Synthèse et simulation d'algorithmes systoliques

Ibrahima Sakho

THESE

présentée à

L'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

pour obtenir

LE TITRE DE DOCTEUR DE 3EME CYCLE
MATHEMATIQUES APPLIQUEES
(OPTION ANALYSE NUMERIQUE)

par

Ibrahima SAKHO

*
**

SYNTHESE ET SIMULATION D'ALGORITHMES SYSTOLIQUES

**
*

Soutenue le 3 Avril 1987 devant la Commission d'Examen

JURY

PRESIDENT : Monsieur François ROBERT

EXAMINATEURS :

Messieurs Jean DELLA DORA

Maurice TCHUENTE

Michel COSNARD

Traïan MUNTEAN

Yves ROBERT

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Président : Daniel BLOCH

Année 1987

Vice - Présidents : René CARRE
Jean-Marie PIERRARD

Professeurs des Universités

BARIBAUD Michel	ENSERG
BARRAUD Alain	ENSIEG
BAUDELET Bernard	ENSPG
BEAUFILS Jean-Pierre	ENSEEG
BESSON Jean	ENSEEG
BLIMAN Samuel	ENSERG
BLOCH Daniel	ENSPG
BOIS Philippe	ENSHMG
BONNETAIN Lucien	ENSEEG
BOUVARD Maurice	ENSHMG
BRISSONNEAU Pierre	ENSIEG
BRUNET Yves	IUFA
BUYLE-BODIN Maurice	ENSERG
CAILLERIE Denis	ENSHMG
CAVAIGNAC Jean-François	ENSPG
CHARTIER Germain	ENSPG
CHENEVIER Pierre	ENSERG
CHERADAME Hervé	UFR PGP
CHERUY Arlette	ENSIEG
CHIAVERINA Jean	UFR PGP
CHOVET Alain	ENSERG
COHEN Joseph	ENSERG
COUMES André	ENSERG
DARVE Félix	ENSHMG
DELLA-DORA Jean	ENSIMAG
DEPORTES Jacques	ENSPG
DOLMAZON Jean-Mar	ENSERG
DURAND Francis	ENSEEG
DURAND Jean-Louis	ENSIEG
FONLUPTJean	ENSIMAG
FOULARD Claude	ENSIEG
GANDINI Alessandro	UFR PGP
GAUBERT Claude	ENSPG
GENTIL Pierre	ENSERG
GREVEN Hélène	IUFA
GUERIN Bernard	ENSERG
GUYOT Pierre	ENSEEG
IVANES Marcel	ENSIEG
JAUSSAUD Pierre	ENSIEG
JOUBERT Pierre	ENSIEG
JOURDAIN Geneviève	ENSIEG
LACOUME Jean-Louis	ENSIEG
LESIEUR Marcel	ENSHMG
LESPINARD Georges	ENSHMG
LONGEQUEUE Jean-Pierre	ENSPG
LOUCHET François	ENSEEG
MASSE Philippe	ENSIEG
MASSELOT Christian	ENSIEG
MAZARE Guy	ENSIMAG
MOREAU René	ENSHMG
MORET Roger	ENSIEG
MOSSIERE Jacques	ENSIMAG
OBLED Charles	ENSHMG
OZIL Patrick	ENSEEG
PARIAUD Jean-Charles	ENSEEG
PAUTHENET René	ENSIEG
PERRET René	ENSIEG
PERRET Robert	ENSIEG
PIAU Jean-Michel	ENSHMG
POUPOT Christian	ENSERG
SAUCIER Gabrielle	ENSIMAG
SCHLENKER Claire	ENSPG
SCHLENKER Michel	ENSPG
SERMET PIERRE	ENSERG
SILVY Jacques	UFR PGP
SIRIEYS Pierre	ENSHMG
SOHM Jean-Claude	ENSEEG
SOLER Jean-Louis	ENSIMAG
SOUQUET Jean-Louis	ENSEEG
TROMPETTE Philippe	ENSHMG
VEILLON Gérard	ENSIMAG
ZADWORNY François	ENSERG

Professeur Université des Sciences Sociales (Grenoble II)

BOLLIET Louis

Personnes ayant obtenu le diplôme d'HABILITATION A DIRIGER DES RECHERCHES

BECKER Monique
BINDER Zdenek
CHASSERY Jean-Marc
COEY John
COLINET Catherine
COMMAULT Christian
CORNUEJOLS Gérard
DALARD Francis
DANES Florin
DEROO Daniel
DIARD Jean-Paul
DION Jean-Michel
DUGARD Luc
DURAND Robert
GALERIE Alain
GAUTHIER Jean-Paul
GENTIL Sylviane
PLA Fernand
GHIBAUDO Gérard
HAMAR Sylvaine
LADET Pierre
LATOMBE Claudine
LE GORREC Bernard
MADAR Roland
MULLER Jean
NGUYEN TRONG Bernadette
TCHUENTE Maurice
VINCENT Henri

Chercheurs du C.N.R.S

Directeurs de recherche 1ère Classe

CAILLET Marcel
CARRE René
FRUCHART Robert
JORRAND Philippe
LANDAU Ioan
MARTIN

Directeurs de recherche 2ème Classe

ALEMANY Antoine
ALLIBERT Colette
ALLIBERT Michel
ANSARA Ibrahim
ARMAND Michel
BINDER Gilbert
BONNET Roland
BORNARD Guy
CALMET Jacques
DAVID René
DRIOLE Jean
ESCUDIER Pierre
EUSTATHOPOULOS Nicolas
JOUD Jean-Charles
KAMARINOS Georges
KLEITZ Michel
KOFMAN Walter
LEJEUNE Gérard
MERMET Jean
MUNIER Jacques
SENATEUR Jean-Pierre
SUERY Michel
TEDOSIU
WACK Bernard

Personnalités agrées à titre permanent à dirige des travaux de recherche (décision du conseil scientifique)

E.N.S.E.E.G

BERNARD Claude
CHATILLON Catherine
CHATILLON Christian
COULON Michel
DIARD Jean-Paul
FOSTER Panayotis
HAMMOU Abdelkader
MALMEJAC Yves
MARTIN GARIN Régina
SAINTFORT Paul
SARRAZIN Pierre
SIMON Jean-Paul
TOUZAIN Philippe
URBAIN Georges

E.N.S.E.R.G

BOREL Joseph
CHOVET Alain
DOLMAZON Jean-Marc
HERAULT Jeanny

E.N.S.I.E.G

DESCHIZEAUX Pierre
GLANGEAUD François
PERARD Jacques
REINISCH Raymond

E.N.S.H.G

BOIS Daniel
DARVE Félix
MICHEL Jean-Marie
ROWE Alain
VAUCLIN Michel

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

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

E.F.P.G

CHARUEL Robert

C.E.N.G

CADET Jean
COEURE Philippe
DELHAYE Jean-Marc
DUPUY Michel
JOUVE Hubert
NICOLAU Yvan
NIFENECKER Hervé
PERROUD Paul
PEUZIN Jean-Claude
TAIB Maurice
VINCENDON Marc

Laboratoires extérieurs

C.N.E.T

DEMOULIN Eric
DEVINE
GERBER Roland
MERCKEL Gérard
PAULEAU Yves

ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

Directeur : Monsieur M.MERMET
Directeur des Etudes et de la formation: Monsieur J. LEVASSEUR
Directeur des recherches : Monsieur J. LEVY
Secrétaire Général : Mademoiselle M. CLERGUE

PROFESSEURS DE 1ère CATEGORIE

COINDE Alexandre	Gestion
GOUX Claude	Métallurgie
LEVY Jacques	Métallurgie
LOWYS Jean-Pierre	Physique
MATHON Albert	Gestion
RIEU Jean	Mécanique-Résistance des matériaux
SOUSTELLE Michel	Chimie
FORMERY Philippe	Mathématiques Appliquées

PROFESSEURS DE 2ème CATEGORIE

HABIB Michel	Informatique
PERRIN Michel	Géologie
VERCHERY Georges	Matériaux
TOUCHARD Bernard	Physique Industrielle

DIRECTEUR DE RECHERCHE

LESBATS Pierre	Métallurgie

MAITRE DE RECHERCHE

BISCONDI Michel	Métallurgie
DAVOINE Philippe	Géologie
FOURDEUX Angeline	Métallurgie
KOBYLANSKI André	Métallurgie
LALAUZE René	Chimie
LANCELOT Francis	Chimie
LE COZE Jean	Métallurgie
THEVENOT François	Chimie
TRAN MINH Canh	Chimie

Personalités habilitées à diriger des travaux de recherche

DRIVER Julian	Métallurgie
GUILHOT Bernard	Chimie
THOMAS Gérard	Chimie

Professeurs à l'UER de Sciences de Saint-Etienne

VERGNAUD Jean-Maurice	Chimie des Matériaux et Chimie Industrielle

A mes parents et à la mémoire de
ma grand'mère Mama DRAME

J' exprime ma très profonde gratitude à Mr M. TCHUENTE qui a accepté de diriger ce travail ; je le remercie pour la qualité traditionnellement excellente et de ses consultations que même pas l'éloignement n'a pu altérer et de ses relations humaines sans lesquelles ce travail n'aurait pu être mener à bien .

Je suis très sensible à l'honneur que me fait le Professeur F. ROBERT du laboratoire TIM3 en acceptant de présider ce jury . Je le remercie sincèrement de m'avoir souvent consacré de son précieux temps .

A Mr T . MUNTEAN du LGI j'exprime également mes sincères remerciements pour la grande disponibilité dont il a fait preuve au cours de mon initiation à la programmation parallèle et pour avoir accepté de faire partie de ce jury .

Je remercie Mrs J . DELLA DORA , M . COSNARD du laboratoire TIM3 , Y . ROBERT de IBM-ROME , pour l'honneur qu'ils me font en acceptant de faire partie de ce jury .

Je remercie les membres du laboratoire TIM3 et singulièrement ceux de l'équipe ALGORITHMIQUE PARALLELE dont j'ai pu apprécier la cordialité .

J'exprime ma profonde reconnaissance en particulier à Mme Fatou SIDIBE à Bamako , Mme Liliane CHABANIS à Grenoble , Mlle Laurence TIRARD à Grenoble et à tous ceux qui comme elles ont contribué à mener à bien ce travail en m'apportant soutien et réconfort .

Mes très sincères remerciements au service de reprographie pour le soin qu'il a apporté au tirage .

Table des matières

ANNEXE :

Une conjecture sur les cycles limites des réseaux booléens monotones

INTRODUCTION GENERALE

Introduits en 1929 par VON NEUMAN , les réseaux d'automates peuvent être définis d'une manière générale comme un ensemble de cellules interconnectées localement et évoluant dans un temps discret par interactions réciproques . Ils recouvrent tout un ensemble de préoccupations scientifiques allant de la biologie à l'informatique en passant , entre autres , par la physique . Une telle symbiose s'explique par le fait que les réseaux d'automates constituent une structure naturelle de modélisation de tant de domaines qui bien que différents entre eux restent conceptuellement semblables . Aussi , à l'heure où le fonctionnement du cerveau (10 milliards de cellules interconnectées) , le calcul parallèle basé sur la notion de réseau de processeurs , la reconnaissance des formes , pour ne citer que ceux-là , constituent des pôles de recherche très actifs , on comprend que les réseaux d'automates connaissent un regain d'intérêt .

Dans cette thèse que nous voulons une modeste contribution à l'étude des réseaux d'automates , nous nous intéressons à deux aspects : système dynamique et structure pour le calcul parallèle .

Dans l'utilisation des réseaux d'automates comme outils de modélisation , la recherche de nouvelles structures pour le calcul parallèle est l'une des applications les plus prometteuses . Grâce au developpement de la technologie VLSI , cette recherche a conduit à la notion de réseaux systoliques c'est à dire des architectures parallèles spécialisées composées de cellules élémentaires interconnectées de manière locale et régulière . A chaque instant t , chaque cellule reçoit des données des cellules qui lui sont voisines en entrée et après avoir effectué quelques transformations élémentaires elle les envoie aux cellules qui lui sont voisines en sortie . En dépit des avantages qu'offre une telle architecture , il est un problème en général inabordé du fait des techniques de conception utilisées ; c'est celui de la taille des réseaux . En effet , la plupart des réseaux publiés dans la littérature sont composés d'un nombre de processeurs au delà de ce qui est nécessaire .
Dans la première partie de la thèse , nous proposons une méthode de conception de réseaux systoliques dite méthode de partitionnement . A partir de quelques exemples de problèmes d'analyse numérique , nous montrons que la méthode conduit , à partir d'un schéma de calcul donné , à une parallélisation optimale en temps de calcul et quelques fois en nombre de processeurs utilisés . Dans certains cas elle conduit à des solutions meilleures que celles publiées jusqu'ici . Nous proposons également , pour chacun des problèmes traités , une solution aux incontournables problèmes de contrôle qu'induisent les réseaux obtenus .
On n'ignore pas les problèmes soulevés par la preuve formelle de correction d'algorithmes . Les méthodes de synthèse systématique constituent l'une des réponses à ce problème de validité , puisque la validité des algorithmes obtenus est une conséquence de la validité de la méthode générale de synthèse .
Une présentation du langage parallèle OCCAM dont nous nous sommes servis pour simuler sur le VAX-VMS les réseaux exhibés termine cette étude . La présentation de ce langage est illustrée à travers un programme de simulation d'un réseau

systolique pour la détection d'une plus longue sous-suite commune à deux chaînes de caractères .
Lorsqu'on s'intéresse à la réalisation pratique des réseaux systoliques , l'un des problèmes les plus cruciaux est celui de la synchronisation globale . En examinant le problème de plus près , on s'aperçoit que celle-ci peut être avantageusement remplacée par des mécanismes de synchronisation locale : c'est ce qui nous a permis par exemple d'effectuer des simulations en OCCAM où la synchronisation se fait par rendez-vous entre cellules voisines . En fait , dans le cas où toutes les cellules du réseau n'ont pas le même cycle de temps,cette synchronisation locale peut se traduire également par un gain de vitesse pour le réseau . Nous consacrons la deuxième partie de la thèse à l'analyse d'un tel modèle théorique issu des procédures de synchronisation dans certains langages CSP tel que OCCAM . En l'occurence , dans le cadre de processeurs dont le cycle des temps est basé sur une permutation périodique de 1 , 2 , . . . , n nous évaluons les performances du modèle. Nous exhibons alors toutes les permutations qui conduisent à la meilleure performance à savoir celle qui n'engendre aucun retard sur les temps des cycles individuels des processeurs .

En tant que système dynamique , l'étude d'un réseau d'automates consiste en général à préciser son comportement asymptotique ; lorsqu'ils existent , un intérêt particulier est porté à la longueur des cycles limites . Parmi les efforts de formalisation de ce type de problèmes , on peut citer entre autres :
- une approche métrique due à F . ROBERT qui consiste à définir sur l'ensemble des états possibles du réseau une distance et ensuite traduire dans le cas discret des résultats connus du cas continu ;
- une approche inspirée de la physique et qui consiste à définir une fonction d'énergie dont les variations indiquent les transitions du réseau . Ceci a conduit E . GOLES , F . FOGELMAN , G . WEISBUCH à établir des résultats très puissants sur la dynamique des réseaux à seuil .
- une approche basée sur l'analyse de phénomènes continus analogues à l'itération discrète et qui a permis à M . COSNARD et E . GOLES d'obtenir de nombreux résultats sur la dynamique des neurones formels .
Cependant dans la plupart des cas la simulation reste le principal outil bien que souvent elle conduise à des résultats qui ne sont vérifiés que numériquement . Le problème étudié en annexe de la thèse illustre bien ce phénomène . Nous nous y intéressons à la longueur maximum des cycles qu'un réseau évoluant selon un opérateur de GAUSS-SEIDEL associé à un opérateur parallèle est susceptible de décrire . Dans ce but , nous rapportons des faits , observés à partir de simulation sur ordinateur , tendant à confirmer la conjecture faite par M . TCHUENTE à savoir la longueur maximum des cycles générés séquentiellement par un réseau booléen monotone de taille n est $C_{n-1}^{\lfloor (n-1)/2 \rfloor}$. Nous donnons une démonstration formelle de cette observation pour des réseaux de petite taille et des indications qui semblent la confirmer pour des réseaux de plus grande taille .

PREMIERE PARTIE

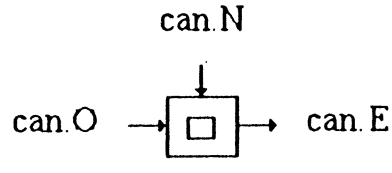
UNE METHODE DE CONCEPTION D'ALGORITHMES PARALLELES POUR RESEAUX REGULIERS

1 - Introduction

Au cours des dix dernières années , on a assisté à un important développement de la technologie des circuits intégrés , ce qui a notamment conduit à de profondes modifications des attitudes dans la conception des architectures des systèmes hautement parallèles . Les architectures qui en ont résulté combinent les deux concepts classiques du parallélisme dans l'espace et du pipeline . Le parallélisme dans l'espace est possible grâce à l'augmentation de la densité d'intégration qu'offre cette technologie . En effet on peut désormais envisager des circuits composés de centaines de milliers de transistors .Un tel circuit , du fait du parallélisme accroit considérablement sa puissance de calcul sans devoir utiliser des technologies ultra rapides et du reste assez coûteuses ; par ailleurs le pipeline minimise énormément ses échanges avec son environnement . Il faut tout de même souligner que les avantages ainsi énumérés supposent résolus les non moins importants problèmes de communication et de synchronisation entre les composants du circuit .
Les réseaux systoliques , introduits en 1978 par KUNG et LEISERSON [10] , sont une alternative dans la recherche d'architectures répondant aux questions soulevées ci-dessus . Depuis leur introduction , ils ont été utilisés avec succès pour la résolution de problèmes provenant de domaines trés variés tels que le traitement d'images [3] , du signal [9] , et des chaînes de caractères [1] , l'algèbre linéaire [11] , etc . . . .
Un réseau systolique est un ensemble de cellules , de structure interne très simple , localement connectées de façon régulière , et à travers lesquelles circulent de manière synchrone des flots de données qui interagissent à chacune de leurs rencontres .
On distingue plusieurs types de réseaux systoliques . En général ils sont classés en fonction de la géométrie des connexions entre cellules ou processeurs .

1) Réseaux linéaires :

Ils sont les plus simples des réseaux systoliques . Ils ont servi à résoudre efficacement de nombreux problèmes .
On peut citer en exemple la résolution d'un système triangulaire $Ax = b$ où A est une matrice triangulaire inférieure d'ordre n .
Un des réseaux qui résolvent ce problème est constitué par une famille de n processeurs dont la structure générale pour $n = 5$ est donnée par la figure 1 . 1 et où les $x(j,0)$ sont initialisés à $b(j)$. Chaque processeur du réseau dispose d'un registre interne et de trois canaux de communication .



