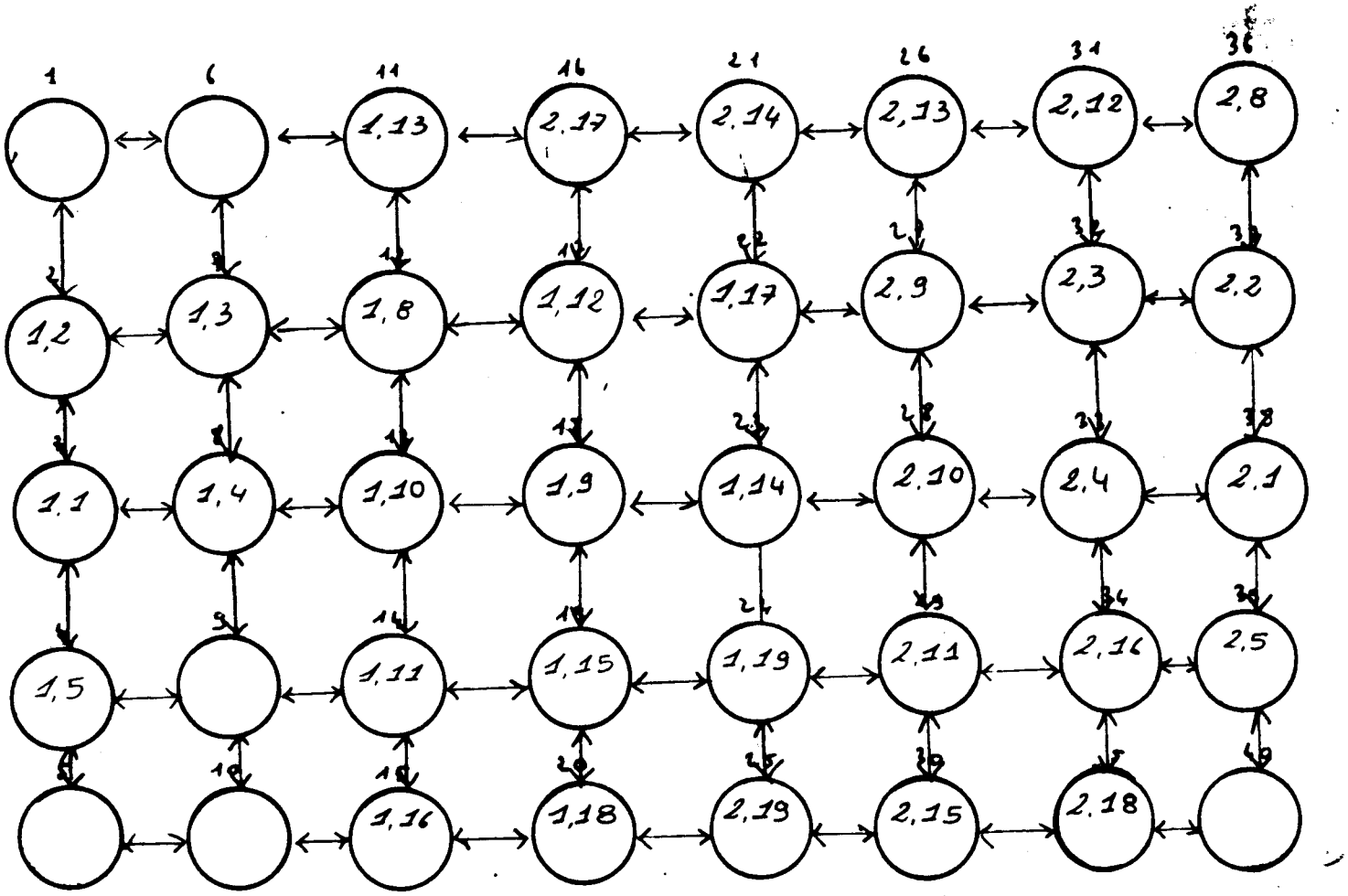


essai n° 1 maillage anisotrope 2 D - 3 prédécesseurs - 3 successeurs
- clôture torroïdale partielle.

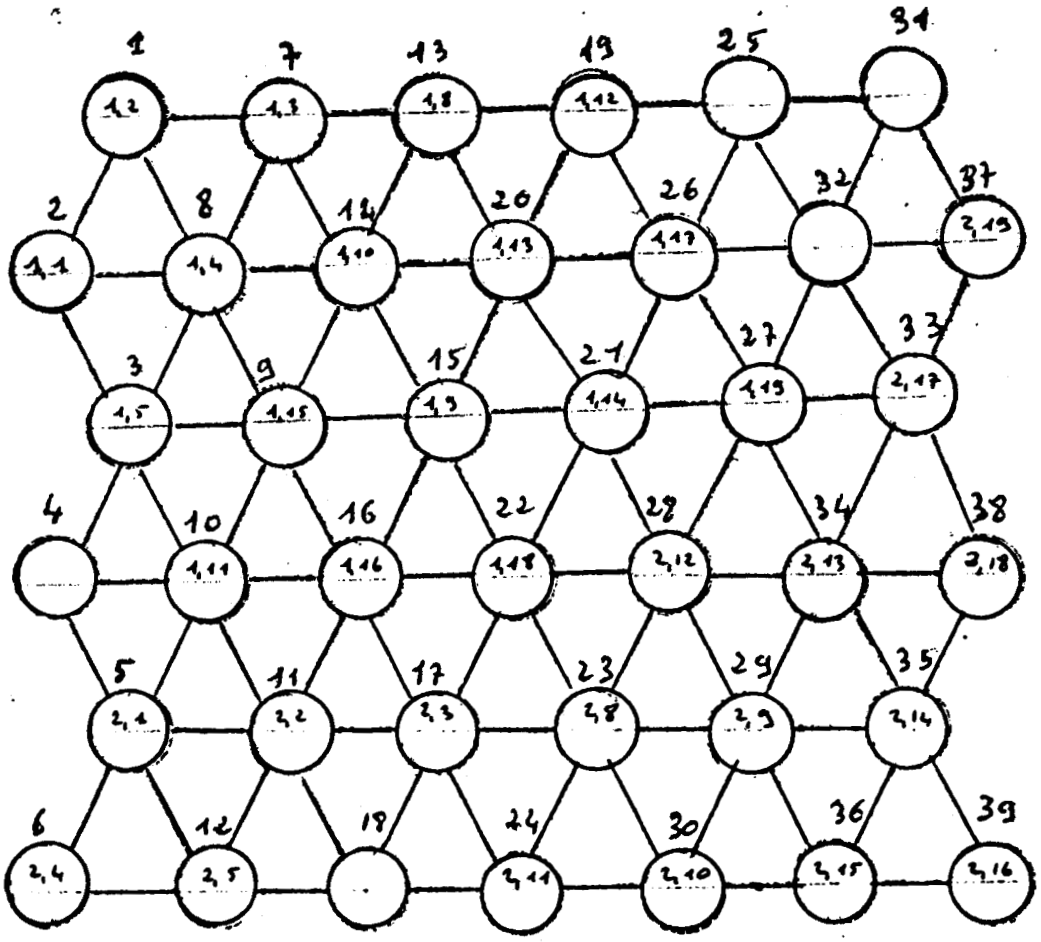


essai n° 2 maillage carré bidirectionnel - Entrée : cellules extrême gauche
Sortie : cellules extrême droite