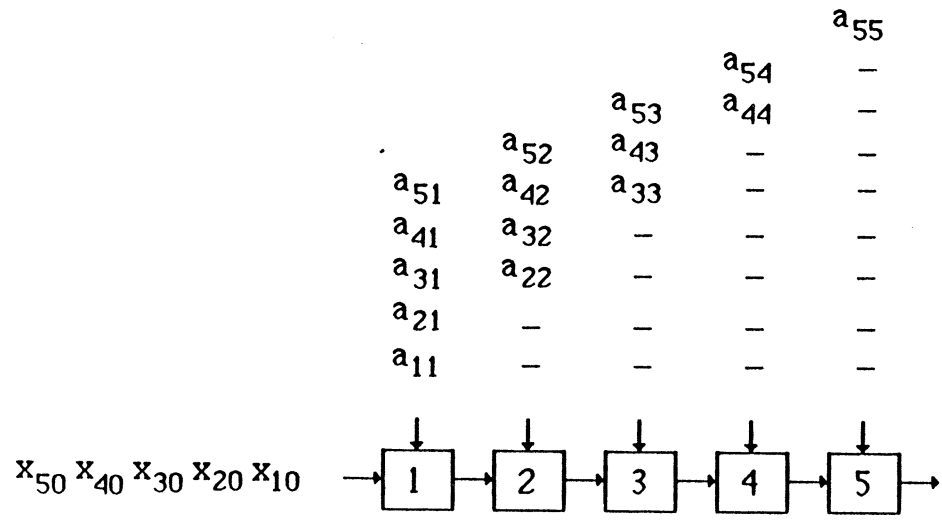
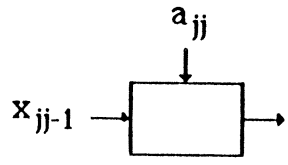
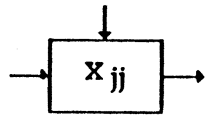


figure 1 . 1

Le fonctionnement d'une cellule en position j dans le réseau comprend trois phases .
- La première débute à la date t = 2(j-1) et dure une unité de temps . La cellule reçoit les valeurs des variables x(j,j-1) , a(j,j) respectivement sur can.O et can.N , calcule x(j,j) = x(j,j-1)/a(j,j) , le stocke dans son registre . Ce premier calcul correspond aux schémas suivants :



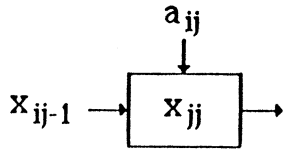
à l'instant t



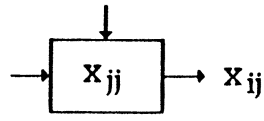
$x_{jj} = x_{jj-1} / a_{jj}$

à l'instant t+1

- La deuxième phase débute à la date 2j-1 et comporte n-(j+1) étapes . A chaque étape , la cellule reçoit en entrées sur can.O et can.N respectivement les valeurs des variables x(i,j-1) et a(i,j) . Après le calcul x(i,j)=x(i,j-1)-a(i,j)*x(j,j) , la valeur de la variable x(i,j) est envoyée en sortie sur can.E .



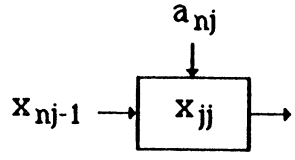
à l'instant t

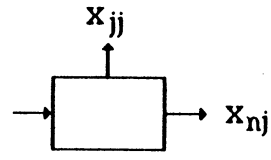


$x_{ij} = x_{ij-1} - a_{ij} * x_{jj}$

à l'instant t+1

- La troisième phase comporte une seule étape qui correspond au dernier calcul de la cellule . Elle se distingue des étapes de la deuxième phase par la sortie sur can.N du contenu du registre .



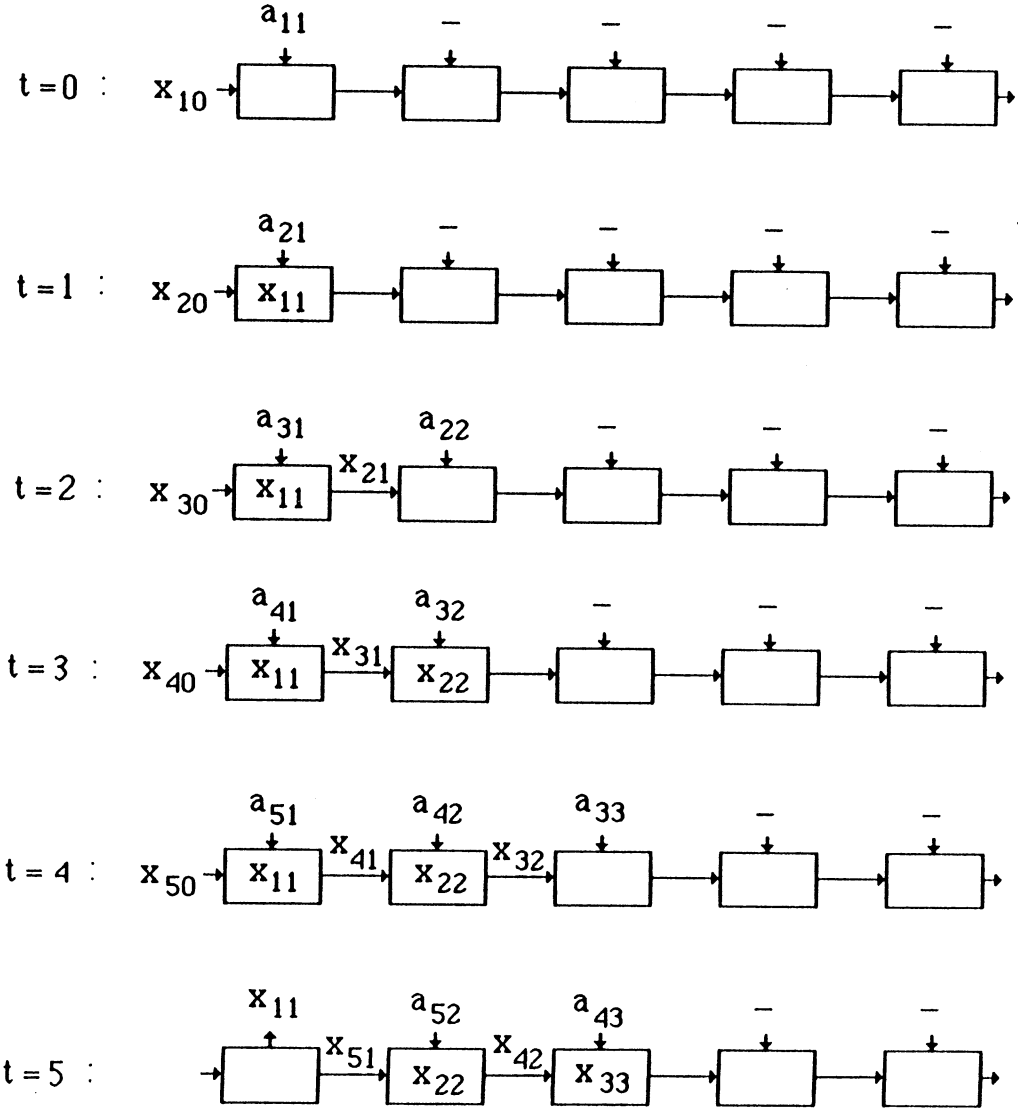


$x_{nj} = x_{nj-1} - a_{nj} * x_{jj}$

à l'instant t

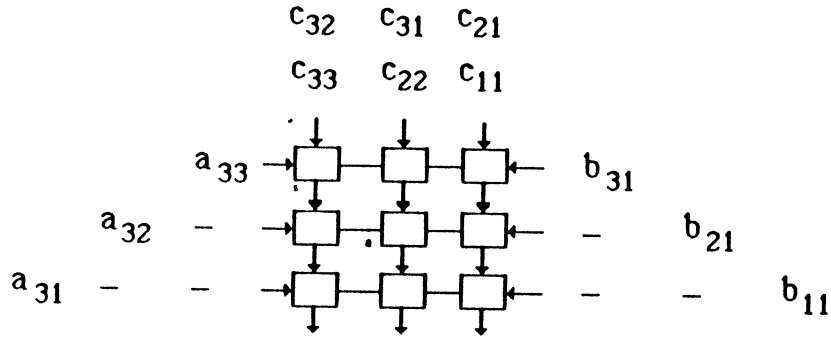
à l'instant t+1

Notons que pour le processeur de rang n , tout ce processus se réduit à la première phase . Pendant les premières unités de temps , le réseau a le fonctionnement suivant

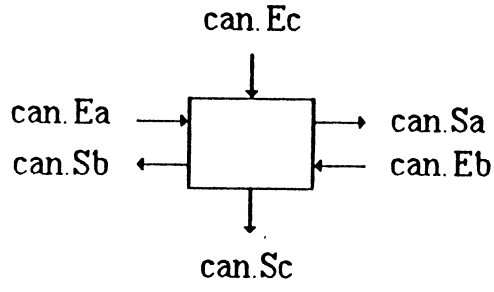


2) Réseaux orthogonaux :

Introduits en 1950 par Von NEUMANN comme modèle d'interconnexion pour la construction d'automates cellulaires permettant d'exhiber des machines auto-reproduisantes, les réseaux orthogonaux présentent la géométrie de connexion entre cellules la mieux connue . Chaque cellule en position (i,j) $\in \mathbb{Z} \times \mathbb{Z}$ est reliée à ses quatre voisines (i-1,j) , (i+1,j) , (i,j-1) , (i,j+1) .
Les réseaux orthogonaux ont été introduits par exemple pour la programmation dynamique [8] . Grâce à une approche combinatoire [15] , des algorithmes très efficaces ont été proposés sur les réseaux orthogonaux pour le calcul du produit C de deux matrices triangulaires A et B d'ordre n . Le réseau correspondant est un tableau de n^2 processeurs dont la structure générale pour n = 3 est la suivante :

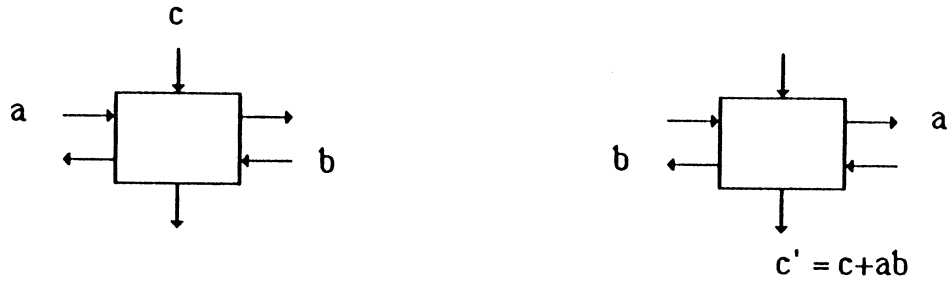


Les entrées A , B , C du réseau sont placées de la manière suivante :
Lorsqu'on balaie de bas en haut les segments de pente 1 (respectivement -1) au dessus du réseau , on voit que ces segments portent les éléments de C appartenant aux lignes numérotées 1 , 2 , . . . , n , n , n-1 , . . . , 2 , 1 (respectivement appartenant aux colonnes numérotées n , n-1 , . . . 1 , 1 , 2 , . . . , n) .
Les variables a_{ik} (respectivement b_{kj}) pour $1 \leq k \leq n$ sont affectées aux droites de pente 1 (respectivement -1) qui contiennent les variables c_{ij} et sont placées à gauche (respectivement à droite) du réseau . La figure ci-dessus illustre cette construction pour i = 3 et j = 1.
Chaque processeur dispose de six canaux de communication avec son voisinage .



Le comportement d'une cellule consiste , au cours de chaque unité de temps à lire les

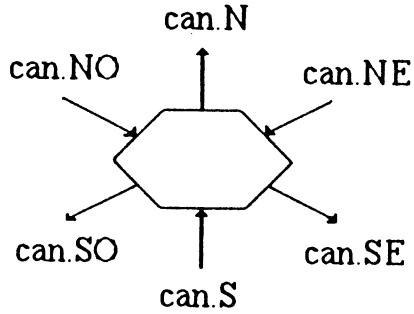
valeurs des variables a , b , c respectivement sur les canaux can.Ea , can.Eb , can.Ec, d'effectuer la transformation c'= c+a*b et d'envoyer en sortie sur can.Sa , can.Sb , can.Sc respectivement les valeurs des variables a , b , c'.



à l'instant t à l'instant t+1

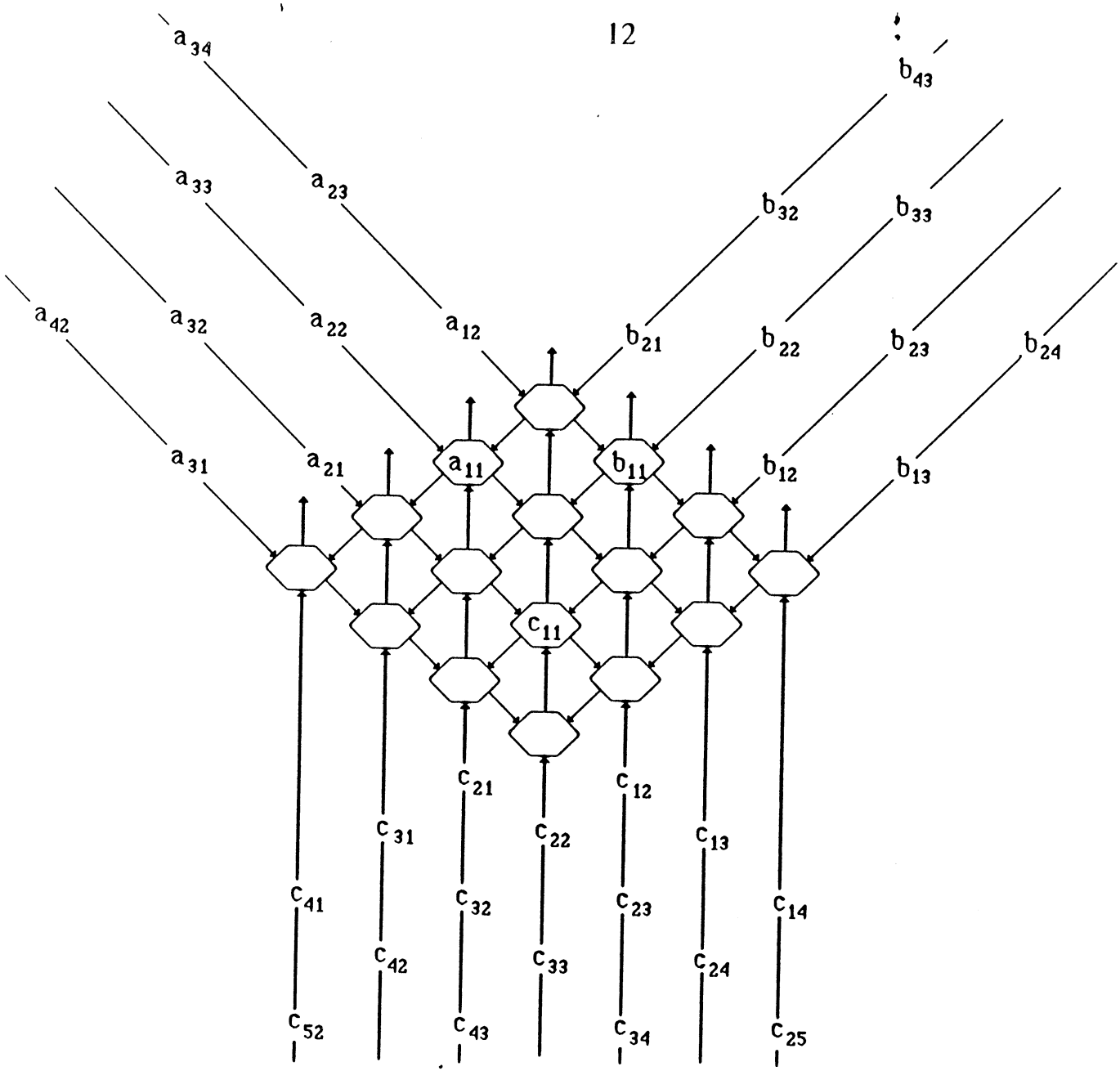
3) Réseaux hexagonaux :

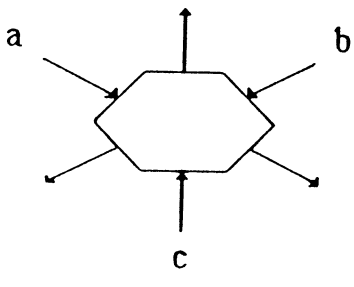
Dérivés des orthogonaux en ajoutant des connexions diagonales , les réseaux hexagonaux sont plus appropriés pour réaliser des opérations sur les matrices bandes . Par exemple pour le produit C de deux matrices bandes carrées A et B d'ordre n , un des réseaux possibles est une famille de l*l' processeurs où l et l' sont les largeurs des bandes de A et B . La figure 1 . 2 illustre le cas où l = 4 et l' = 4 . Chaque processeur dispose de six canaux de communication avec son voisinage .



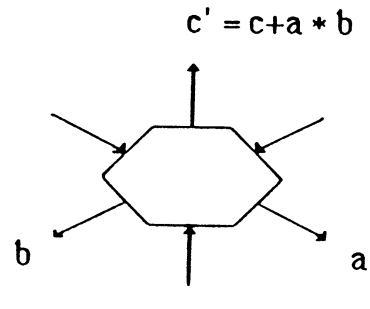
A chaque top d'horloge , le processeur reçoit sur can.NO , can.NE et can.S respectivement les valeurs des variables a , b , c .
Après avoir réalisé la transforlation c'= c+a*b , les valeurs des variables a , b , c' sont envoyées en sortie respectivement sur les canaux can.SE , can.SO , can.N .

La plupart des algorithmes systoliques publiés dans la littérature ont été découverts et sont présentés de manière intuitive . La preuve de validité de tels algorithmes nécessite souvent une simulation à travers quelques pulsations , ce qui , bien que suffisant pose clairement le problème d'une méthodologie de conception d'algorithmes et d'architectures systoliques . Ce problème de conception automatique ou semi-automatique d'algorithmes et d'architectures systoliques a été





à l'instant t



à l'instant t + 1

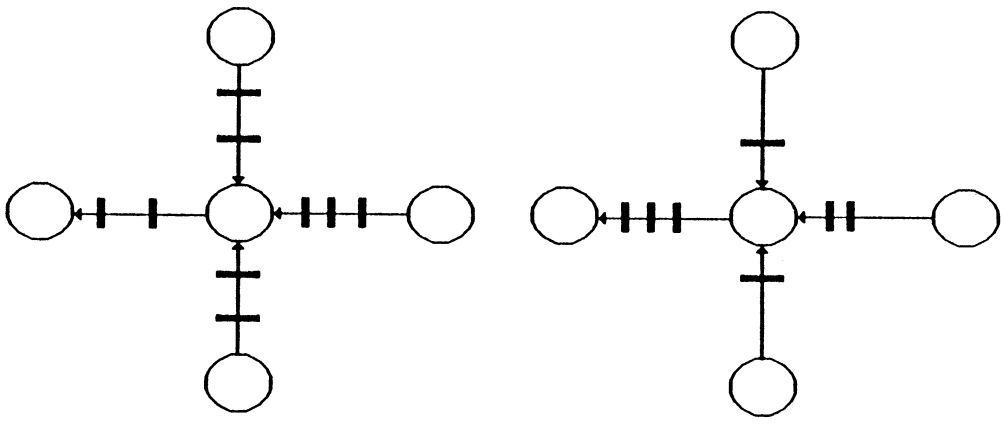
figure 1 . 2

abordé par plusieurs auteurs dont LEISERSON et SAXE [7] , QUINTON [18] , MOLDOVAN [8] , MIRANKER [5] , KUNG [12] , WEISER et DAVIS [2] ... .