BIBLIOTHEQUE

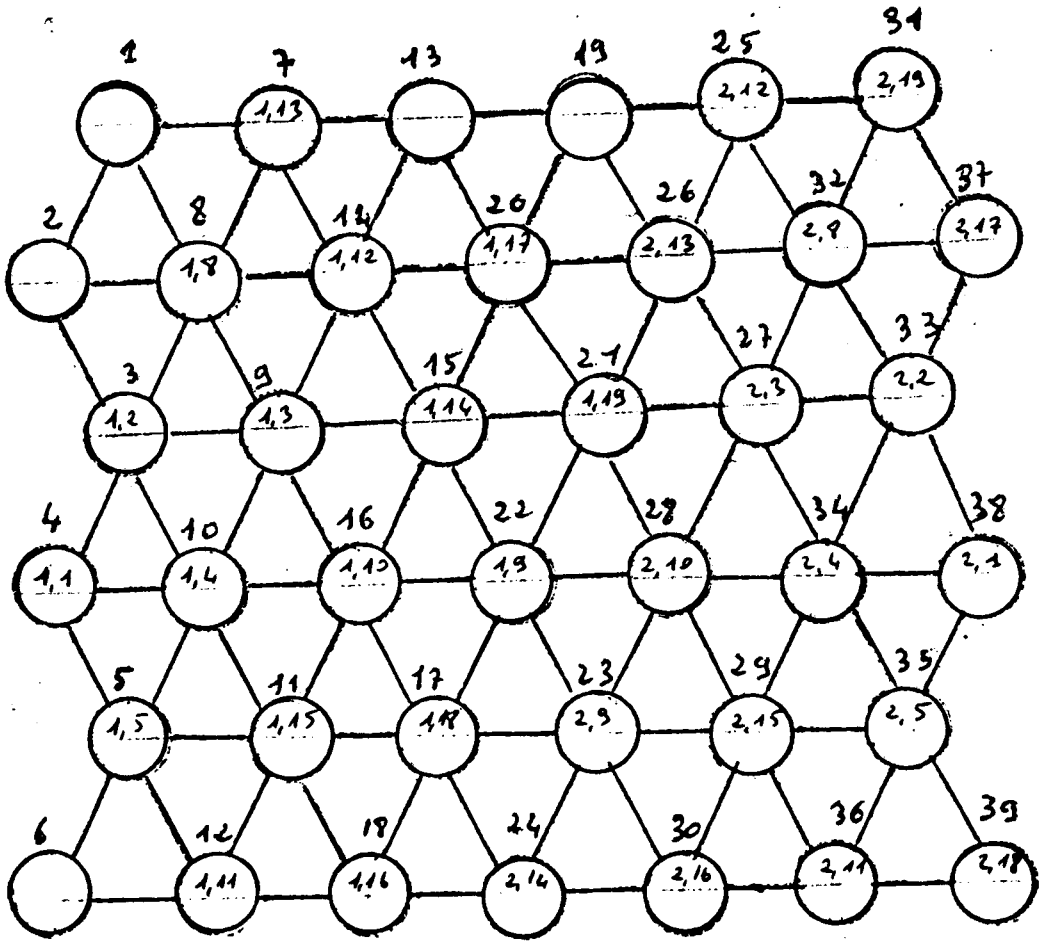


essai n° 3 : maillage carré bidirectionnel E/S sur les bords



essai n° 4 - Maillage hexagonal (6 voisins) bidirectionnel

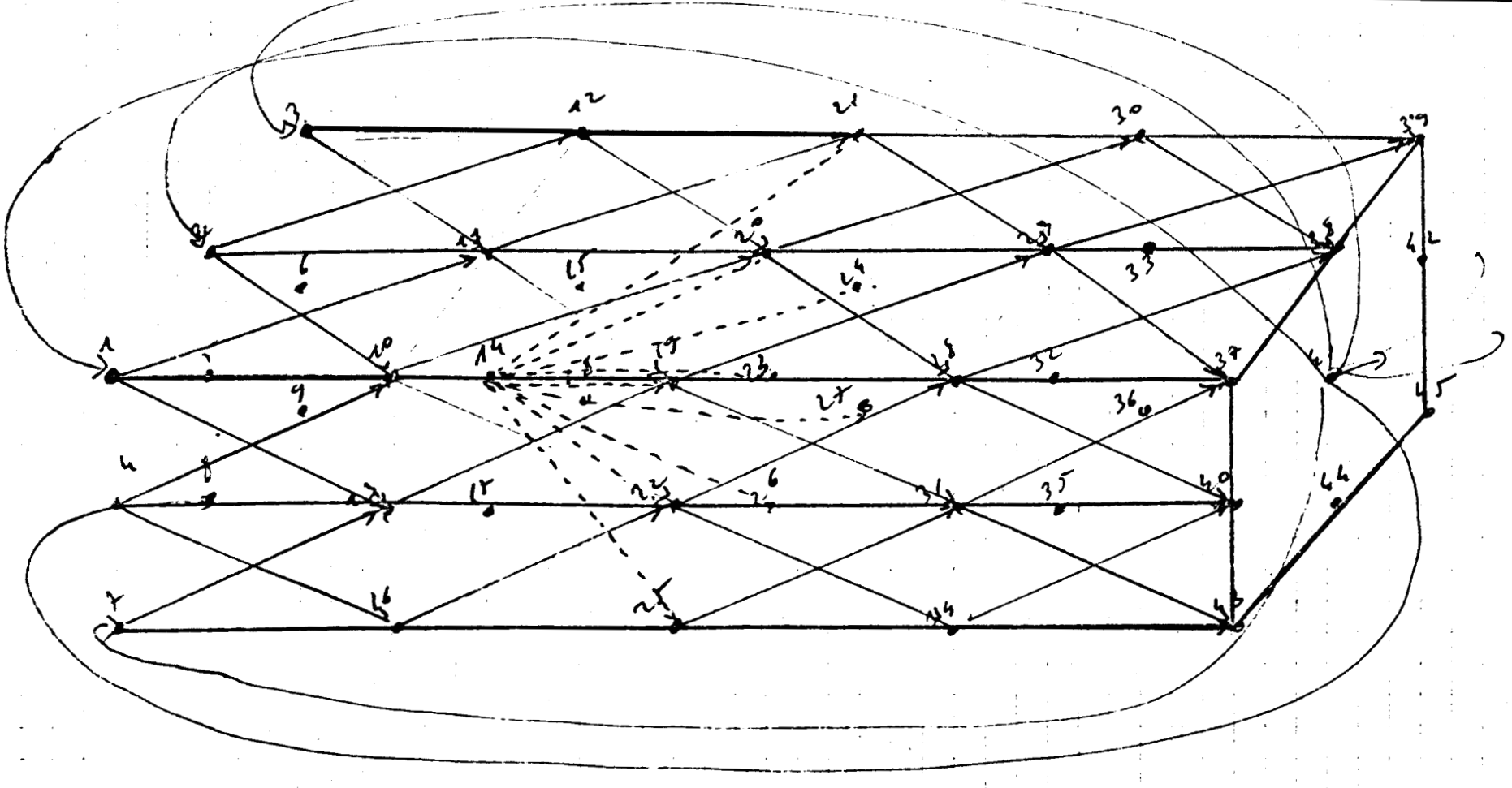
- 6 cellules d'entrée côté gauche
- 6 cellules de sortie côté droit.