LEISERSON et SAXE partent du fait que tout circuit est un système S qu'on peut partitionner en éléments fonctionnels et en registres .
Soit G = (V,E) le graphe de communication de S .

Un sommet v ∈ V représente un élément fonctionnel ; un sommet particulier représente la structure hôte qui permet au système de communiquer avec le monde extérieur .

Chaque arc e ∈ E représente une connexion entre deux éléments fonctionnels . Formellement , chaque arc e est un triplet (x , y , w) où w est un entier dit poids désignant le nombre de registres le long de l'arc joignant x et y .
Par le jeu d'une redistribution des registres , on peut obtenir des systèmes dont le comportement des éléments fonctionnels n'est en rien modifié vis à vis de ses voisins et dont le nombre de registres sur tout circiut élémentaire reste invariant . Il suffit en effet de retirer un registre sur chaque arc entrant d'un sommet et d'en rajouter un sur chaque arc sortant de la même cellule .



avant après

Il est alors clair qu'une condition nécessaire pour que de telles transformations conduisent à un circuit systolique est que sur chaque circuit élémentaire de G le nombre de registres soit supérieur ou égal au nombre d'arcs .
LEISERSON et SAXE ont montré que cette condition est aussi suffisante d'où la méthodologie suivante pour la conception d'architectures systoliques :
- Définir un réseau R simple et en général non systolique où à chaque top d'horloge les résultats doivent , grâce à une logique combinatoire se propageant en cascade , s'accumuler le long d'un chemin sans registre .
- Déterminer le plus petit entier k tel que le réseau R_k , obtenu de R en multipliant par k le poids de tous les arcs , soit systolisable .

- Systoliser R_k à l'aide de la transformation précédente . On obtient ainsi un réseau systolique S_k pour la résolution du problème posé .

Par cette méthode ont pu être obtenus certains réseaux systoliques pour les problèmes de mots [16] .
L'inconvénient de cette méthode est qu'elle conduit à un réseau systolique k fois moins rapide que le réseau synchrone de départ . M . TCHUENTE et Y . ROBERT [17] ont montré que dans le cas de problèmes décomposables , tels que la convolution généralisée , une technique de type "divide and conquer" permet de déduire à partir de S_k un réseau systolique S* de même vitesse que le système synchrone S .

D'un autre côté , P. QUINTON , D.I.MOLDOVAN et MIRANKER ont développé une approche basée sur une formulation en termes d'équations récurrentes . Ils associent à un problème (P) , généralement linéaire :
- Un domaine de calculs D , en général une partie de $\mathbb{N}^2$ ou $\mathbb{N}^3$;
- Un ensemble de variables indicées par les points de D ;
- Un système d'équations qui définit en chaque point de D les calculs qui lui sont associés .

La construction d'une architecture systolique comporte alors deux étapes :
- Définir une fonction de temps T qui à chaque point x de D associe la date d'exécution T(x) des calculs qui lui correspondent . Cette fonction de temps doit être telle que $T(x) < T(x')$ si les résultats des calculs en x sont nécessaires à ceux à effectuer en x' .
- Définition d'une fonction d'allocation A qui alloue les points du domaine de calculs D à m processeurs . Cette fonction doit être compatible avec la fonction de temps , c'est à dire que deux calculs de D ayant la même date d'exécution ne doivent pas être affectés au même processeur . Cette fonction est obtenue par projection .

En guise d'exemple de réseau obtenu par cette méthode , on peut citer le réseau linéaire pour la résolution d'un système triangulaire de la figure 1.1 . En effet ce réseau n'est rien d'autre qu'une projection dans la direction (0,1) du plan (i,j) du réseau de la figure 2.1 que nous présentons en détail plus loin .

Commentaire :

Le caractère quasi-affine de la fonction de temps et de la fonction d'allocation comporte plusieurs avantages :
- d'abord cela permet d'utiliser les outils mathématiques d'algèbre linéaire et d'analyse convexe , et donc de formuler de manière rigoureuse les résultats obtenus
- ensuite les réseaux obtenus sont bien réguliers et les cellules de base ont une structure assez simple .

Ces méthodes de projection permettent dans certains cas d'obtenir des algorithmes optimaux en temps mais rarement optimaux en nombre de processeurs utilisés . Par

exemple pour le produit de matrices carrées d'ordre n on obtient un réseau dont le temps est 3n-2 mais qui nécessite $3n^2$-o(n) processeurs . Pour la triangularisation d'une matrice carrée d'ordre n , on obtient un réseau comportant au moins $n^2/2$ processeurs et ce , quelle que soit la direction de projection utilisée ; de même pour la résolution d'un système triangulaire d'ordre n , on obtient un réseau comportant au moins n processeurs .

Notre approche dite méthode de partitionnement est basée sur le même schéma général que les méthodes de projection .

Partant d'une solution séquentielle exprimée à l'aide d'équations récurrentes , on procède en trois étapes .

Etape 1 :

Construction d'un diagramme espace-temps ou graphe de dépendance .
On construit un graphe orienté G=(V,U) dans lequel
- chaque sommet v de V est associé à un calcul ,
- il existe un arc e=(u,v) si et seulement si l'un des résultats produits en u est nécessaire au calcul effectué en v .

Etape 2 :

Définition d'une fonction de temps optimale .
Admettons que tous les calculs associés aux sommets de G aient la même durée , et prenons cette durée comme unité de temps . Il est clair que le temps minimum d'exécution de l'algorithme est 1+d où d est la longueur du plus long chemin élémentaire de G . Il s'agit pour nous de construire une fonction de temps optimale T ; c'est à dire associer à chaque sommet v une date T(v) de réalisation de ces calculs de sorte que :

$$\text{Max } \{T(v) : v \in V \} = 1+d .$$

Pour ce faire on procède comme suit :
- cas 1 : considérer les sommets v qui sont extrémités d'un plus long chemin élémentaire de G et leur affecter la date T(v) = 1+d .
- cas 2 : considérer tous les sommets u dont les voisins en sortie ont déjà une date d'exécution et leur affecter la date

$$T(u) = -1+\text{Min}\{T(v) : (u,v) \text{ est un arc de } G \} .$$

- cas 3 : répéter le cas 2 tant qu'il y a des sommets dont la date d'exécution n'est pas encore fixée .

Etape 3 :

Définition de la fonction d'allocation .
Il est clair que pour une fonction de temps T le nombre minimum de processeurs nécessaires pour effectuer en parallèle les calculs de date d'exécution t est

$$\text{Card}\ (\ \{x : T(x) = t\}\ )\ .$$

En conséquence , la taille minimum d'un réseau systolique permettant d'exécuter en temps minimum , le schéma séquentiel de départ , est

$$m = \underset{T}{\text{Min}}\ \underset{t}{\text{Max}}\ \text{Card}\ (\ \{x : T(x) = t\}\ )\ .$$

Comme son nom l'indique , la phase d'allocation consiste à allouer les calculs aux processeurs , ce qui revient à construire une partition des sommets de G en k classes $C_1 , C_2 , \ldots , C_k , k \geq m$.
Une manière de formaliser le problème est de colorier les sommets de G en k couleurs . Les sommets de même couleur sont alors affectés au même processeur , et ne peuvent pas être effectués simultanément ; en conséquence , ils doivent avoir des dates de calcul différentes . Par ailleurs le réseau d'interconnexion du réseau systolique ainsi obtenu se déduit de G en contractant en un seul point tous les sommets de même couleur ; il faut donc que cette contraction donne un graphe planaire régulier et localement connecté .

REMARQUES :

1 - La méthode de partitionnement se distingue des méthodes de projection par le fait que , par construction , elle conduit toujours à une solution optimale en temps d'exécution . Rappelons que cette optimalité se réfère non pas à tous les algorithmes systoliques possibles pour la résolution du problème donné , mais à toutes les réalisations systoliques du schéma séquentiel de départ .
2 - Il n'y a pas d'unicité de la fonction de temps optimale ; mais la stratégie de définition de la fonction que nous avons présentée permet d'éviter les dispositifs de mémorisation que d'autres stratégies induiraient .
3 - Dans la plupart des cas , le graphe G obtenu à l'étape de définition du graphe de dépendance , peut être considéré comme un oignon . Une stratégie d'allocation peut alors consister à décomposer cet oignon en pelures . Si les pelures obtenues vérifient les conditions requises pour les classes C_i on s'arrête ; dans le cas contraire , on considère à leur tour ces pelures comme des oignons entiers qu'on décompose à leur tour en pelures . Cette description imagée s'éclaircira ultérieurement , à la lumière des exemples traités .
Notons enfin que , contrairement aux méthodes de projection , la méthode de

partitionnement est applicable même lorsque le graphe G n'est pas immergé dans un espace euclidien $\mathbb{Z}^n$.

2 - Résolution d'un système triangulaire

Considérons un des algorithmes séquentiels de résolution de ce problème :

```
BEGIN
    FOR i := 1 TO n DO x [i] := b [i] ;      (* initialisation *)
   x [1] := x [1] / a [1,1] ;
   FOR i := 2 TO n DO
      BEGIN
          FOR j := 1 TO i-1 DO x [i] := x [i] - a [i,j]*x [j] ;
         x [i] := x [i] / a [i,i] ;
      END
END ;
```

En adjoignant à la variable x [i] un second indice pour distinguer les calculs successifs auxquels elle participe , on obtient une nouvelle formulation de l'algorithme qui consiste en :
- une phase d'initialisation :

$$x[i,0] = b[i] \qquad 1 \leq i \leq n$$

- une phase de calculs :

$$\left.\begin{array}{l} x[i,j] = x[i,j-1] - a[i,j] * x[j] \\ x[i] = x[i,i] = x[i,i-1] / a[i,i] \end{array}\right\} \quad 2 \leq i \leq n \text{ et } 1 \leq j \leq i-1 \ .$$

Ces équations récurrentes définissent un graphe G dont chacun des sommets ou points de calculs , correspond à une étape du déroulement de l'algorithme .
En examinant les dépendances des points de calcul , on observe que seules les variables x [i,j] circulent tandis que les variables a [i,j] restent figées . Aussi un déroulement correct de l'algorithme suppose que les variables a [i,j] sont lues dans la troisième direction (verticale) k .
Concrètement cela correspond par exemple pour n = 5 au diagramme de la figure 2.1 où un rond en position (i,i) correspond au calcul x [i,i] = b [i,i-1] / a [i,i] tandis que un carré en position (i,j) , i > j , correspond au calcul x[i,j]=x[i,j-1] - a[i,j]*x[j,j].

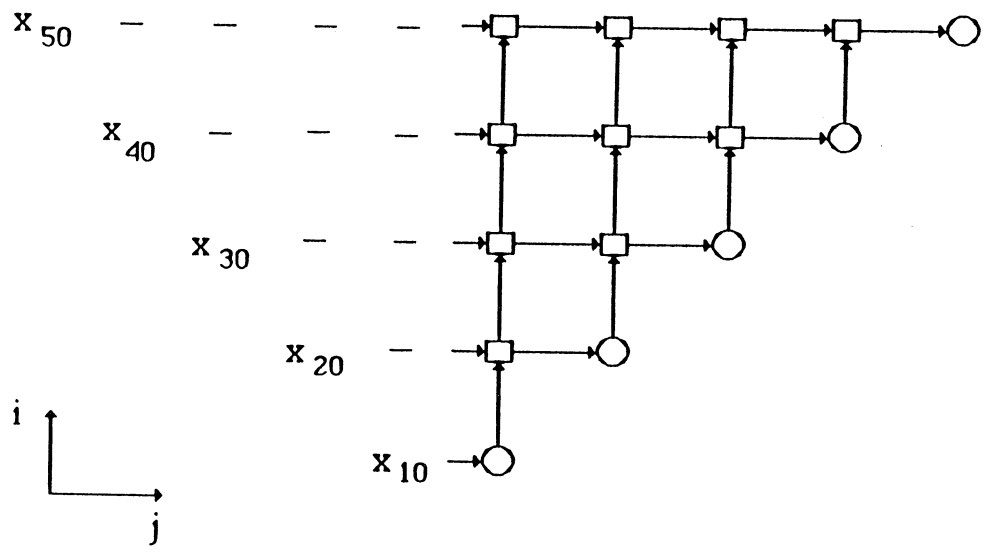
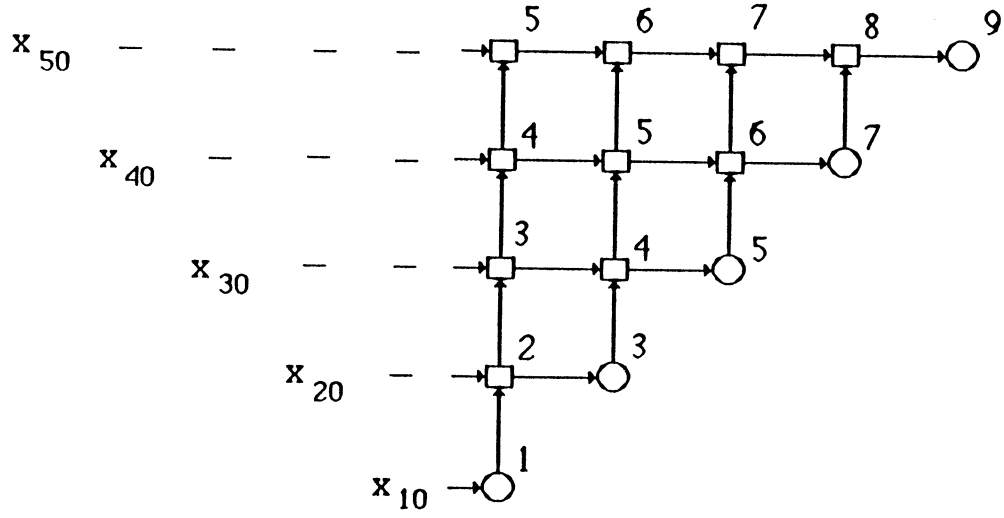


figure 2 .1

On vérifie que la fonction de temps optimale associée au diagramme est unique et consiste à réaliser les calculs de (i,j) à la date t = i+j-1 ; d'où le diagramme espace-temps :



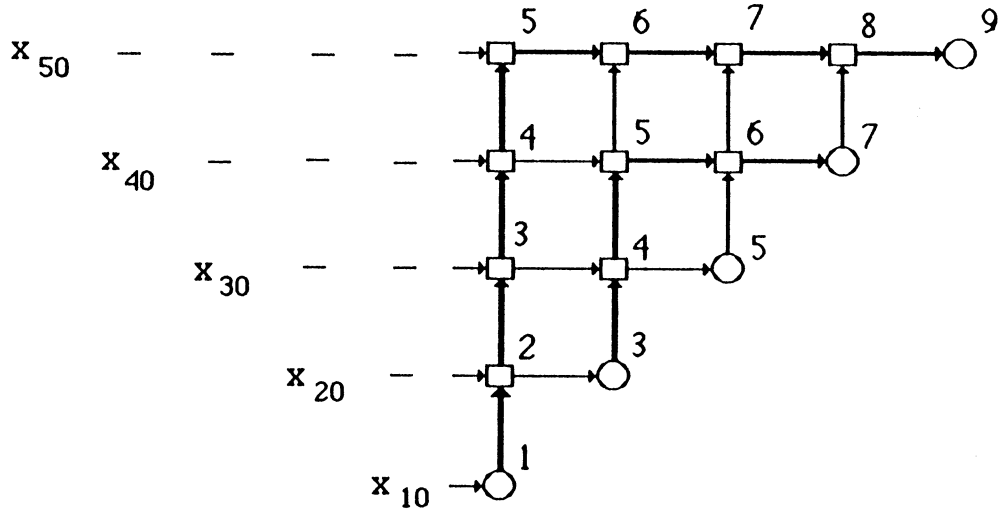
A présent on est à même de déterminer le plus petit nombre de processeurs nécessaires à la réalisation du nouvel algorithme . Il s'agit comme annoncé précédemment de dénombrer le maximum de points ayant la même date de calcul . Il suffit pour cela de résoudre le système d'inéquations :

$$\begin{cases} 1 \le j \le n \\ j \le i \le n \\ i+j-1 = t \,. \end{cases}$$

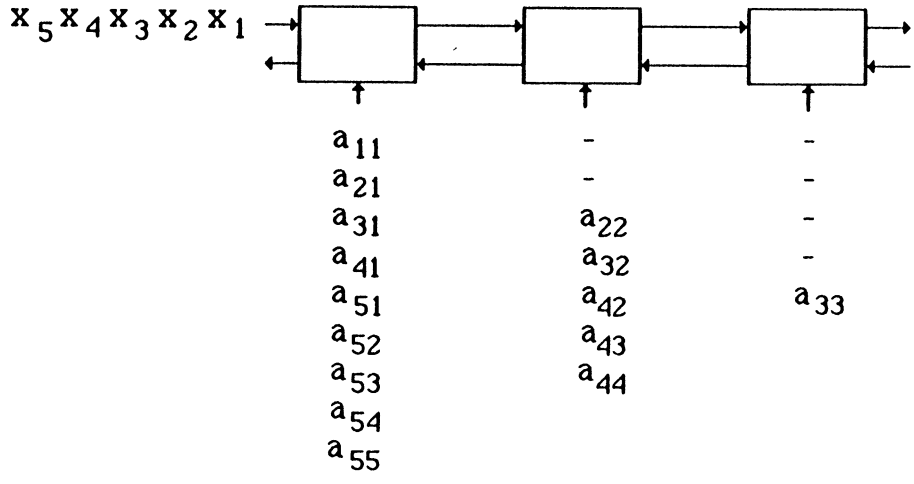
On vérifie que ce maximum est ⌈n / 2⌉ qui correspond au temps t = n .

L'étape suivante consiste à chercher une répartition des points de calcul entre ⌈n / 2⌉ processeurs de sorte que :
- deux points de même date de calcul ne soient pas affectés au même processeur ,
- l'ordre des calculs soit respecté ,
- les processeurs aient des comportements simples , et le réseau régulier .

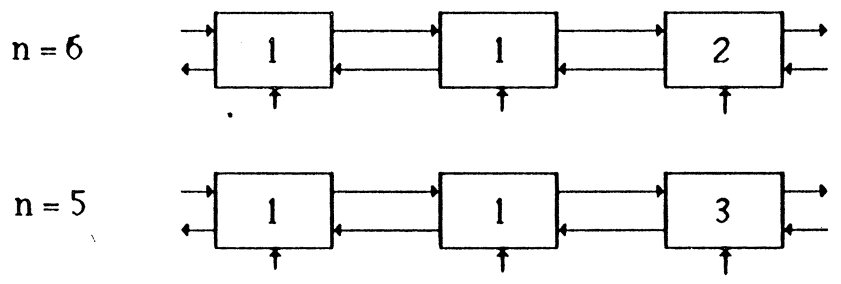
Parmi les solutions possibles , nous considérons celle qui consiste à affecter tout point (i,j) au processeur de rang min(n-i+1,j) . Dans notre exemple cela correspond au partitionnement ci-après indiqué en gras .



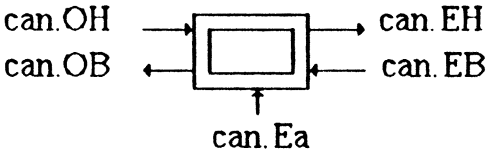
Le réseau induit par ce partitionnement a la structure suivante :



En général comme l'indiquent les exemples ci - après , selon que n est pair ou impair , on distingue les processeurs 1 et 2 ou 1 et 3 .



Chaque processeur dispose de trois canaux de communication et d'un registre interne.



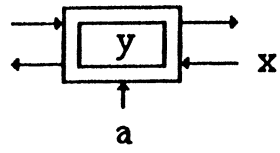
Le comportement d'un processeur du type 1 en position i dans le réseau comprend cinq phases :
- La première comporte une étape qui correspond au premier calcul de la cellule . Elle débute à la date 2(i-1) et consiste à lire les valeurs des variables x et a respectivement sur can.OH , can.Ea pour effectuer le calcul x' = x / a . La valeur de la variable x' est stockée dans le registre .

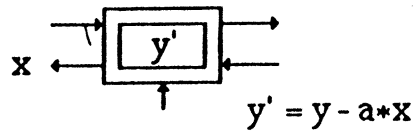
- La deuxième phase comporte n-2i étapes dont chacune consiste à lire les valeurs des variables a et y respectivement sur can.Ea et can.OH ; après avoir effectué la transformation y' = y - a*x , la valeur de la variable y' est envoyée en sortie sur can.EH tandis que x reste stockée dans le registre interne .

- La troisième phase débute à la date n-1 et comporte une étape dont le schéma est le suivant :

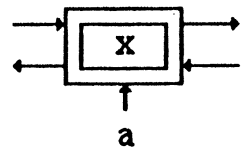
Les valeurs des variables a et y sont lues sur can.Ea et can.OH . Après la transformation y' = y - a*x , tandis que y' est stocké dans le registre , x est envoyé en sortie sur can.OB .

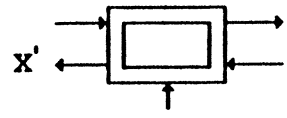
- La quatrième phase comporte n-2i étapes dont chacune correspond aux schémas suivants :





- La cinquième correspond à la date 2n-2i et comporte une étape qui consiste à lire a sur can.Ea , calculer x' = x / a et à sortir x' sur can.OB .



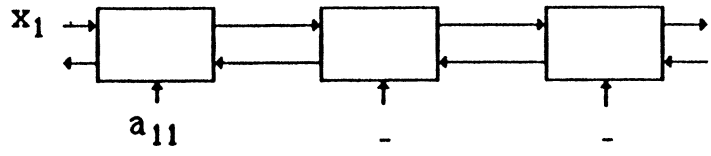


Le programme d'une cellule de type 2 se déroule en trois phases correspondant , dans cet ordre , à la première , la troisième et la cinquième phase d'une cellule du type 1 .

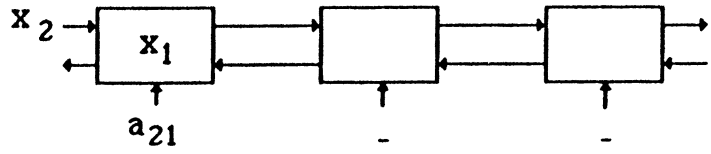
Quant au programme d'une cellule du type 3 il comprend une seule phase d'une étape : la cellule lit sur can.OH et can.Ea respectivement les valeurs des variables x et a ; après avoir effectué le calcul x := x / a la valeur de la variable x est envoyée en sortie sur can .OB .

Pour n = 5 le fonctionnement général du réseau est le suivant où les x_i sont initialisés à b_i :

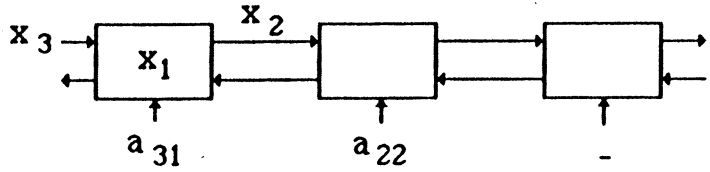
t = 0 :



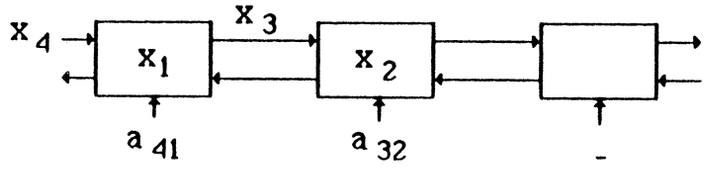
t = 1 :



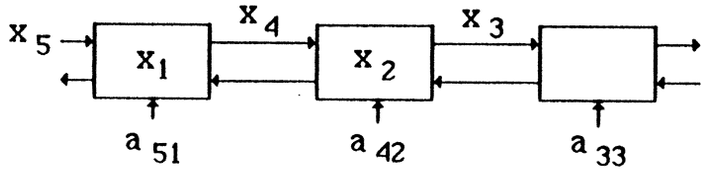
t = 2 :



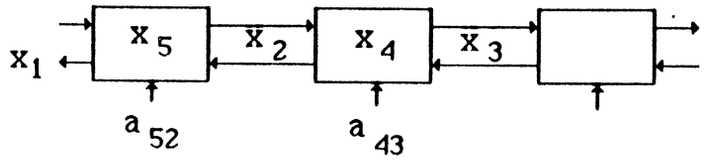
t = 3 :



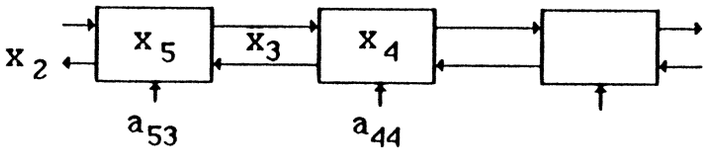
t = 4 :



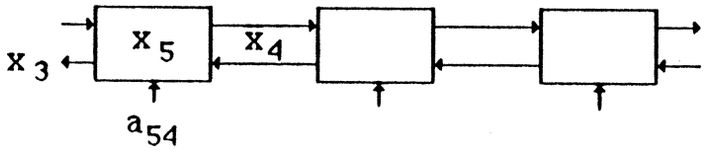
t = 5 :



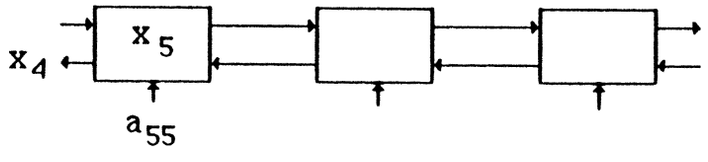
t = 6 :



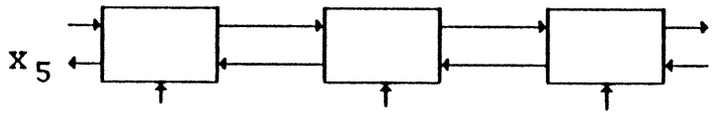
t = 7 :



t = 8 :



t = 9 :



En réalité on n'a pas besoin d'avoir trois types de cellules dans le réseau ; il suffit d'avoir des cellules programmables et d'adjoindre aux éléments de la matrice Λ des signaux de contrôle qui assureront un comportement correct pour chaque processeur .

- La première phase correspond au premier calcul d'une cellule ; elle se repère naturellement .
- Comme la première phase ne comporte qu'une seule étape , le passage à la deuxième phase se fait naturellement .
- La troisième phase correspond à la date à laquelle le processeur lit sur can.S , la

valeur de la variable a [i,j] telle que i+j = n+1 . On peut repérer cette date par exemple en affectant aux variables a [i,j] telles que $i+j \leq n+1$ le signal s [i,j] qui vaut 0 si $i+j < n+1$ et 1 sinon .
- Comme la troisième phase ne comporte qu'une seule étape , le passage à la quatrième se fait naturellement .
- La cinquième phase correspond à l'arrivée sur can.S d'une variable a [i,i] avec $2i > n+1$; on peut repérer cet instant en affectant aux variables , a [i,j] telles que $i+j > n+1$, le signal s [i,j] qui vaut 1 si i = j et 0 sinon .
Ainsi une cellule du type 3 dont tout le programme correspond à son premier calcul est contrôlé par l'arrivée sur can.S d'un signal qui vaut 1; en effet c'est le seul processeur dont le premier signal de contrôle vaut 1 .

En résumé il est possible , moyennant un bit de contrôle , de résoudre en temps $T = 2n - 1$un système triangulaire d'ordre n sur un réseau de taille $\lceil n / 2 \rceil$. Ceci constitue une nette amélioration par rapport aux réseaux obtenus par projection et qui s'exécutent en temps T = 2n -1 sur un réseau linéaire de taille n .

3 - Inversion d'une matrice triangulaire

Soit $U = (u_{ij})$ une matrice triangulaire inférieure d'ordre n . Nous nous intéressons au calcul de son inverse $V = U^{-1}$.

3.1 - Principe de la méthode

Partant d'une matrice $A^{(0)} = (U . I)$ d'ordre n x 2n où I désigne la matrice unité , on effectue des combinaisons de lignes de manière à obtenir la matrice (I , U^{-1}) de même ordre .
Supposons que partant de la matrice $A^{(0)}$ les i premières lignes soient déjà traitées ; la matrice $A^{(i-1)}$ ainsi obtenue est de la forme suivante :

$$\begin{bmatrix}
1 & & & & & v_{11} & & & \\
0 & 1 & & & & v_{21} & v_{22} & & \\
\vdots & & \ddots & & & \vdots & & \ddots & \\
0 & \cdots & 0 & 1 & & v_{i-11} & \cdots & v_{i-1i-1} & \\
u_{i1} & \cdots & u_{ii-1} & u_{ii} & & 0 & \cdots & 0 & 1 \\
\vdots & & & & & \vdots & & & \\
u_{n1} & \cdots & & u_{nn} & & 0 & \cdots & 0 & 1
\end{bmatrix}$$

Considérons la $i^{ème}$ ligne ; notre but est de lui ajouter une combinaison des i-1 premières lignes de manière à obtenir une matrice $A^{(i)}$ dans laquelle

$$u_{i1} = u_{i2} = . \ . \ . = u_{ii-1} = 0 \text{ et } u_{ii} = 1 .$$

En notant $A^{(i)}(k,.)$ la $k^{ème}$ ligne de $A^{(i)}$, on voit qu'il suffit d'effectuer d'abord la transformation :

$$A^{(i)}(i,.) := A^{(i-1)}(i,.) - \sum_{j=1}^{i-1} u_{ij} A^{(i-1)}(j,.)$$

puis la normalisation

$$A^{(i)}(i,.) := (1/u_{ii}) A^{(i)}(i,.) .$$

En conclusion , l'algorithme peut se formaliser de la manière suivante :

- Initialisation :

```
FOR i := 1 TO n DO w [i,i] := 1 ;
FOR i := 2 TO n DO
    FOR j := 1 TO i-1 DO w [i,j] := 0 ;
```

- Calculs :

```
v [1,1] := 1/u [1,1] ;
FOR i := 2 TO n DO
   BEGIN
      v [i,i] := 1/u [i,i] ;
      FOR j := i-1 DOWNTO 1 DO
         BEGIN
            FOR k := j TO i-1 DO w [i,j] := w [i,j]-u [i,k]*v [k,j] ;
            v [i,j] := w [i,j]/u [i,i] ;
         END
   END ;
```

En adjoignant à chaque variable un troisième indice pour distinguer les calculs successifs auxquels elle participe , on obtient la nouvelle formulation de l'algorithme sous forme d'équations récurrentes :

- Initialisation :

$$w_{ij}^{(j-1)} = 0 \qquad \text{pour } i = 2 , \ldots , n \text{ et } j = 1 , \ldots , i-1$$

$w_{ii}^{(i-1)} = 1$ pour i = 1 , . . . , n

$u_{ij}^{(j+1)} = u_{ij}$ pour i = 1 , . . . , n et j = 1 , . . . , i

Calculs :

```
BEGIN
    w11(1) := w11(0) ; u11(1) := u11(2) ; v11(1) := w11(1) / uii(1) ;
    FOR i := 2 TO n DO
      BEGIN
          uii(i) := uii(i+1) ; wii(i) := wii(i-1) ; vii(i) := wii(i) / uii(i) ;
        FOR j := i-1 DOWNTO 1 DO
          BEGIN
            FOR k := j TO i-1 DO
              BEGIN
                  uik(j) := uik(j+1) ; vkj(i) := vkj(i-1) ;
                  wij(k) := wij(k-1) - uik(j) * vkj(i)
              END ;
            uii(j) := uii(j+1) ; vij(i) := wij(i-1) / uii(j)
          END
      END
END ;
```

Le domaine de calculs correspondant à ces équations récurrentes est

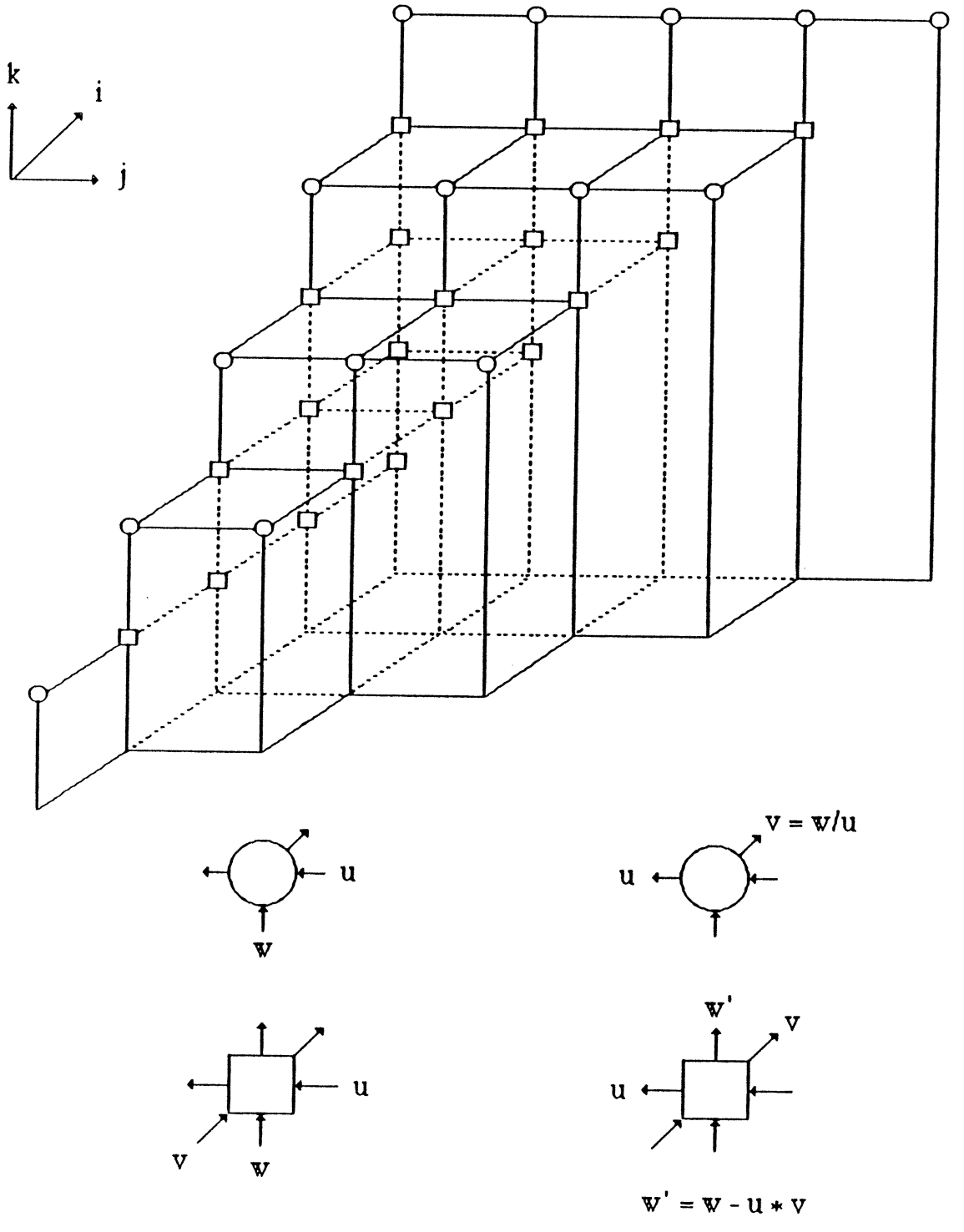
$$\mathbf{D} = \{ (i,j,k) \in \mathbb{N}^3 : i \leq j \leq k \leq n \}$$

dont nous donnons une représentation à la page 26 .
Ce diagramme s'interprète comme suit :
- En un point représenté par un rond a lieu le calcul $v_{ij}^{(i)} := w_{ij}^{(i-1)} / u_{ii}^{(j)}$.
Un tel point correspond à une cellule qui lit les valeurs des variables w et u respectivement dans les directions (0,0,1) et (0,-1,0) . Après son calcul ,v est envoyé au point suivant dans la direction (1,0,0) tandis que u est transmis inchangé dans la direction (0,-1,0) .
- En un point représenté par un carré a lieu le calcul suivant :
La variable w circule dans la direction (0,0,1) et en chacun des points visités elle subit la transformation w := w - u *v . Les valeurs des variables u , v restent inchangées et circulent respectivement dans les directions (0,-1,0) et (1,0,0) .