essai n° 5 Maillage hexagonal

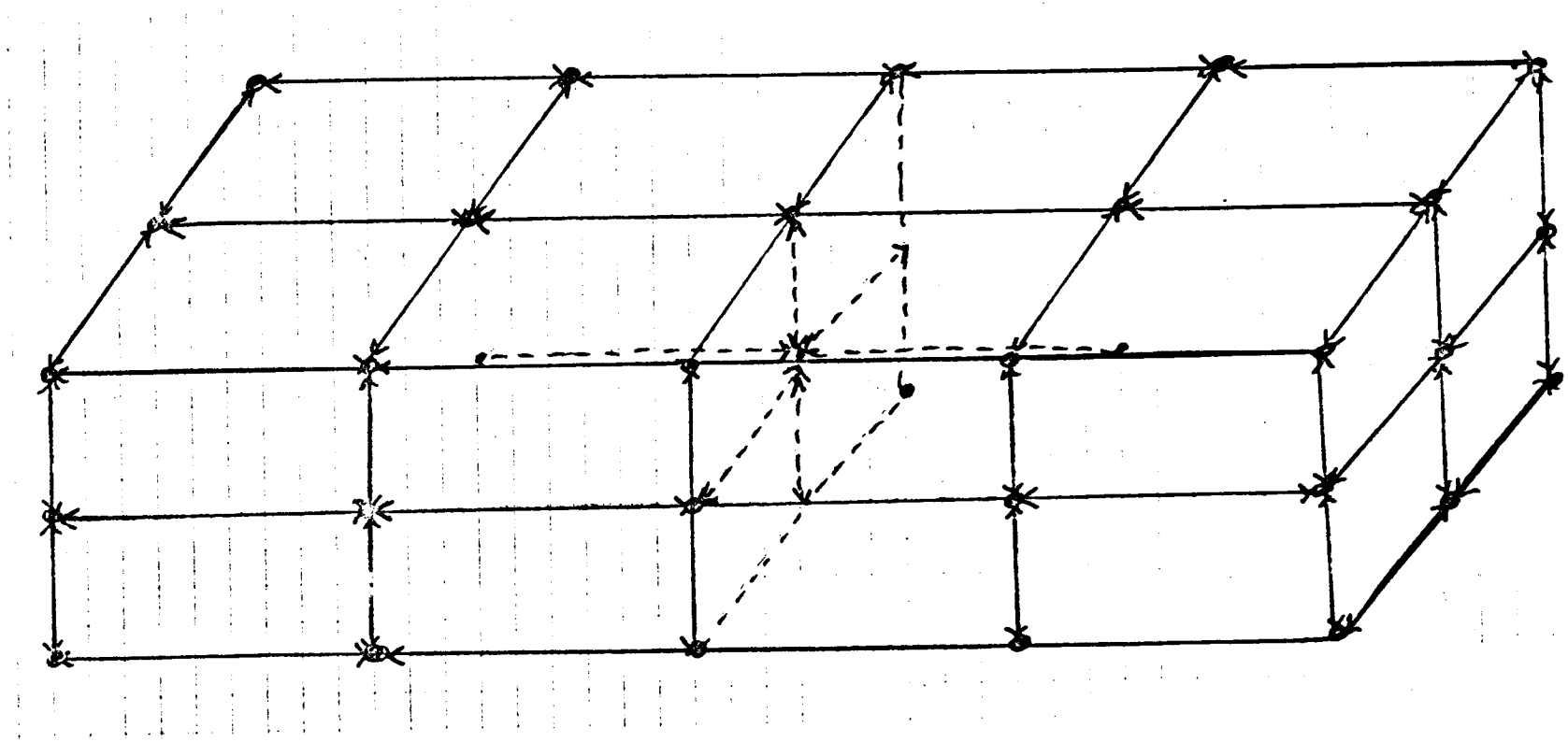
Toutes les cellules périphériques sont utilisables en E/S.

BUS LILLE



essai 6. 3 dimensions.

- maillage anisotrope 9 suc. 9 Pred.
- clôture torroïdale partielle.



essai 7	maillage cubique	- E et S séparés (gauche et droite)
essai 8	maillage cubique	- E et S commun sur la périphérie

5.12.2 FONCTIONNEMENT DYNAMIQUE

On fait se succéder des occurrences de tâches avec à chaque fois reconstruction du graphe d'exécution. Le temps d'exécution d'un noeud est supposé prendre deux unités de temps alors que pour construire le successeur d'un noeud, une seule unité de temps est nécessaire quelque soit la "fonction graphe" effectuée. Compte tenu de la faible connexité du graphe de référence, on a choisi la structure à trois voisins pour illustrer cet exemple.

5.12.2.1. FRÉQUENCE D'INJECTION = 0,5

On injecte de nouvelles occurrences de tâches dans l'espace cellulaire à la fréquence $\frac{1}{2}$.

Au bout d'un temps t = 12, on obtient un régime permanent dont on a résumé dans le tableau suivant quelques caractériques.

Taux occupation global (TOG) = Taux occupation cellule (TOC) =

$$\frac{1}{NT}$$

où N = nombre de processeurs

T = période de la mesure

δ_{ij} = 1 si la cellule j est occupée à l'instant i

δ_{ij} = 0 sinon.

$\alpha >$ to, temps établissement régime permanent.

Taux réservation global (TRG) = taux réservation cellule (TRG) =

$$\frac{1}{NT} \sum_{i=\alpha}^{\alpha+T} \sum_{j=1}^{N} \delta'_{ij}$$

δ'_{ij} = 1 si la cellule j est réservée à l'instant i

= 0 sinon.

$$l = \text{longueur moyenne des chemins} = \frac{1}{T} \sum_{i=\alpha}^{N+T} li \text{ où}$$

li est la longueur moyenne des chemins au temps i.

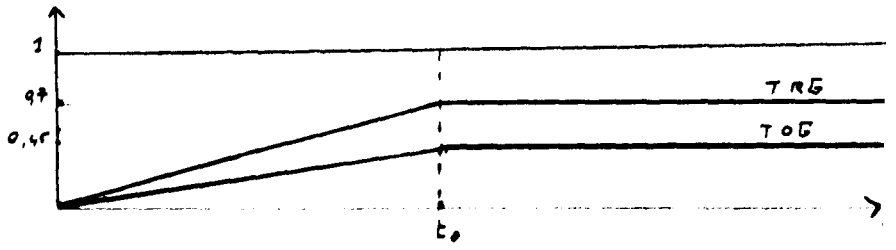
N T L et T.O.L. : déjà explicités.

T O G	45 %
T R G	70 %
1.	2,33
N T L	2,80
T O L	97 %

5.12.2 - Fréquence d'injection = 1

Le taux d'occupation de l'espace cellulaire et faible dans le cas précédent. Il faut alors noter que si on veut augmenter ce dernier sans modifier les conditions préalables de l'expérience précédente, on arrive à une situation de blocage, artificielle en ce sens qu'elle est uniquement dûe au fait que la vitesse de réservation étant le double de la vitesse d'éxécution e les deux opérations lancées en même temps, des messages constructeurs vont se trouver rapidement sans correspondants, puisque la quasi totalité du niveau inférieur sera saturée bien avant l'établissement du régime permanent. Les deux figures suivantes expliquent ce phénomène dans notre cas précis :

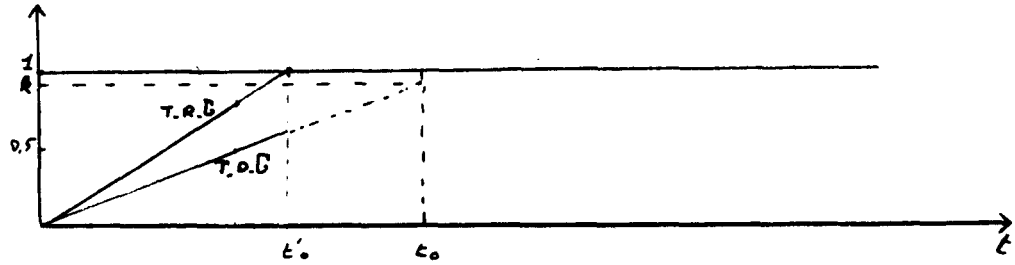
Fréquence
injection = 0,5



to: temps établissement
régime permanent.
A to : libérations et réservations s'équilibrent.

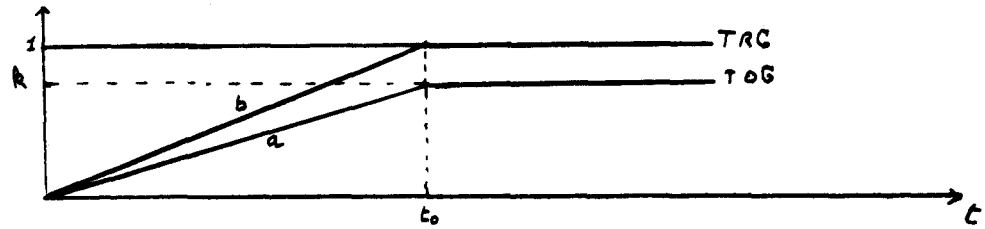
Si on augmente le taux occupation global en doublant la fréquence d'injection, l'espace cellulaire est entièrement réservé avant l'établissement du régime permanent et tout lancement de nouvelle tâche se traduit par des messages bouclant inutilement et se génant entre les différentes ocurences de tâches.

Fréquence injection = 1



k = le taux occupation global recherché.

On a supposé que les taux T-O-G et T-RG étaient en progression linéaire (ce qui n'est plus vrai lorsqu'on approche de la saturation comme on va le voir par la suite). Dans ce cas les conditions pour éviter tout blocage artificiel apporté par les conflits de réservation tout en gardant le comportement simple défini dans le paragraphe précédent devient :



si : a vitesse d'exécution

b vitesse construction.

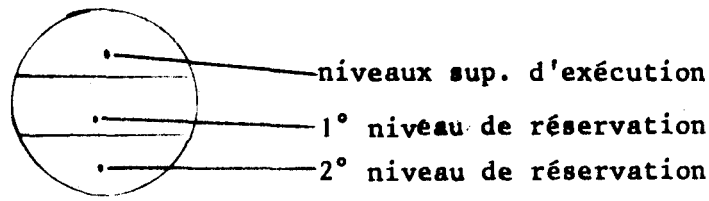
k T-O-G désiré.

$$\begin{array}{l} a.to=k \\ b.to \leq 1 \end{array} \rightarrow \frac{b}{a} \leq \frac{1}{k}$$

Nous avons ici un $\frac{b}{a}$ égal a 1/2 et donc pour atteindre les 85 % d'occupation du fonctionnement statique qui est notre objectif, il faut modifier les conditions de l'expérience. On a adopté deux approches différentes pour résoudre ce problème.