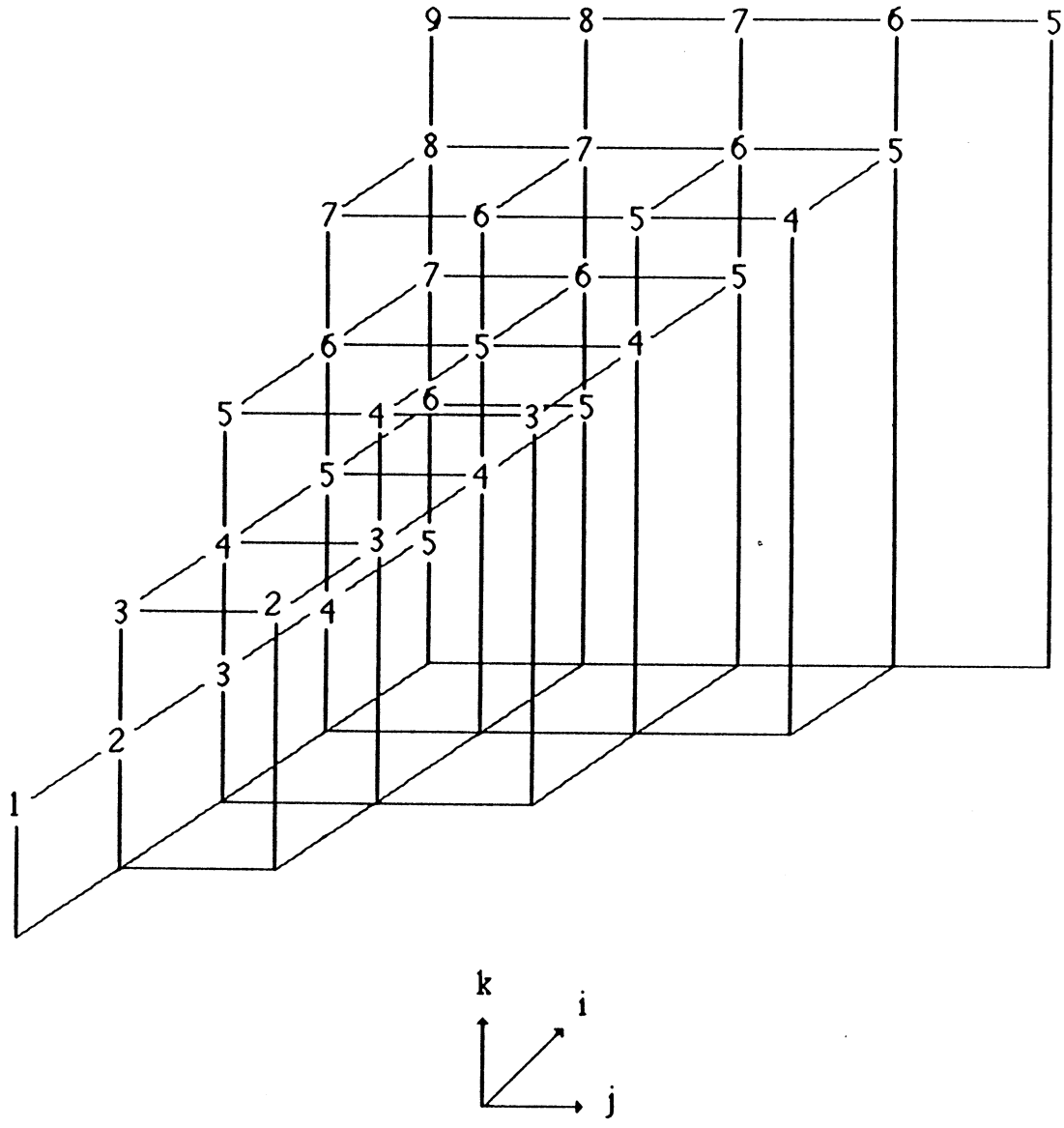
3.2 - Date de calcul :

L'application de l'algorithme de construction d'une fonction de temps optimale

Diagramme des calculs pour l'inversion d'une matrice triangulaire d'ordre 5 .



Les variables u , v , w circulent respectivement dans les directions (0 , -1 , 0) , (1 , 0 , 0) , (0 , 0 , 1) .



$$T(i, j, k) = i - j + k$$

Diagramme espace-temps de l'algorithme systolique d'inversion d'une matrice triangulaire inférieure d'ordre $n = 5$.

exposé précédemment donne pour tout point (i , j , k)

$$T(i , j , k) = i + k - j .$$

Le temps minimum nécessaire pour mettre en oeuvre le schéma de départ est donc

$$\max (T(i , j , k) : 1 \leq j \leq i \leq k) = 2n - 1 .$$

La figure de la page 27 donne le diagramme espace-temps pour n = 5 .
Nous nous intéressons à la classe des fonctions optimales qui n'induisent pas de mécanisme de mémorisation entre points de calculs .

3.3 - Réseau minimal d'exécution :

Considérons le sous-ensemble $\mathbf{D}_k = \{(x,y,z) \in \mathbf{D} : z = k\}$. Désignons par $m_k(t)$ le nombre de points de $\mathbf{D}_k$ qui sont simultanément calculés à la date t .

Lemme :

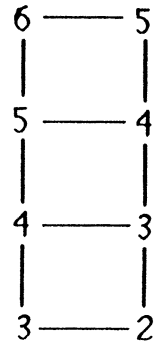
Pour tout t appartenant à l'intervalle [k , n+k-1] la suite $(m_k(t))$ est de la forme :

$$\underbrace{1 , 2 , 3 , \ldots , m-1 , m , \ldots , m}_{r} , m-1 , \ldots , 2 , 1$$

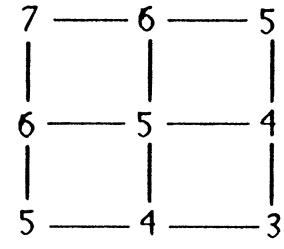
où m = min (k , n-k+1) et r = n-2m+2 .

Démonstration :

$\mathbf{D}_k$ est un rectangle dont les côtés ont pour longueurs k , dans la direction j et n-k+1 , dans la direction i . Par exemple pour n = 5 on a :



a) $\mathbf{D}_2$



b) $\mathbf{D}_3$

Par ailleurs les points ayant la même date sont situés sur les droites d'équations

$$i-j = t-k$$

et le premier point à être calculé est un sommet du rectangle .
On voit alors que :
- si $k \leq n-k+1$ (voir exemple (a)) alors les bissectrices sont de longueurs

$$1 , 2 , 3 , \ldots , k-1 , \underbrace{k , \ldots , k}_{r} , k-1 , \ldots , 2 , 1$$

où $r = n-2k+2$.
- si $n-k+1 \leq k$ (voir exemple (b)) alors les bissectrices sont de longueurs

$$1 , 2 , 3 , \ldots , m-1 , \underbrace{m , \ldots , m}_{r} , m-1 , \ldots , 2 , 1$$

où $m = n-k+1$ et $r = n-2m+1$.

Proposition 1 :

Lorsqu'on part du système d'équations récurrentes que nous avons introduit précédemment , le plus petit nombre de processeurs nécessaires à l'exécution d'un algorithme systolique d'inversion d'une matrice triangulaire est :

$$m = \begin{cases} (n+1)^2/4 & \text{si n est impair} \\ n(n+2)/4 & \text{sinon .} \end{cases}$$

La démonstration est immédiate . En effet , en vertu du lemme , les nombres m(t) de points de **D** calculés à la date t forment une suite palindrome dont le terme général pour tout $t \in [1 , n]$ est de la forme :

$$m(t) = \sum_{k=1}^{n} m_k(t) = \begin{cases} \sum_{r=1}^{t/2} 2r \cdot & \text{si t est pair} \\ \sum_{r=1}^{(t-1)/2} 2r + (t+1)/2 & \text{sinon .} \end{cases}$$

Evidemment cette fonction atteint son maximum pour $t = n$; d'où la proposition .

Exemple : Pour $n = 5$ le tableau des valeurs de la double suite $(m_k(t))$ est

t :	1	2	3	4	5	6	7	8	9
$m_1(t)$:	1	1	1	1	1				
$m_2(t)$:		1	2	2	2	1			
$m_3(t)$:			1	2	3	2	1		
$m_4(t)$:				1	2	2	2	1	
$m_5(t)$:					1	1	1	1	1
m(t) :	1	2	4	6	9	6	4	2	1

Connaissant le nombre optimal de processeurs , il ne nous reste plus qu'à déterminer une fonction d'allocation optimale et compatible avec la fonction de temps .
Pour ce faire , remarquons que le diagramme **D** n'est rien d'autre que l'interprétation de l'équivalence entre l'inversion d'une matrice triangulaire U et la résolution de n systèmes linéaires :

$$Ux = e_i \qquad i = 1, 2, \ldots, n$$

où e_i désigne le $i^{ème}$ vecteur de la base canonique de $\mathbb{R}^n$. La solution de $Ux = e_i$ est alors la $i^{ème}$ colonne de U^{-1} .
En effet , d'après la description des points de calcul de **D** , U n'est pas modifié au cours de l'inversion ; mais alors en considérant les tranches de **D** obtenues pour j = n , j = n-1 , . . . , j = 1 , on retrouve le diagramme espace-temps de la résolution de systèmes triangulaires inférieures d'ordre n-j+1 ci-dessous .

$$\begin{bmatrix} u_{jj} & & & \\ \cdot & \cdot & O & \\ \cdot & \cdot & \cdot & \\ \cdot & & \cdot & \\ u_{nj} & \cdot & \cdot & u_{nn} \end{bmatrix} \begin{bmatrix} y_j \\ \cdot \\ \cdot \\ \cdot \\ y_n \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix}$$

Considérons alors la fonction d'allocation **A** introduite pour la résolution d'un système triangulaire . A tout point (i,j,k) du domaine de calculs de l'inversion d'une matrice triangulaire elle associe un processeur en position (x,y) dans $\mathbb{N}^2$ tel que
y = j et x = min(n-i , k-j) +1 .

Proposition 2 : **A** est optimal et est compatible avec la fonction de temps .

Démonstration :

Commençons par montrer que **A** est compatible avec la fonction de temps .
Soient (i_1,j_1,k_1) et (i_2,j_2,k_2) deux points distincts de **D** calculés à la même date et appartenant au même sous-domaine D_k . Alors $k_1 = k_2$ et $(i_1 , j_1) \neq (i_2 , j_2)$.
- si $j_1 \neq j_2$ alors $y_1 \neq y_2$ et par suite les deux points ne sont pas affectés au même processeur ;
- si $j_1 = j_2$, les deux points ayant la même date , alors $i_1 = i_2$; ce qui contredit notre hypothèse .

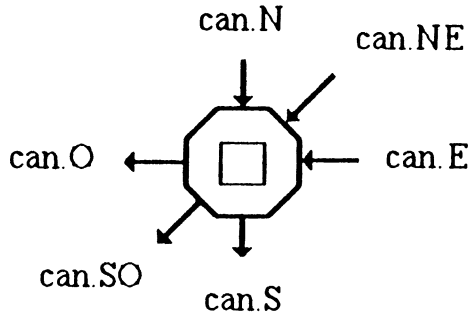
Montrons à présent que **A** est optimale :

Par construction , pour chaque tranche de **D** obtenue pour j fixé , **A** induit un réseau linéaire de taille $\lceil m_j \rceil$ où $m_j = (n-j+1) / 2$. Pour **D** tout entier , le nombre total de processeurs du réseau induit est :

$$m = \sum_{j=1}^{n} m_j = \begin{cases} n(n+2)/4 & \text{si n est pair} \\ (n+1)^2/4 & \text{sinon .} \end{cases}$$

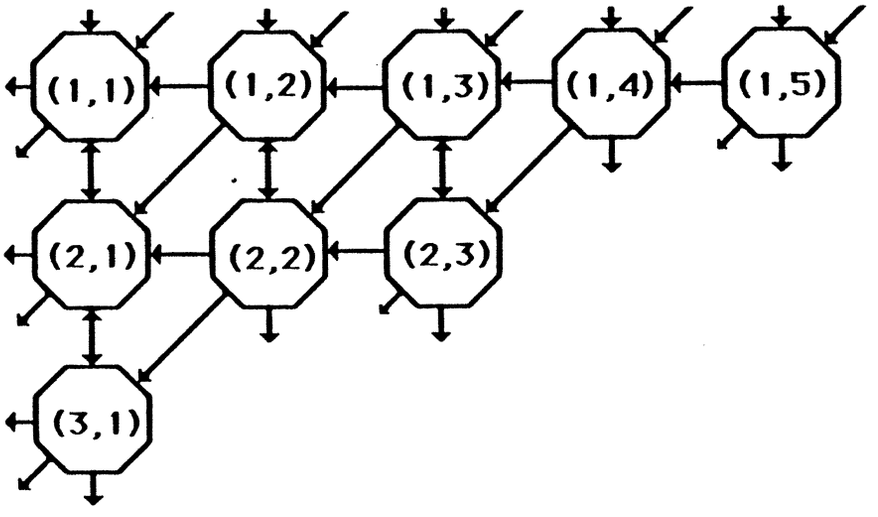
3.4 - Fonctionnement du réseau :

Comme l'indique sa structure générale (voir page suivante) , le réseau est composé d'un ensemble de processeurs disposant en général chacun de six canaux de communication avec son voisinage et d'un registre interne .

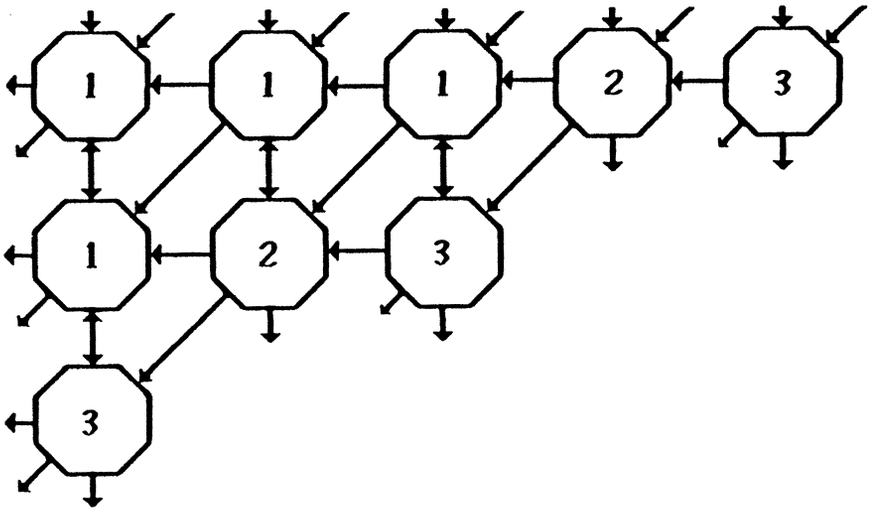


Par rapport à leur programme , ces processeurs peuvent être classés en trois groupes que nous dirons être du type 1 , du type 2 ou du type 3 (voir page suivante). Nous distinguons dans le comportement d'un processeur de type 1 en position (i,j) dans le réseau cinq phases dont chacune se compose d'étapes de calculs identiques .

w_{51}	u_{51}	w_{52}	u_{52}	w_{53}	u_{53}	w_{54}	u_{54}	w_{55}	u_{55}
w_{41}	u_{41}	w_{42}	u_{42}	w_{43}	u_{43}	w_{44}	u_{44}	-	-
w_{31}	u_{31}	w_{32}	u_{32}	w_{33}	u_{33}	-	-	-	-
w_{21}	u_{21}	w_{22}	u_{22}	-	-	-	-	-	-
w_{11}	u_{11}	-	-	-	-	-	-	-	-

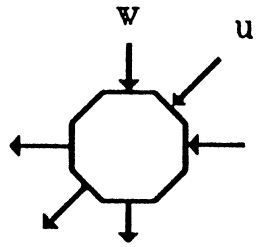


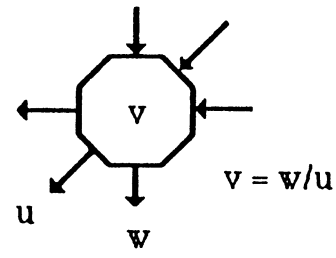
Réseau induit par A pour n = 5 .



Répartition des processeurs selon leur programme .

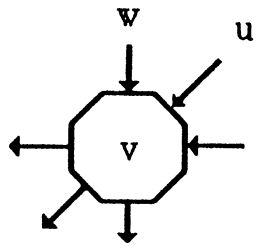
- La première phase comporte une seule étape qui correspond aux schémas suivants :

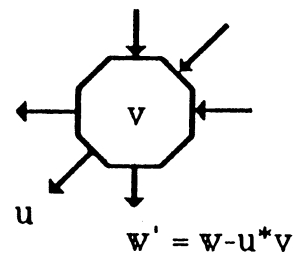




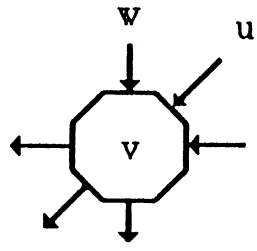
La cellule reçoit w sur can.N , u sur can.NE , calcule v = w/u , le stocke dans son registre interne puis envoie w en sortie sur can.S et u sur can.SO ; cette phase correspond au temps t = 2(i-1)+j .

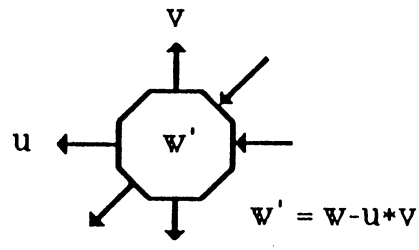
- La deuxième phase va de la date t = 2(i-1)+j à la date t = n-1 . Pendant cette phase , la cellule reçoit en entrée w sur can.N et u sur can.NE , effectue la transformation w:= w-u*v , envoie la nouvelle valeur de w en sortie sur can.S et la valeur inchangée de u sur can.SO .

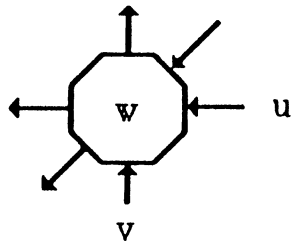


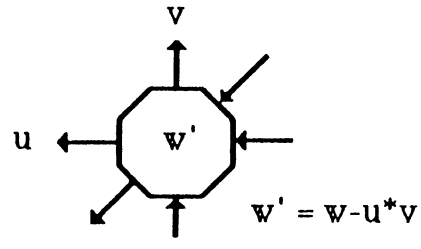


- La troisième phase comporte uniquement l'étape à la date t = n qui correspond aux deux schémas suivants :



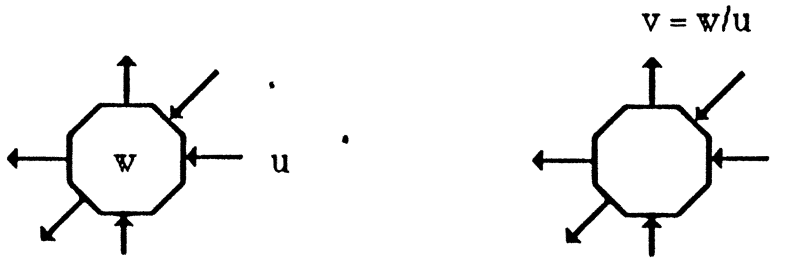






- Chaque étape de la quatrième phase qui commence à la date t = n+1 correspond aux deux derniers schémas de la page 29 :
w est stocké dans le registre interne de la cellule et subit la transformation w:=w-u*v à la réception des variables u et v .