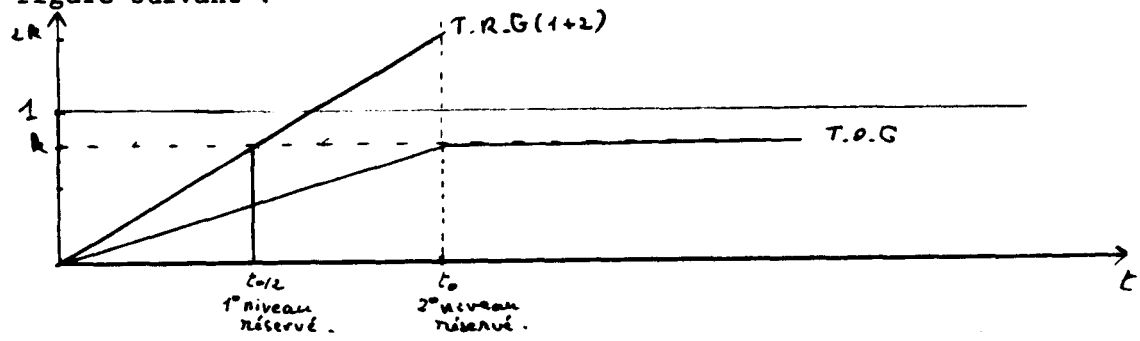
A - On suppose l'existence au niveau inférieur de deux niveaux de réservation et non plus un seul.

symbole.



Une cellule est considérée comme libre tant qu'elle n'est pas en double réservation, la scrutation (cf. 5.10) se faisant à trois niveaux au lieu de deux afin qu'une cellule élise comme successeurs les cellules les moins chargées.

En partant de l'hypothèse d'une progression linéaire des T O G et T R G avant le régime permanent, on espérait que cette solution s'approcherait du cas de figure suivant :



Le régime permanent est atteint avant la saturation des deux niveaux de réservation tout en permettant le taux d'occupation visé (en l'occurence les 85 % du comportement statique).

L'expérience a montré dans ce cas que notre hypothèse d'une progression linéaire était fausse dès lors qu'on s'approchait d'un T O G de 70 %.

La figure 5.12 donne les nombres de l'évolution de divers paramètres déjà définis (T O G, T R G, 1). On constate que le taux d'occupation atteint un maximum de 70 % pour décroître ensuite. L'explication de ce comportement décevant est que localement et tout à fait aléatoirement des cellules ont élu des successeurs au niveau 2 alors que par ailleurs des cellules non voisines restaient libres. La répétition de plus en plus fréquente de ce phénomène amenait des conflits de réservation puis d'exécution entre des occurences de tâches successives qui interdisaient d'atteindre les 85 % espérés.

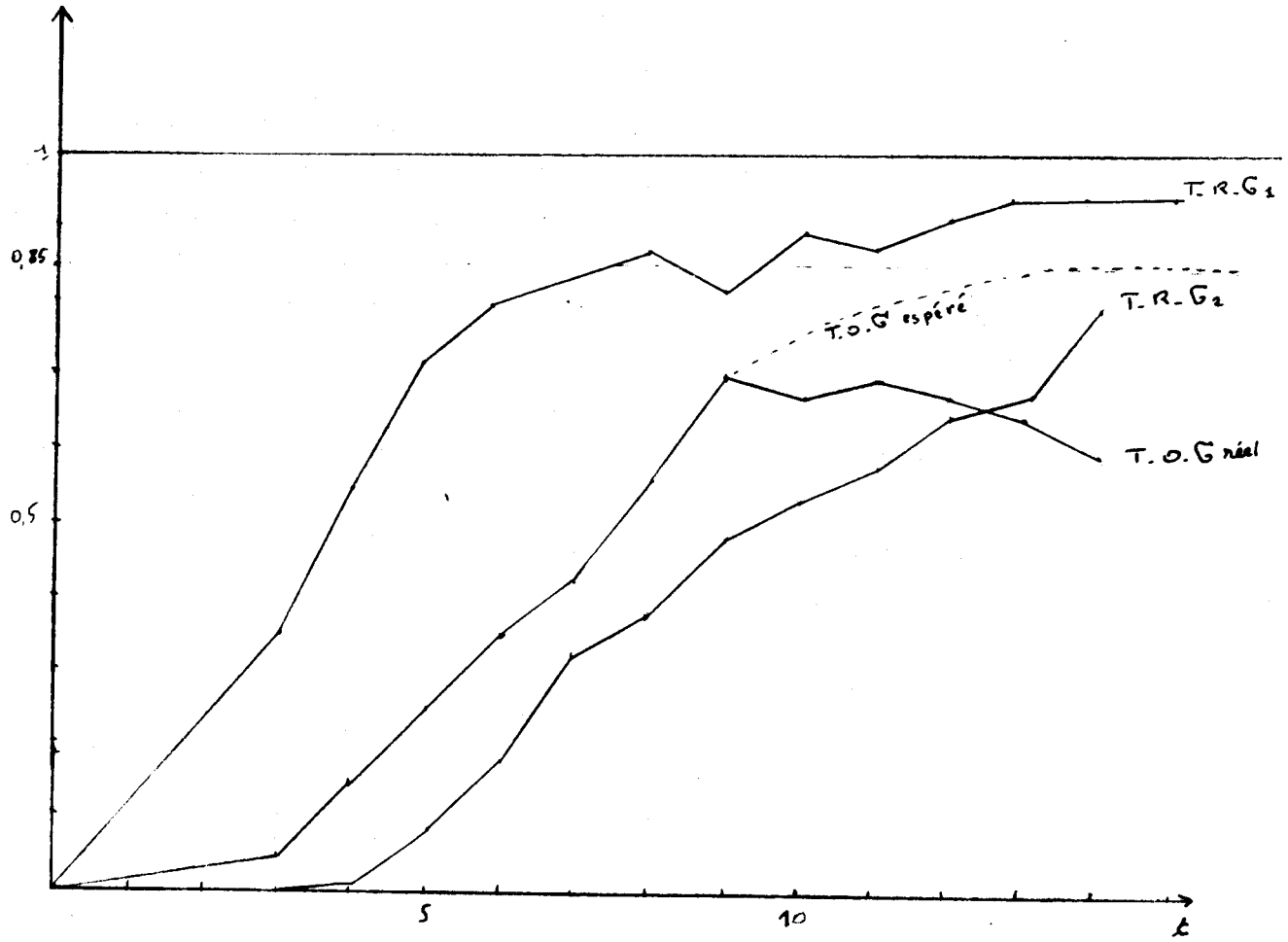
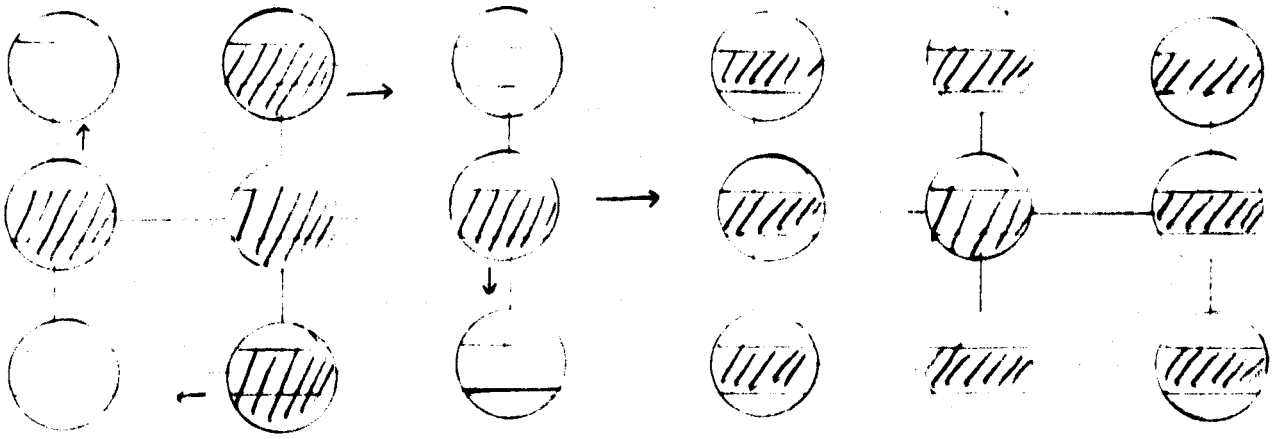