- La cinquième phase est le dernier calcul de la cellule et correspond à la division de w par u .

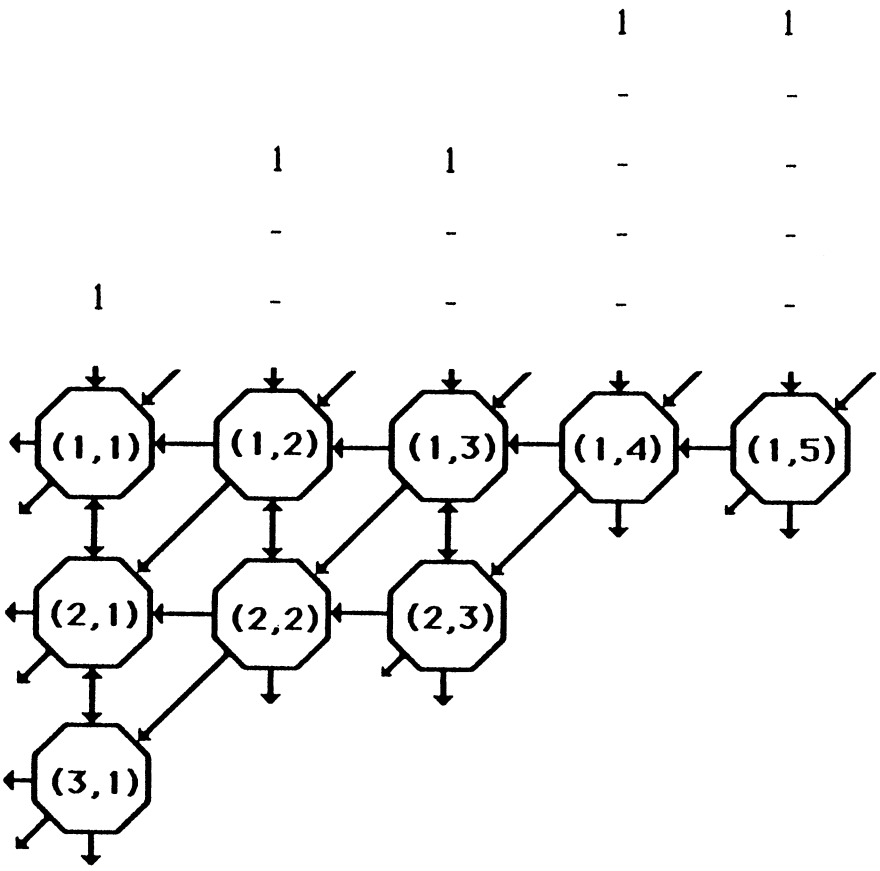


Le comportement d'une cellule du type 2 correspond dans cet ordre à la première , la troisième et la cinquième phases d'une cellule du type 1 .
Le programme d'une cellule du type 3 comporte une seule phase d'une étape . La cellule reçoit sur les canaux can.N et can.NE respectivement les valeurs des variables w et u ; après le calcul v = w / u , les valeurs des variables v et u respectivement sont envoyées en sortie sur can.N et can.O .

A l'issue de cette description , la question qui se pose est de savoir comment une cellule peut déterminer la date à laquelle elle doit passer d'une phase à une autre . Nous allons introduire des signaux de contrôle pour résoudre ce problème .

- La première phase correspond au premier calcul d'une cellule . Ce calcul intervient lorsqu'elle reçoit pour la première fois un signal u_{ij} tel que i = j . Il suffit donc d'adjoindre à chaque variable u_{ij} un signal booléen qui vaut 1 si i = j et 0 sinon.
- Comme la première phase ne comporte qu'une étape , la deuxième est facile à repérer ; il suffit par exemple de prévoir dans la cellule un registre booléen initialisé à 0 et qui passe à 1 après le premier calcul .
- Comme on le constate sur le réseau pour n = 5 , la troisième phase comporte pour toutes les cellules l'unique calcul correspondant à la date t = n . Il suffit donc , pour repérer cette phase , de mettre en oeuvre un mécanisme permettant à toutes les cellules de se synchroniser à la date t = n . Si on décide de résoudre ce problème en parallèle et de manière indépendante sur les colonnes du réseau , on retrouve le célèbre problème de la synchronisation d'une ligne de fusiliers connu sous le nom de Firing Squad Problem . On sait depuis WAKSMAN [4] que le délai minimum pour résoudre ce problème sur une ligne de taille q est 2q-1 , et qu'on peut y arriver avec des signaux à huit valeurs possibles . Récemment , MAZOYER [13] a montré qu'on peut y arriver avec des signaux à six valeurs possibles . Pour obtenir une synchronisation de toutes les colonnes à la date t = n , il suffit de retarder convenablement les signaux de démarrage du Firing Squad comme l'illustre la

figure ci-dessous .



- La troisième phase ne comportant qu'une seule étape , la quatrième est facile à repérer .
- La cinquième phase correspond à la réception par une cellule d'une variable u_{ii} ; elle peut donc se repérer exactement comme la phase 1 grâce à une variable booléenne attachée à chaque coefficient de U et qui vaut 1 si c'est un élément diagonal et 0 sinon .

Nous avons montré que la méthode de partitionnement permet d'inverser une matrice triangulaire en temps 2n-1 sur un réseau de taille $n^2/4 + O(n)$. Ceci constitue une amélioration notable par rapport aux solutions proposées jusqu'ici . En effet la meilleure des solutions précédemment connues et qui est due à LI et WAH [6] , s'exécute en temps 2n-1 sur un réseau de taille n(n+1)/2 .

4 - Triangularisation d'une matrice :

On se propose de trianguläriser , par la méthode d'élimination de GAUSS , une matrice $A = (a(i,j))$ carrée d'ordre n en vue de la résolution du système linéaire $Ax = b$.

Le principe de la méthode consiste pour chaque colonne $j < n$, à éliminer les $a(i,j)$ pour $i > j$ par une combinaison de la $j^{ème}$ ligne et de la $i^{ème}$ ligne de la matrice (A,b) d'ordre $n \times (n+1)$.

Supposons donc que les k-1 premières colonnes aient été traitées ; la matrice $A^{(k-1)}$ obtenue est de la forme :

$$A^{(k-1)} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k-1} & a_{1k} & \cdots & a_{1n} & a_{1n+1} \\ 0 & a_{22} & & & & & & a_{2n+1} \\ \vdots & \ddots & \ddots & & & & & \vdots \\ & & \ddots & a_{k-1k-1} & & & & \vdots \\ & & & 0 & a_{kk} & & & a \\ & & & \vdots & \ddots & \ddots & & a \\ & & & \vdots & & \ddots & \ddots & \vdots \\ & & & \vdots & & & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & a_{nk} & \cdots & a_{nn} & a_{nn+1} \end{bmatrix}$$

où la $(n+1)^{ème}$ colonne représente le vecteur b . Pour ne pas alourdir les notations nous avons volontairement omis le troisième indice k-1 des $a(i,j)$.

Nous voulons éliminer dans $A^{(k-1)}$ les coefficients $(a(i,k) ; k+1 \leq i \leq n)$. Il suffit pour cela de procéder en deux étapes :

1 - normalisation de la $k^{ème}$ ligne :

$$\text{Pour } j = k \text{ à } n+1 \text{ faire } a_{kj}^{(k)} := a_{kj}^{(k-1)} / a_{kk}^{(k-1)}$$

2 - combinaisons permettant d'éliminer les a_{ik} pour $i = k+1, \ldots, n$:

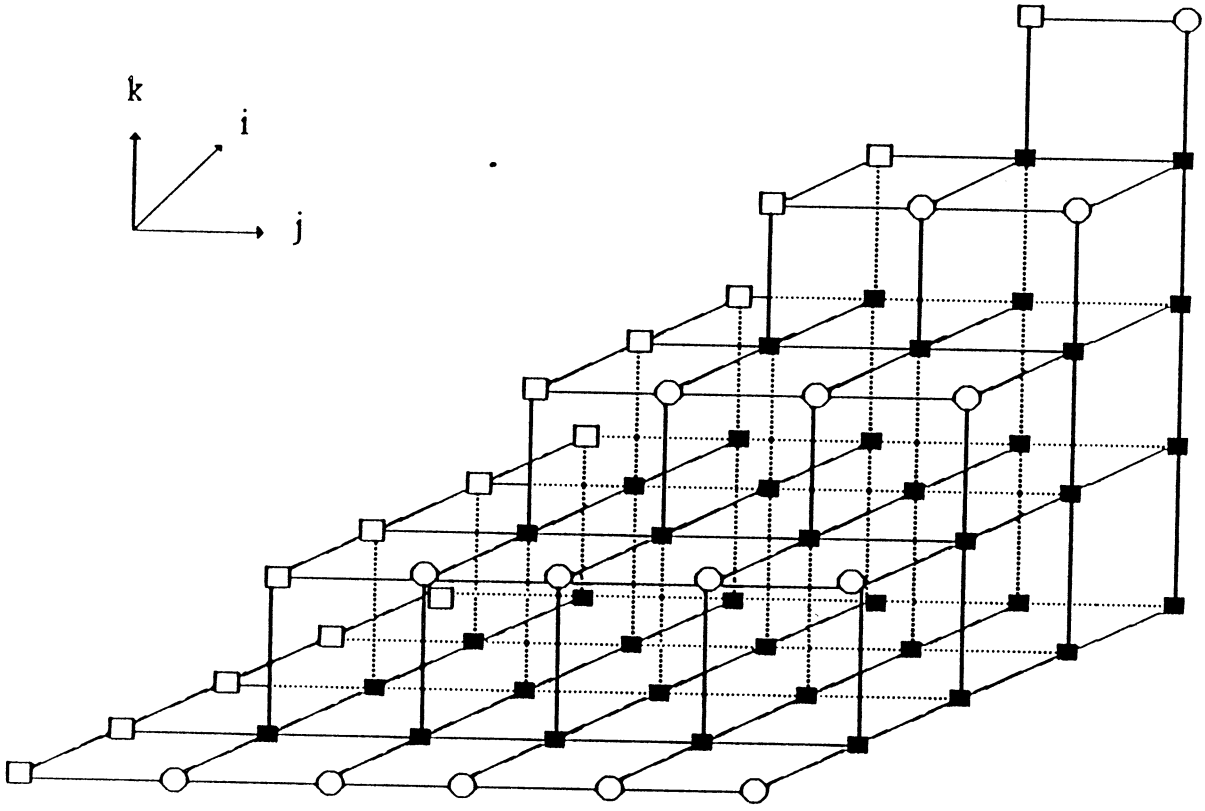
$$\begin{array}{l} \text{Pour } i = k+1 \text{ à } n \text{ faire} \\ \quad \text{Pour } j = k+1 \text{ à } n+1 \text{ faire } a_{ij}^{(k)} := a_{ij}^{(k-1)} - a_{ik}^{(k-1)} * a_{kj}^{(k-1)} . \end{array}$$

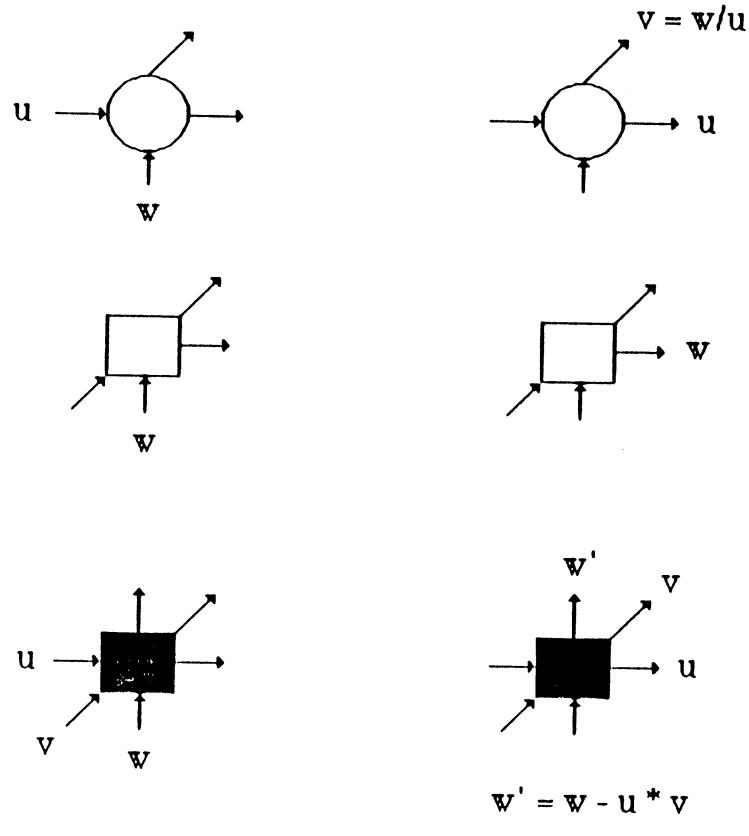
L'algorithme est donc le suivant :

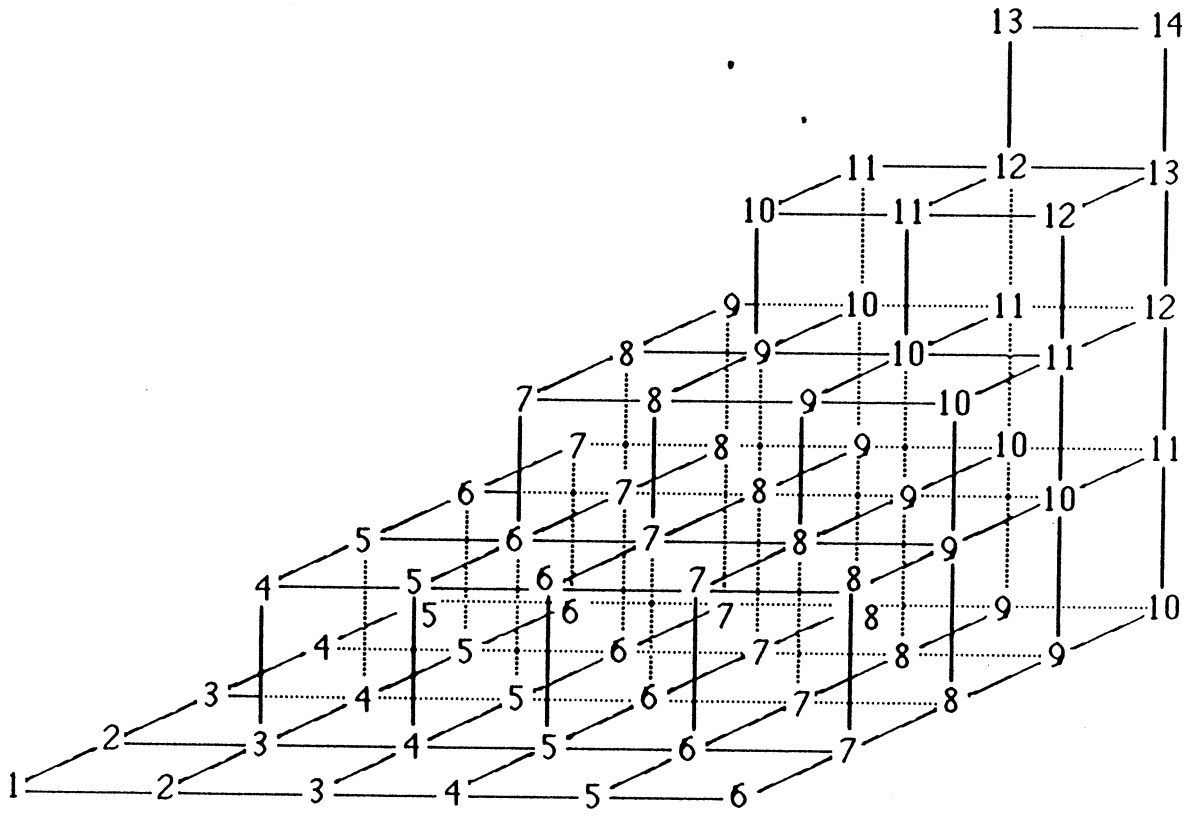
```
BEGIN
      (* initialisation *)
      FOR i := 1 TO n DO
          FOR j := 1 TO n+1 DO a_ij^(0) := a_ij ;
      (* élimination *)
      FOR k := 1 TO n-1 DO
         BEGIN
             (* normalisation de la k^ème ligne *)
             FOR j := k TO n+1 DO a_kj^(k) := a_kj^(k-1) / a_kk^(k-1) ;
             (* élimination de la k^ème colonne *)
             FOR i := k+1 TO n DO
                 (* élimination de a_ik *)
                 FOR j := k+1 TO n+1 DO
                     a_ij^(k) := a_ij^(k-1) - a_ik^(k-1) * a_kj^(k-1) ;
         END ;
      (* normalisation de la n^ème ligne *)
      FOR j := n TO n+1 DO a_nj^(n) := a_nj^(n-1) / a_nn^(n-1)
END;
```

Le domaine de calcul **D** associé à ces équations récurrentes est un sous ensemble de $\mathbb{N}^3$ dont chacun des points correspond à un des calculs de l'algorithme .
La figure de la page 38 donne l' exemple pour n = 5 .
- Le carré en blanc désigne un point de synchronisation . Il reçoit dans la direction (0,0,1) la valeur de la variable $a_{ij}^{(k-1)}$ qu'il retarde d'une unité de temps avant de l'envoyer en sortie dans la direction (0,1,0) .
- Le rond désigne un point de division . Il reçoit les valeurs des variables $a_{kj}^{(k-1)}$ et $a_{kk}^{(k-1)}$ respectivement dans les directions (0,0,1) et (0,1,0) . Après le calcul $a_{kj}^{(k)} := a_{kj}^{(k-1)} / a_{kk}^{(k-1)}$, les valeurs des variables $a_{kj}^{(k)}$ et $a_{kk}^{(k-1)}$ sont envoyées aux points voisins respectivement dans les directions (1,0,0) et (0,1,0) .
- Partout ailleurs un point de calcul reçoit les valeurs des variables $a_{ij}^{(k-1)}$, $a_{ik}^{(k-1)}$ et $a_{kj}^{(k-1)}$ respectivement dans les directions (0,0,1) , (0,1,0) et (1,0,0) . Après la transformation $a_{ij}^{(k)} := a_{ij}^{(k-1)} - a_{ik}^{(k-1)} * a_{kj}^{(k-1)}$, les valeurs des variables $a_{ij}^{(k)}$, $a_{ik}^{(k-1)}$ et $a_{kj}^{(k-1)}$ sont envoyées aux points voisins dans la même direction qu'à l'entrée .

Domaine de calculs de l'algorithme d'élimination de GAUSS pour n = 5







$$T(i, j, k) = i + j + k - 2$$

Diagramme espace-temps de l'algorithme d'élimination de GAUSS pour n = 5 .