Fig. : 5.12 - Courbes TOC, TRG = f(t)

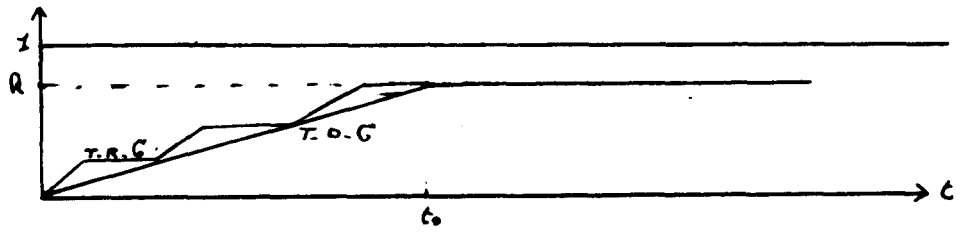
En fait la modification que nous avions apporté au comportement initial était incomplète. Il est vraisemblable qu'en sus du niveau supplémentaire de réservation, la solution complète était de donner des possibilités de "diffusion" (25), c'est-à-dire permettre à une cellule ayant ses deux niveaux inférieurs réservés de pouvoir éventuellement se décharger sur des voisines qui seraient libres, des réservations les plus récentes.



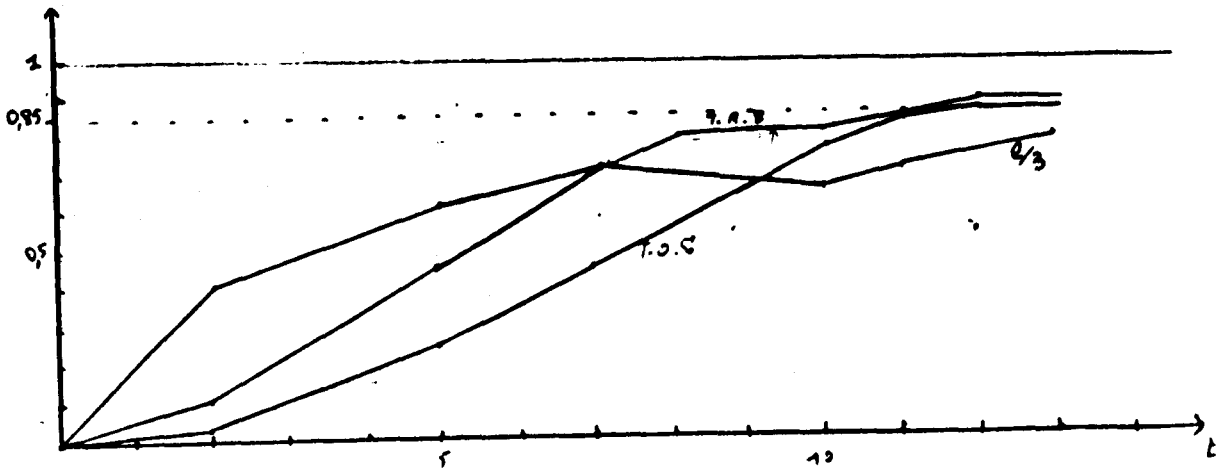
BUS LILLE

B - La 2° solution envisagée pour éviter le phénomène de blocage dit artificiel a été de ne pas construire le graphe tout entier avant l'exécution mais seulement un niveau après l'autre de sorte à garder l'avantage de l'anticipation de la construction avant l'exécution tout en évitant des conflits inutiles entre les constructions de différentes occurences de tâches.

On revient à un seul niveau de réservation. La modification du comportement initial est que lorsqu'une cellule reçoit un message constructeur, elle traite l'entête mais ne passera les "messages suite" que lorsque le niveau supérieur va commencer l'exécution. On a aligné en quelque sorte l'évolution de la construction sur l'exécution de la façon suivante :



On obtient bien alors un régime permanent avec le taux d'occupation de 85 % espéré et une longueur moyenne de chemin raisonnable (cf figure). Il faut ajouter que les expériences menées jusqu'à présent, du type de la précédente, ont été basées sur des bases trop peu générales et simplifiées pour donner lieu à des conclusions formelles. D'autres expériences qui tiennent compte de facteurs écartés ici telle la variation des temps d'exécution ou des temps de construction selon la nature des "fonctions graphes" (jonction, éclatement) et de leurs degrés, la longueur des chemins et des retards qu'ils introduisent devraient nous permettre d'aller plus loin.



CONCLUSION

Au cours de cette étude exploratoire, nous avons vu que le concept d'architecture cellulaire de processeur peut se retrouver dans deux formes différentes de réalisation. La première est la réalisation de structures spécialisées, basées sur l'emploi de composants L S I bon marché car largement commercialisé. La seconde est la réalisation d'une puissante machine (milliards d'instructions par seconde) caractérisées par une redondance inévitable des unités de traitement et mémoire, la contre partie de cette redondance étant une grande insensibilité à des pannes ponctuelles d'éléments de traitements, l'abandon de la notion de contrôleur centralisé "actif" qui caractérise la plupart des systèmes existant. Cela réduit à néant la notion d'optimalité qu'on pouvait être tenté d'attacher à la présence dans un système de traitement d'un "deus ex machina", homme ou O S, qui possédant une vision globale de la situation aurait été à même de prendre les meilleures décisions concernant la gestion du système. Il est apparu que cette notion d'optimalité est déjà très relative dans le cas de quelques processeurs travaillant sur des tâches interactives ou issues d'un graphe de précédence "touffu", particulièrement si les traitements considérés sont très dépendants des valeurs de données, les problèmes étant en général NP-complete*. Nous pensons qu'elle est tout à fait illusoire si on envisage l'emploi simultané d'un très grand nombre de processeurs échangeant fréquemment de l'information. La proposition que nous avons faite va dans ce sens. A partir d'une définition préalable des traitements et des échanges d'information qu'ils nécessitent, la réalisation physique de ces traitements et échanges se fait indépendamment de toute action externe grâce à la circulation de divers types de message entre des éléments dont le comportement est fonction à la fois du contenu des messages et de la connaissance qu'ils ont de leur voisinage immédiat.