4.1 - Date de calcul d'un point et réseau d'exécution de l'algorithme :

Comme le point de coordonnées (1,1,1) est le seul à recevoir de l'extérieur toutes les données qui lui sont nécessaires , la date de calcul au plus tôt d'un point (i,j,k) est le nombre de sommets du plus long chemin d'extrémité (i,j,k) . Clairement , la fonction de temps associe à chaque point de calcul la date t = i+j+k-2 . L'algorithme s'exécute alors en temps 3n-1 . (Voir exemple diagramme espace-temps pour n = 5 à la page 39) . Cette fonction de temps optimale est unique .

A présent déterminons le nombre minimal de processeurs nécessaires à l'exécution de l'algorithme .

Comme dans l'étude précédente , désignons par $\mathbf{D}_k$ le sous-ensemble

$$\{(x,y,z) \in \mathbf{D} : z = k\} .$$

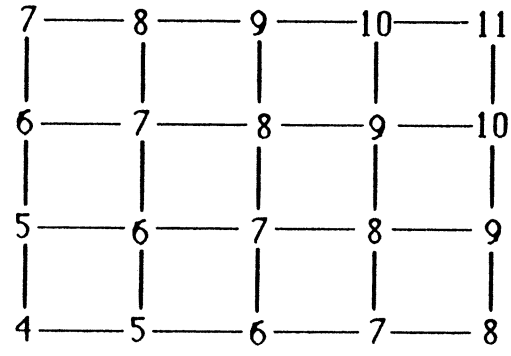
D'après la fonction de temps , les dates des points de $\mathbf{D}_k$ correspondent à l'intervalle $[3k-2 , 2n+k-1]$. Soit $m_k(t)$ le nombre de points de $\mathbf{D}_k$ calculés à la date t .

Lemme :

La séquence $(m_k(t) ; 3k-2 \leq t \leq 2n+k-1)$ est une suite palindrome des n-k+1 premiers entiers naturels :

$$1 , 2 , \ldots , n-k+1 , n-k+1 , \ldots , 1 .$$

En effet $\mathbf{D}_k$ est un rectangle de longueur n-k+2 dans la direction j et de largeur n-k+1 dans la direction i . Nous donnons ci dessous l'exemple de $\mathbf{D_2}$ pour n = 5 .



Des points (i,j,k) de même date t appartiennent à la droite d'équation i+j = t-k+2 ; et comme le premier point calculé est un sommet du rectangle , on voit que la suite des nombres de points par droite est de la forme :

$$1, 2, \ldots, n-k+1, n-k+1, \ldots, 1 .$$

Rappelons que le nombre minimal de processeurs pour l'exécution de l'algorithme est le maximum de points de même date . Aussi s'agit-il pour nous de déterminer le maximum de la fonction

$$m(t) = \sum_{k=1}^{n} m_k(t) .$$

Proposition :

Le nombre minimal de processeurs d'un réseau systolique pour la triangularisation par élimination de GAUSS d'une matrice d'ordre n selon le schéma de départ est

$$m = \begin{cases} (n+1)^2/4 & \text{si } n \text{ est impair} \\ n(n+2)/4 & \text{sinon .} \end{cases}$$

Démonstration :

Nous commencerons par faire remarquer que le lemme précédent est équivalent aux relations :

$$m_k(t) = \begin{cases} t-3(k-1) & \text{si } 3k-2 \leq t \leq n+2k-2 \\ 2n+k-t & \text{si } n+2k-1 \leq t \leq 2n+k-1 . \end{cases}$$

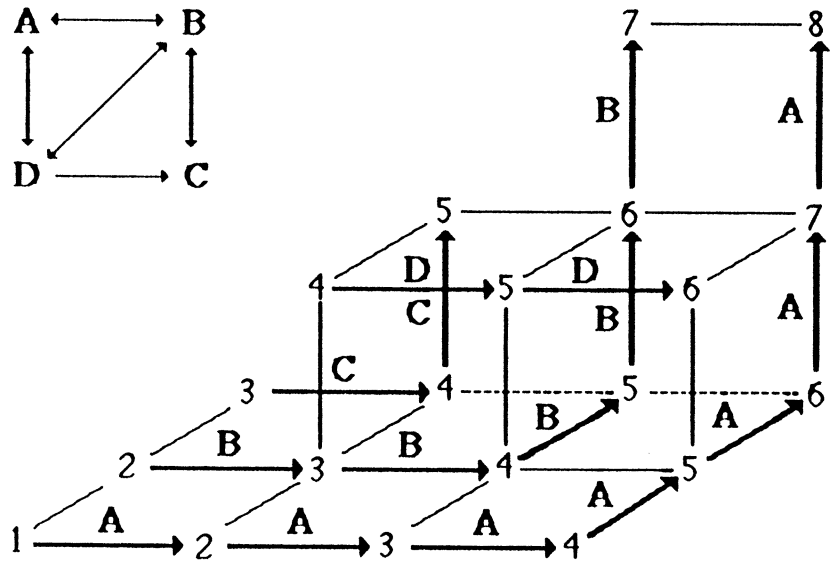
Considérons l'exemple pour n = 5 . Le tableau $(m_k(t))$ se présente comme suit :

t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$m_1(t)$	1	2	3	4	5	5	4	3	2	1				
$m_2(t)$				1	2	3	4	4	3	2	1			
$m_3(t)$							1	2	3	3	2	1		
$m_4(t)$										1	2	2	1	
$m_5(t)$													1	1
m(t)	1	2	3	5	7	8	9	9	8	7	5	3	2	1

On vérifie que la suite (m(t) ; $1 \leq t \leq 3n - 1$) est palindrome et croissante sur l'intervalle $[1 , \lfloor (3n-1)/2 \rfloor]$; on en déduit :
- si n est pair , m(t) atteint son maximum n(n+2)/4 pour t = 3n/2; ce maximum est aussi atteint pour t = (3n-2)/2 , (3n+2)/2 ;

- si n est impair , m(t) atteint son maximum $(n+1)^2/4$ pour t = (3n-1)/2 , (3n+1)/2 .
Connaissant le nombre minimal de processeurs nécessaires pour l'exécution de l'algorithme , nous allons nous intéresser au problème de l'existence d'un partitionnement optimal des points de calcul de **D** de sorte que le réseau induit ait une structure régulière .
Soit P un tel partitionnement . En vertu de la démonstration de la proposition , tout chemin C induit par P contient au moins un sous-chemin dont les sommets sont les points de date (3n-1)/2 et (3n+1)/2 ou (3n-2)/2 , 3n/2 et 3n+2/2 selon que respectivement l'ordre de la matrice est impair ou pair .

Considérons le diagramme espace-temps pour n=3 . Les dates auxquelles le maximum de points est simultanément calculé sont 4 et 5 .

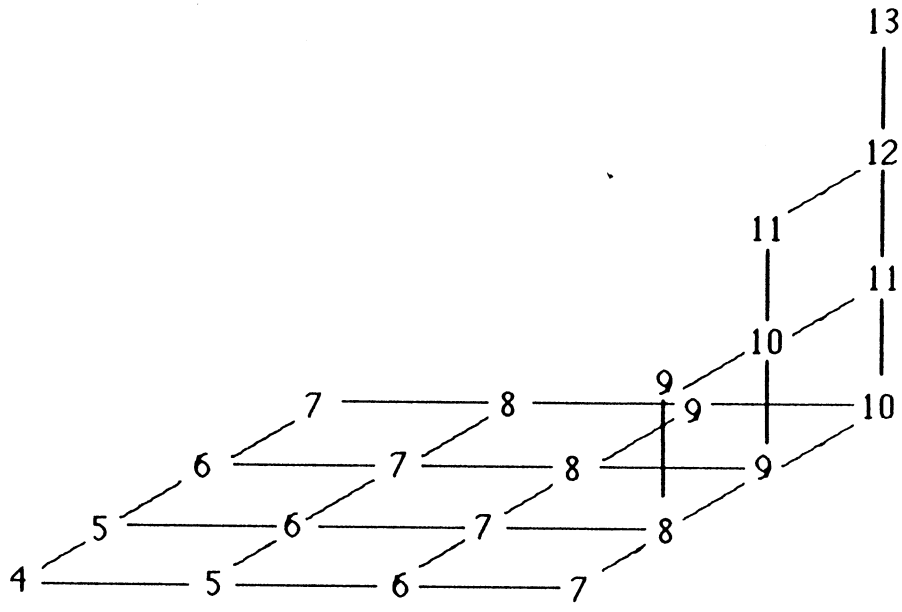


Un partitionnement optimal consiste , comme indiqué ci-dessus , à affecter aux processeurs **A** , **B** , **C** , **D** les points de calcul appartenant au même chemin . Le graphe d'interconnection du réseau induit est le réseau planaire et régulier ci-dessus.

Pour les valeurs supérieures de n , nous conjecturons que tout réseau planaire induit par un partitionnement optimal n'est pas régulier .

On est alors amené à résoudre le problème de l'existence d'un réseau minimal régulier dont le nombre de processeurs est compris entre m et $n^2/2$.
Désignons par Δ_k le sous ensemble de points (x,y,z) tels que

$$(z = k \text{ et } k \leq y \leq n-k+2) \quad \text{ou} \quad (z \geq k \text{ et } y = n-k+2).$$

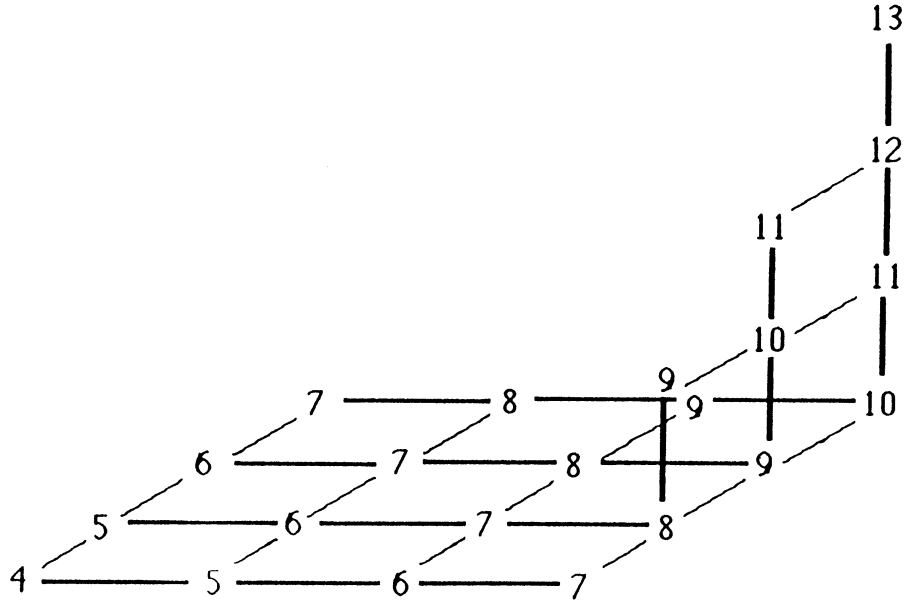


Δ_2 pour n = 5 .

Dans **D** on distingue ⌊(n+2)/2⌋ sous-domaines de ce type .
Considérons l'application **A** définie comme suit :

$$\mathbf{A}: \Delta_k \dashrightarrow N^2$$
$$(x,y,z) \dashrightarrow (k,x)$$

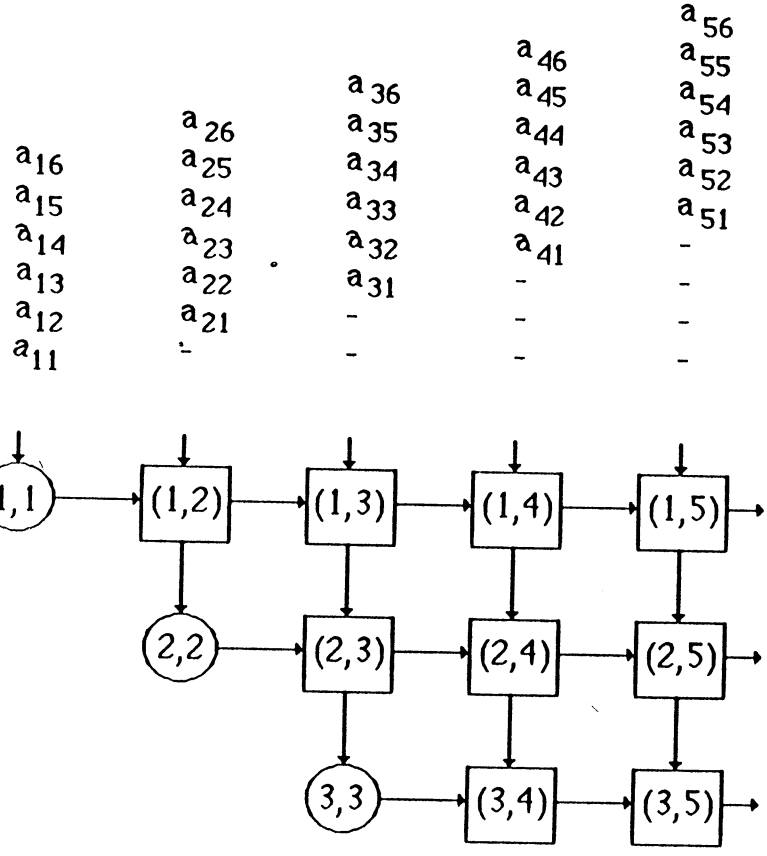
Elle associe à chaque sous domaine un réseau linéaire de processeurs (P(k,j) ; $k \leq j \leq n$) et à chaque processeur P(k,j) affecte les calculs à effectuer aux points de mêmes abscisses . Pour n = 5 le partitionnement de Δ_2 donne :



D'une manière globale , **A** associe à **D** un réseau de calculs dont le nombre de processeurs est :

$$\sum_{k=1}^{\lfloor (n+2)/2 \rfloor} (n-k+1) = \begin{cases} (3n+1)(n+1)/8 & \text{si n est impair} \\ 3n(n+2)/8 & \text{sinon} \end{cases}$$

On vérifie aisément que cette fonction d'allocation est compatible avec la fonction de temps .

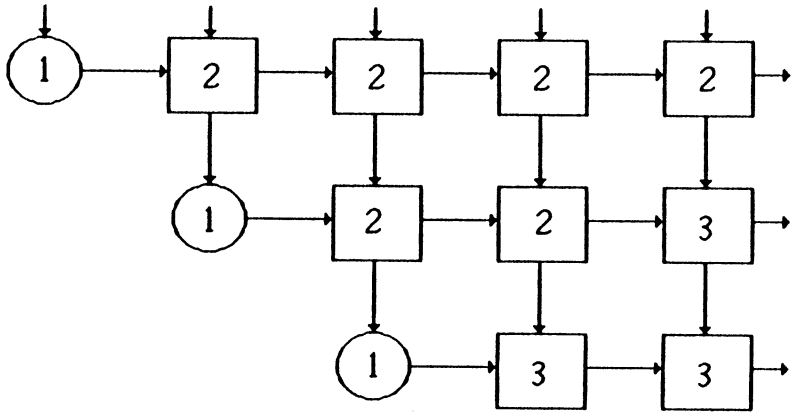


Réseau induit par **A** pour n = 5 .

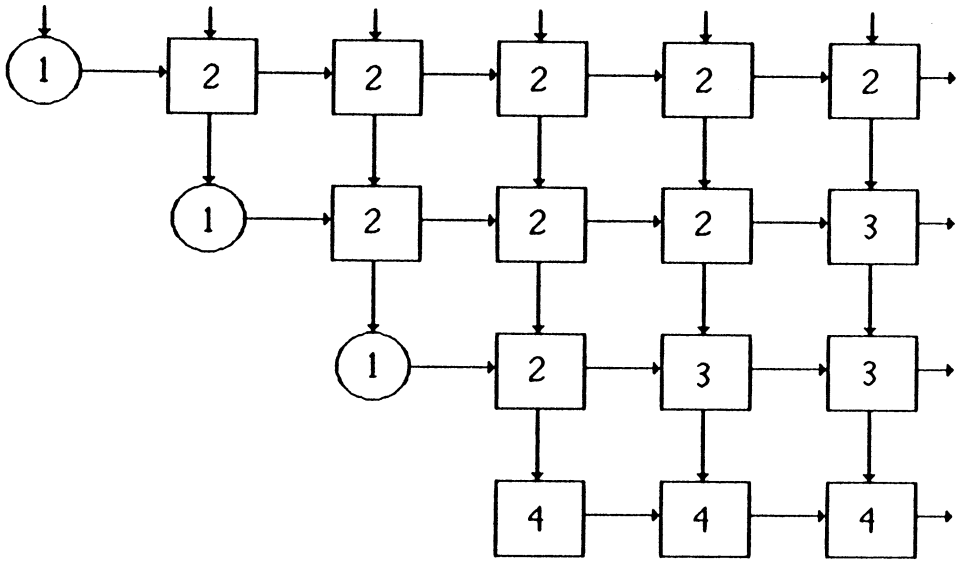
En effet considérons dans **D** deux points de calculs (x_1 , y_1 , z_1) et (x_2 , y_2 , z_2) distincts , de même date et appartenant au même domaine Δ_k . Il nous suffit de montrer que $x_1 \neq x_2$. Supposons $x_1 = x_2$; les deux points étant distincts , ils ont des ordonnées ou côtes différentes . Par définition de Δ_k on a :

- si $y_1 \neq y_2$ alors $z_1 = z_2$ et par suite les dates sont différentes d'où une contradiction
- si $z_1 \neq z_2$ alors $y_1 = y_2$; on a la même contradiction .

Topologie de la répartition des différents types de processeurs



Cas où n = 5

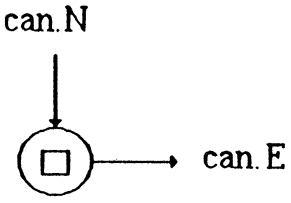


Cas où n = 6

4.2 - Fonctionnement du réseau :

Selon que n est impair ou pair , on distingue respectivement trois ou quatre types de cellules que nous désignons par 1 , 2 , 3 et 4 (Voir figures de la page précédente) .

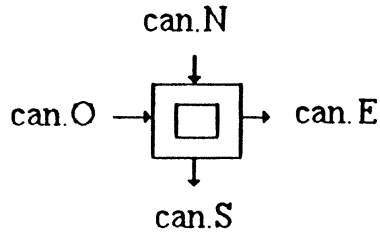
1) Les cellules de type 1 : Outre son registre interne , une cellule de type 1 dispose de deux canaux de communication avec son voisinage :



Son comportement comprend deux phases :
- La première comporte une étape . La cellule reçoit sur can.N la valeur de la variable u qu'elle stocke dans son registre .

- Pour une cellule en position (i,i) la deuxième comporte n-2(i-1) étapes . A chacune d'elles , la cellule reçoit w en entrée sur can.N , calcule v = w/u , et envoie en sortie sur can.E la valeur de la variable v :

2) Les cellules du type 2 :Une cellule du type 2 dispose d'un registre interne et de quatre canaux de communication avec son voisinage .