Par ailleurs nous ne voulons pas que cet abandon volontaire paraisse un moyen facile d'éviter un problème en l'escamotant. En effet peut-être existe-t-il des modes d'utilisation de vastes architectures multimicro-ordinateurs qui aient un caractère plus déterministe et plus efficace. Le problème reste ouvert. En outre nous sommes conscients que l'étude entreprise est fort incomplète, à la fois dans son but avoué qui était d'aboutir à une architecture mais aussi dans le contexte plus large au sens informatique qui veut que réaliser une machine n'est pas une finalité en soi mais doit s'inscrire dans un système de traitement qui soit utile et utilisable, ce qui implique l'existence de besoins d'utilisation et de moyens d'utilisation, essentiellement le langage. Si donc l'étude architecturale peut se poursuivre avec notamment

* NP - complete = no polynomial deterministic time

le développement de l'automate de niveau inférieur qu'on pourrait émuler par une maquette à base de 8 X 300 ou AMD 2900 en relation avec un microprocesseur V L S I 16 bits pour émuler le processeur de niveau supérieur, une exploration des besoins ainsi que des moyens d'utilisation s'avère nécessaire si on veut garder à l'étude sa cohérence car il faut admettre que l'approche "machine graphe" et message ne nous situe que l'embryon d'un langage machine intermédiaire. Ce sont là des prolongements naturels de la recherche entreprise sans compter les développements d'éventuelles architectures spécialisées, les plus rentables certainement à court terme.

BIBLIOGRAPHIE

1 - AKERS S.B.
"A rectangular logic array"
I.E.E.E. Trans on Comp. Vol C 21 n° 8 Août 1972.

2 - AGRAWAL D.P.
"High speed arithmetic arrays"
I.E.E.E. Trans on Comp Vol C 28 n° 3 Mars 1979.

3 - AMELING W
"The development and application of digital differential analyzers"
Parallel Computers - Parallel Mathematics - IMACS 1977.

4 - ARBIB
"Theories of abstract automata"
Englewood Cliffs - N.J. 1969.

5 - BAER J.L.
"A survey of some theorical aspects of multiprocessing"
Computing surveys Mars 1973.

6 - BARNES AND ALL
"The Illiac IV computer"
I.E.E.E. Trans on Comp Vol C 17 n° 8 August 1968.

7 - BATCHER K.E.
"Sorting networks and their applications"
AFIPS Proceedings. Spring joint Computer Conference 1968.

8 - BODE A. - HAENDLER W.
"Classification d'architectures parallèles : introduction de la notation E.C.S. et application au projet E.G.P.A."
Rairo Informatique/Computer Science Vol 12 n° 4 1978.

9 - BRINTON J.B.
"New architectures goes with flow"
Electronics April 1979

10 - CHASSERY J.M.
"Quelques problèmes liés à l'analyse d'images numériques obtenus par un système automatique de microphotométrie"
Thèse 3° cycle - Grenoble 1976.

11 - CHEN T.C.
"Overlap and parallel processing" dans
"Introduction to computer architecture" de H.S. Stone - (S.R.A.) 1972.

12 - CENNAMO F. DE CARLINI U.
"A new ultra fast digital differential analyzer employing multimicroprocessor's techniques"
Euromicro Worshop Nice June 1975.

13 - CODD E.F.
"Cellular automata"
New York : Academic Press 1968.

14 - COOPER R.G.
"The distributed pipe-line"
I.E.E.E. Trans on Comp Vol C 26 n° 11 Nov 1977.

15 - CORDONNIER V.
"Processeur de communication cellulaire pour un système de traitement en parrallèl
Journées I.R.I.A. St Pierre de Chartreuse 1973.

16 - CORDONNIER V.
"Architectures pipe-lines"
Ecole de l'I.R.I.A. Juin 1978. Lille.

17 - COUSIN R.
"Etude et définition d'un processeur cellulaire de communication à haut degré de parallélisme"
Thèse de 3° cycle. Université de Lille I - 1976.

18 - DEGRENDEL B.
"Communications et protocoles dans une architecture multiprocesseurs"
Thèse de Docteur-Ingénieur. Université de Lille I 1977.

19 - DESPAIN D. and PATTERSON D.A.
X-tree : "a tree structured multi-processor Computer architecture"
Proceedings of the 5th annual symposium as Computer architecture. Avril 1978.

20 - FLYNN M.
"Some computer organization and their effectiveness"
I.E.E.E. Trans on Comp Vol C 21 n° 9 Sept. 1972.

21 - FRECON L.
"Reseaux logiques microcellulaires"
Thèse d'Etat - Université Claude Bernard - Lyon 1975.

22 - FRERICHS R.R. - PRAWDA J.
"A computer simulation model for the control of rabies in an urban area of Colombia"
Management Science Vol 22 n° 4 Dec. 1975.

23 - GREGORY J. - Mc REYNOLDS R.
"The Solomon Computer"
I.E.E.E. Trans on Comp Vol C 12 Dec. 1963.

24 - GUILD M.H.
"Fully iterative fast array for binary multiplication and addition"
Electronics letter. 5. 1969.

25 - HALSTEAD JR. - WARD S.A.
"The Mu.NET : a scalable decentralized architecture for parallel computation"
Proceedings of the 7th annual symposium on Computer Architecture 1980.

26 - HÄNDLER W.
"Aspects of parrallelism in Computer architecture"
Parallel Computer - Parallel Mathematics IMACS 1977.

27 - HEPPNER D.
"Can D.D.A. like computers replace the analog Computer"
Parallel Computers - Parallel Mathematics IMACS 1977.

28 - HOFFMAN W.
"Queing models for parallel processing and their application to a hierarchically organized multiprocessing system"
1st European Conférence on Parallel and distributed Processing. Février 1979. TOULOUS

29 - INTEL 8087
Electronic design. May 10. 1980

30 - JENSEN J.E.
"A graphical representation of tasks for multiprocessing"
Proceedings of A.C.M. annual Conference. October 1977.

31 - JUMP S.R. FRITSCHE D.R.
"Microprogrammed arrays"
I.E.E.E. Trans on Comp. Vol C 21 n° 9 Sept. 1972.

32 - KAUTZ W.M.
"An augmented content adressed memory array for implementation with large scale integration"
Journal of A.C.M. Vol 18 n° 1 Janv. 1971

33 - KEMPKEN E. AMELING W.
"Parallel D.D.A. based on universal microprogrammable computing units
Parallel Computer. Parallel Mathematics. IMACS 1977.

34 - KOBER R. KOPP M. KUZNIA CH.
"SMS 101"
Euromicro Newsletter. Avril 1976.

35 - KUHL J.G.
"Distributed fault tolerance for large multiprocessor system "
Proceedings of the 7th annual symposium on Computer architecture. 1980.

36 - KUKRIA S.N. and CHEN I.N.
"Combinational and sequential cellular structures"
I.E.E.E. Trans on Comp Vol C 22 n° 9 Sept. 1973.

37 - LAWRIE D.H.
"Access and alignment of data in an array processor"
I.E.E.E. Trans on Comp. Vol C 24 n° 12 Dec 1975.

38 - LEBART L. FENELON J.P.
"Statistique et Informatique appliquées"
DUNOD

39 - LINDENMAYER A. ET CULIK III. K.
"Parallel graph generating and graph recurrence systems for multicellular development"
J. Général Systems. 3. 1976

40 - LIPOVSKY G.J.
"On a varistructured array of microprocessors"
I.E.E.E. Trans on Comp Vol C 26 n° 2 Feb. 1977

41 - LIPOVSKY G.J.
"On virtual memories and micronetworks"
Proceedings of the 4th annual Symposium on Computer architecture. 1977.

42 - MAC DUFF J.B. - WATSON D.M. - FOUNTAIN T. - SHARS G.K.
"A cellular logic array for image processing"
Pattern Recognition Vol 5 Sept. 1972.

43 - MAITRA K.K.
"Cascaded switching networks of two-input flexible cells"
I.E.E.E. Trans on Comp Vol C 11 n° 4 April 1962.

44 - MALEK M.
"A comparison connection assignment for diagnosis of multiprocessor systems"
Proceedings of the 7th annual symposium on Computer Architecture. 1980.

45 - MARTIN D.F. ESTRIN G.
"Experiments on models of computations and systems"
I.E.E.E. Trans on Comp Vol C 16 n° 1 Feb. 1967.

46 - MAWSON. J.B.
"Pacer 600 autopatch computing system"
Parallel Computer - Parallel Mathematics. IMACS 1977.

47 - MINNICK R.C.
"Cutpoint cellular logic"
I.E.E.E. Trans on Comp. Vol C 13 n° 12 Dec 1964.

48 - MINSKY M.L.
"Computation : finite and infinite machines"
Englewood Cliffs Prenctice Mall 1967.

49 - NOURAI F. - KASHEFF R.S.
"A universal four state cellular computer"
I.E.E.E. Trans on Comp Vol C 24 n° 8 Aug 1975.

50 - OKADA Y. - TAJIMA M. - MORI R.
"A novel multiprocessor array"
2nd Symposium on Microarchitecture - Euromicro Venice 1976.

51 - PATTERSON D.A. - FEHN E.S. - SEQUIN C.M.
"Design consideration for the V.L.S.I. processor of X-tree"
Proceeding of the 6th annual symposium on compute architecture. 1978.

52 - PEASE M.C.
"The indirect binary n-cube microprocessor array"
I.E.E.E. Trans on Comp. Vol C 26 n° 5 May 1977.

53 - RAMAMOORTHY C.V. AND GONZALEZ M.J.
"A survey of techniques for recognizing parallel processable streams in computer programs"
AFPIS Fall joint on Computer 1969.

54 - REDDAWAY S.F.
"DAP a distributed array processor"
First annual symposium on Computer Architecture. 1973.

55 - ROUCAIROL G.
"Contribution à l'étude des équivalences syntaxiques et transformations de programmes parallèles"
Thèse de Doctorat d'Etat. Université Pierre et Marie Curie 1978.

56 - ROY B.
"Algèbre moderne et théorie des graphes"
DUNOD

57 - SAYRE G.E.
"STARAN : an associatine approach to multiprocessor architecture"
RAIRO informatique. Vol 10 n° 5 May 1976.

58 - SCHULZ H.D.
"Garden of Eden Konfigurationen"
Publication de l'Informatik - Forschungsgruppen - 1977 - Darmstadt.

59 - SEQUIN S.H. - DESPAIN A.M. - PATTERSON D.A.
"Communication in X-tree, a modular multiprocessor system"
Proceedings of 1978 Annual Conference of A.C.M.

60 - SHANTE V. AND KIRKPATRICK S
"An introduction to percolation theory"
Advanced Physics Vol 20 - 1971.

61 - SIEGEL M.J. - Mc MILLEN - R.J. MUILLER J.R.
"A survey of interconnection methods for reconfigurable parallel processing systems"
A FIPS Proceedings of National Computer Conference 1979.

62 - SIGNETICS.
notices techniques : microprocesseurs bipolaires et MOS
Ed 1978

63 - SIZER TRH
"The digital differential analyzer : an incremental computer"
Ed : Chapman and Hall.

64 - SULLIVAN M. - BASHKOW T.R.
"A large scale homogemous, fully distributed parallel machine"
Proceedings of the 4th annual symposium on Computer Architecture 1977.

65 - SUNG C.H.
"Testable Sequential Cellular arrays"
I.E.E.E. Trans on Comp Vol C 25 n° 1 Janv. 1976

66 - TEXAS INSTRUMENT
TMS 9940 : 16 bits microcomputer data manuel.

67 - VERNEL P.
"Conception et réalisation d'un multimicrocalculateur temps réel à grande sureté de fonctionnement"
Thèse de Doctorat d'Etat - Nancy 1977.

68 - VOLLMAR R
"Cellular spaces and parallel algorithms- an introductory survey"
Parallel Computers. Parallel Mathématics - IMACS 1977. Felmeier ed.

69 - WALLACE C.S.
"A suggestion for a fast multiplier"
I.E.E.E. Trans on Comp Vol C 13 n° 2 Fev. 1964.

BIBLIOTHEQUE UNIVERSITAIRE
Section de SCIENCES
LILLE