Le programme d'une cellule en position (i,j) comporte cinq phases .

- La première est identique à la première phase d'une cellule du type 1 .

- La deuxième comporte n-2i+1 étapes aucours de chacune desquelles la cellule reçoit en entrée sur can.N et can.O respectivement les variables w et v . Après la transformation w := w-u*v , les variables w et v sont envoyées en sortie respectivement sur can.O et can.E tandis que u reste stocké dans le registre .

- La troisième phase comporte une seule étape . La cellule reçoit sur can.N et can.O respectivement les variables w et v . Après la tranformation w := w-u*v , tandis que les variables u et v sont envoyées en sortie respectivement sur can.N et can.E , la variable w est stockée dans le registre interne .

- La quatrième comporte j-(i+1) étapes correspondant chacune à la tranformation w' = w-u*v. A chaque étape la cellule reçoit sur can.S et can.O respectivement les variables u et v puis envoie en sortie , après transformation , les variables u et v respectivement sur can.N et can.E tandis que w' reste stocké dans le registre .

- La cinquième phase comporte une seule étape . La cellule reçoit sur can.S la valeur de la variable u , effectue le calcul v = w / u puis envoie en sortie sur can.N et can.E respectivement les valeurs des variables u et v .

3) Les cellules du type 3 : Elles ne se distinguent des cellules du type 2 que de par leur cinquième phase . En effet cette phase comporte une seule étape qui consiste pour la cellule à retarder la valeur contenue dans son registre d'une unité de temps avant de l'envoyer en sortie sur can.N .

Notons que la quatrième phase comporte $\lfloor (n+2) / 2 \rfloor - i$ étapes pour une cellule en position (i,j) .

4) Les cellules du type 4 : De même structure que les deux précédentes , elles jouent un simple rôle de synchronisation . En effet , une telle cellule lit sur can.N la valeur de le variable u qu'elle envoie une unité de temps plus tard en sortie sur can.N .

Le déroulement correct du programme d'une cellule indépendamment de sa position dans le réseau nécessite des signaux de contrôle dont la définition fera l'objet des paragraphes qui vont suivre .

4 . 2 . 1 - Contrôle d'une cellule de type 1 :

- Une cellule du type 1 commmence son programme lorsqu'elle reçoit sur can.N un signal a_{ii} provenat de la diagonale de A . On peut repérer la date correspondante en affectant aux variables a_{ij} telles que $1 \leq j \leq \lfloor (n+1)/2 \rfloor$ et $j \leq i \leq n$ le signal s_{ij} qui vaut (0,1) si i = j et (0,0) sinon .
- Comme la première phase ne comporte qu'une seule étape , le passage à la deuxième se fait naturellement . La dernière étape de cette phase correspond à l'arrivée sur can.N d'une variable a_{ij} telle que i+j = n+2 . Cette date peut être repérée en adjoignant à chaque variable non codée a_{ij} telle que $i+j \leq n+2$ le signal s_{ij} qui vaut (1,0) si i+j = n+2 et (0,0) sinon .

4 . 2 . 2 - Contrôle d'une cellule du type 2 , 3 ou 4 :

- D'après le contrôle d'une cellule du type 1 , le premier calcul d'une cellule du type 2 ou 3 correspond à l'arrivée sur can.N d'une variable dont le signal vaut (0,0) .
Pour une cellule du type 4 dont tout le programme est réduit à une seule étape , la date correspondante est l'arrivée sur can.N d'une variable a_{ij} telle que $j = \lfloor (n+3)/2 \rfloor$
On peut assurer un comportement correct de ces processeurs en adjoignant aux variables a_{ij} un signal s_{ij} qui vaut (0,-1) si i = j et (-1,-1) sinon .

- Comme la première phase ne comporte qu'une seule étape , le passage à la deuxième se fait aisément .
- Tout comme dans le réseau d'inversion d'une matrice triangulaire , l'unique étape de la troisième phase correspond à la date t = n+1 à laquelle toutes les cellules de la même colonne du réseau se synchronisent . On est tenté de proposer la même solution à savoir celle du Firing Squad . Nous éviterons cette solution qui induit un réseau d'automates à six états internes . En effet , grâce au codage déjà effectué , le nombre d'états internes du réseau d'automates associé peut être réduit à deux . Pour ce faire remarquons que cette date correspond à celle de l'arrivée sur can.O d'une variable $v = a_{ij} / a_{ii}$ telle que $i+j = n+2$. Aussi suffit-il d'émettre en même temps que v le signal de a_{ij} . Cette étape est alors repérée par la lecture pour la première fois sur can.O d'une variable dont le signal vaut (1,0) .
- Le passage à la quatrième phase est naturel car la troisième ne comporte qu'une seule étape .
- Pour une cellule du type 2 , l'unique étape de la cinquième phase correspond à l'arrivée sur can.S d'une variable a_{ij} telle que i = j dont le signal de contrôle est (0,-1) ou non encore précisé . Cette étape est par exemple repérée en affectant aux variables concernées , à savoir les variables non codées a_{ij} telles que $j \leq i$, le signal s_{ij} qui vaut (0,1) si i=j et (1,1) sinon .
- Pour une cellule du type 3 , la dernière étape de la quatrième phase correspond à l'arrivée sur can.O d'une variable $v = a_{ij} / a_{ii}$ telle que j-i = 1 . Il suffit d'affecter à chacune de ces variables non encore codées le signal s_{ij} qui vaut (1,0) si j = i+1 et (0,0) sinon . Ainsi la dernière étape de la quatrième phase correspondra à l'arrivée pour la deuxième fois sur can.O d'une variable dont le signal vaut (1,0) .

5 - Introduction à OCCAM

Avec l'essor de la recherche dans les méthodologies de conception d'architectures parallèles , un formalisme de construction de logiciel de simulation s'avère d'une grande utilité . OCCAM , langage parallèle issu de CSP défini autour du traitement parallèle des tâches indépendantes et de la communication entre ces tâches répond bien à ce besoin .
Le langage dispose de deux entités de base :
- Le processus : il permet d'exprimer des actions et en tant que tel est formé d'une instruction unique , d'un groupe d'instructions ou même d'un groupe de processus .
- Le canal de communication joue le rôle d'un composant élémentaire de synchronisation entre deux processus communicants . C'est donc un buffer qui peut être lu par un processus dit processus-entrée seulement lorsqu'il est plein et écrit par un processus dit processus-sortie si et seulement si il est vide .
En OCCAM , deux processus concurrents communiquent de manière asynchrone par échange de messages sur un canal selon un processus de "rendez-vous". Ils disposent donc d'un moyen pour le transfert des données et pour signaler leur

attente de données . Les effets de cet asynchronisme sont résolus par le canal de communication qui au lieu de se comporter comme une mémoire commune se comporte plutôt comme un processus intermédiaire s'exécutant en parallèle avec les processus communicants .

5 .1 - Processus élémentaires

On distingue trois processus de base en OCCAM .

5.1.1 - Affectation : il transfère la valeur d'une variable ou d'une expression y à une varible locale x et termine .

x := y

5.1.2 - Entrée : il transfert la valeur lue sur un canal can.entrée à une variable locale x puis termine .

can.entrée ? x

5.1.3 - Sortie : il écrit sur un canal can.sortie la valeur d'une variable locale x puis termine .

can.sortie ! x .

A ceux-ci on peut rajouter les processus :

5.1.4 - STOP qui commence sans jamais terminer .

5.1.5 - SKIP qui termine à l'instant où il commence ; il sert , comme nous le verrons plus loin à assurer une terminaison correcte pour un processus conditionnel.

5.1.6 - Timer input : met une variable x à une valeur lue à partir d'une horloge qui change à intervalle régulier .

TIME ? x .

Une variante de ce processus est TIME ? AFTER x consiste à attendre jusqu'à ce que la valeur lue à partir de l'horloge atteigne la valeur AFTER x où x est une expression .

5.2 - Les constructeurs OCCAM

Ils servent à la construction de nouveaux processus à partir d'autres déjà existants . Par une technique de niveau d'indentation , les constructeurs OCCAM facilitent une construction en profondeur par une combinaison d'un nombre quelconque de processus communicants dont l'ensemble peut être vu par l'environnement extérieur comme un et un seul processus .

5.2.1 - SEQ : sert à construire un processus séquentiel dont les processus composants sont exécutés l'un après l'autre de sorte que l'un soit effectivement terminé avant que l'autre ne commence ;

```
SEQ
    processus 1
    .
    .
    .
    processus n
```

5.2.2 - WHILE : c'est le constructeur répétitif ; sa syntaxe est :

```
WHILE expression
    processus
```

le processus composant "processus" est répété tant que l'évaluation de "expression" donne la valeur TRUE .

5.2.3 - PAR : est l'opérateur parallèle ; il a pour syntaxe :

```
PAR
    processus 1
    processus 2
    .
    .
    .
    processus n
```

Ce constructeur impose à ses processus composants d'être exécutés de façon concurrente . Tous les processus commencent en même temps et le processus PAR termine lorsque tous ses composants ont terminé .

5.2.4 - ALT : en général un processus possède plus d'un canal sur lesquels des lectures sont possibles dès que un au moins est plein ; le comportement ultérieur du processus dépend du canal lu . Le constructeur ALT choisit un seul de ses processus-entrées .

```
ALT
    processus 1
    processus 2
    .
    .
    .
    processus n
```

Le processus ALT attend jusqu'à ce que un de ses composants , qui en fait sont des processus gardés , soit prêt à être exécuté . Alors un et un seul est choisi de manière non déterministe et exécuté puis le processus ALT termine .
Un autre type de constructeur alternatif est PRIALT qui induit un processus alternatif avec priorité sur les gardes .

5.2.5 - IF : c'est le constructeur conditionnel dont la syntaxe est

```
IF
  expression
    processus
```

Le processus conditionnel "processus" est prêt si "expression" est évalué à TRUE . Dans le cas d'une multiplicité de composants conditionnels , le processus IF est prêt si l'un au moins de ses composants l'est ; alors le premier à être prêt est exécuté et le processus termine . Si par contre aucun des processus composants n'est prêt alors il est nécessaire que le constructeur IF termine immédiatement pour qu'il termine correctement d'où l'utilité du processus SKIP . Alors la syntaxe correcte est

```
IF
  expression
    processus
  TRUE
    SKIP
```

2.6 - FOR : Comme tout multiplexeur , il sert à créer une liste de processus identiques qui peuvent s'exécuter selon le type du constructeur qu'il accompagne . Par exemple le processus

```
PAR i= [k FOR n]
    processus (i)
```

construit les processus identiques "processus(k)" , . . . , "processus(n-k+1)" qu' il exécutera en paralèlle .

5.3 Déclarations

En OCCAM , les déclarations introduisent des identificateurs locaux auxquels on ne peut accéder que par des processus . Les types des identificateurs sont VAR pour les variables , CHAN pour les canaux , DEF pour les constantes et PROC pour les processus .

5.3.1 - VAR : cette déclaration introduit une liste d'identificateurs ou une liste de vecteurs d'identificateurs . Par exemple :

VAR x , y , z :

déclare les variables x , y , z ;

VAR vecteur[4] , tableau[byte 80] :

déclare d'une part un vecteur "vecteur" de 4 composantes et d'autre part un vecteur "tableau" de 80 octets dont les composantes de rang 0 correspondent à la taille du vecteur .

5.3.2 - CHAN : Tout comme dans le cas de la déclaration VAR , CHAN introduit une liste de canaux ou de vecteurs de canaux . En exemple :

CHAN can.in , can.out , can[10] :

5.3.3 - DEF : introduit également une liste d'identificateurs ou de vecteurs d'identificateurs à valeurs constantes pour chacune de leurs occurences dans le processus qui suit leur déclaration .
En exemple :

DEF min = 10 , max = 20 :
DEF voyelles = aeiou :

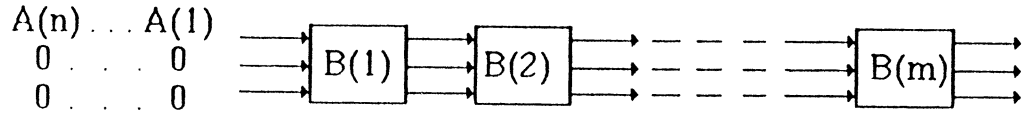
5.3.4 - PROC : Tout processus construit en OCCAM peut être nommé . L'utilisation du nom d'un processus P(1) dans un autre P(2) a pour effet de substituer dans P(2) toutes les occurences de P(1) par le texte de celui-ci . En clair , un processus OCCAM peut comporter des paramètres formels dont les types sont VAR , VALUE CHAN . Lorsqu'un tableau est utilisé comme paramètre formel d'un processus , sa taille n'a pas besoin d'être explicitée . Aussi lors d'une substitution , un vecteur de taille différente peut être utilisé comme paramètre effectif .

5.4 - Les expressions

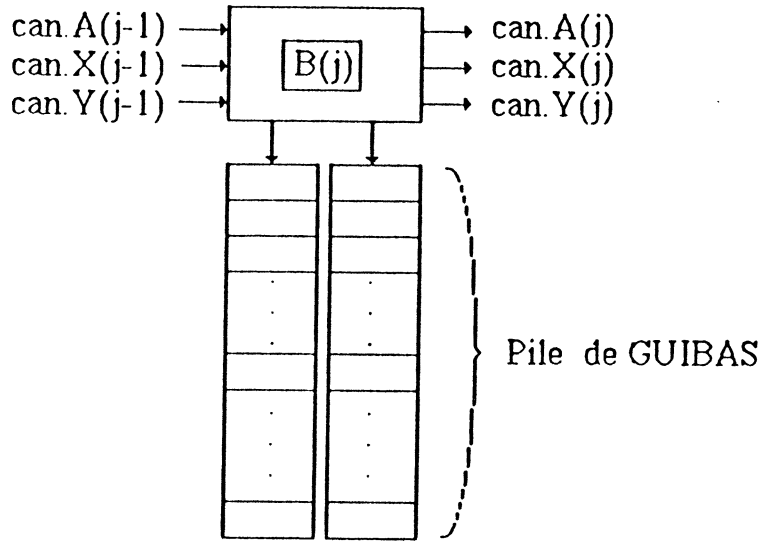
En OCCAM , une expression est évaluée à une valeur unique . Les expressions sont classées en fonction des opérateurs utilisés pour leur construction . On distingue tous les types d'expressions habituellement utilisés dans des langages informatiques tel que PASCAL .Toutefois , notons que les opérations arithmétiques ne s'appliquent que sur des entiers .

A l'issue de cette sommaire et brève description , on se fait une idée de toutes les possibilités que OCCAM présente également pour la simulation de systèmes synchrones tels que les réseaux systoliques .
Pour mieux fixer les idées sur le langage , nous donnons le programme d'un réseau

systolique pour la détermination d'une plus longue sous-suite commune à deux chaînes de caractères (A(i) ; $1 \leq i \leq n$) et (B(i) ; $1 \leq i \leq m$) avec $n \geq m$ [17].
Le réseau est un ensemble de m processeurs P(j) en pipeline dont un des registres internes contient B(j) .



Comme l'indique la figure ci-après , chaque processeur comprend une cellule de calculs arithmétiques et logiques et une pile de GUIBAS [14] .



Désignons par llcs(i,j) la plus longue sous-suite commune aux chaînes (A(k) ; $1 \leq k \leq i$) et (B(k) ; $1 \leq k \leq j$) .

On sait [17] que pour tout (i,j) tel que $i \geq 1$ et $j \geq 1$,

$$llcs(i,j) = \max \{llcs(i-1,j) , llcs(i,j-1) , llcs(i-1,j-1)+e(i,j)\}$$

où llcs(0,j) = llcs(i,0) = 0 et e(i,j) est un booléen qui vaut 1 si A(i) = B(j) et 0 sinon .
Le comportement d'une cellule P(j) comporte deux phases :
- La première est composée de n étapes dont chacune consiste dans un premier temps à lire sur can.A(j-1) , can.X(j-1) et can.Y(j-1) respectivement les valeurs des variables

$$a = A(i)$$
$$x = llcs(i-1,j-1)$$
$$y = llcs(i,j-1)$$

pour le calcul de k = llcs(i,j) . Soit (i' , k') le couple stocké au sommet de la pile du

processeur ; le couple (i,k) est stocké dans la pile si et seulement e(i,j) = 1 et k ≠ k'. Dans un second temps les valeurs des variables

$$a = A(i)$$
$$x = llcs(i-1,j)$$
$$y = llcs(i,j)$$

sont envoyées en sortie respectivement sur les canaux can.A(j) , can.X(j) et can.Y(j)
- La seconde phase consiste à extraire de P(j) un des caractères de la chaîne de longueur p = llcs(n,m) .
Pour cela , dans un premier temps le processeur P(j) reçoit sur can.X(j) et can.Y(j) le signal (u ,v) où v désigne la longueur de la chaîne qui reste à extraire du réseau . Notons que pour j = m , (u , v) = (m+1 , llcs(n,m)).
Si le couple (i,k) qui à cet instant est au sommet de la pile est tel que i ≥ u ou k ≠ v , le processeur se contente de transmettre à son voisin de gauche par les canaux can.X(j-1) et can.Y(j-1) le même signal . Dans le cas contraire , le processeur envoie à tous les processeurs en position h < j un signal dl(v) dont l'effet est d'extraire du sommet de la pile tout couple dont la seconde composante est égale à v ; simultanément B(j) est envoyé en sortie sur can.A(j) . Pour finir le processeur envoie à son voisin de gauche la longueur v -1 de la suite qui reste à extraire du réseau et ce par le signal (i,v-1) .

```
CHAN Keyboard AT 2 :
CHAN Screen   AT 1 :
CHAN FILEIN   AT 3 :
CHAN FILEOUT  AT 11:
CHAN CA[100],CX[100],CY[100]:
VAR AA[150],BB[100],M,N,J:
DEF EndBuffer = -3 , OPENFORWRITE = -12 , ENDNAME = -5 ,
    ENDRECORD = -7 , NEXTRECORD = -9 , ENDFILE = -4 :

PROC OUTPUTSTR(CHAN OUT,VALUE STRING[]) =
  VAR I :
  SEQ I =[1 FOR STRING[BYTE 0]]
    OUT ! STRING[BYTE I] :

PROC WRITELN =
  Screen ! '*C';'*N';EndBuffer :

PROC WRITELNBIS =
  VAR X :
  SEQ
    FILEOUT ! '*C';'*N';ENDRECORD
    FILEIN ? X :

PROC ENTREE=
-- chargement du reseau --
  VAR K,X,Y,LONG:
  SEQ
    PAR
      K:=1
      Y:=0
      X:=0
    WHILE K<=N
      SEQ
        PAR
          CA[0]!AA[K]
          CX[0]!X
          CY[0]!Y
        K:=K+1
    CX[0]?LONG
    IF
      LONG<>0
        SEQ
          K:=1
          WHILE K<=(M+2)
            SEQ
              PAR
                CX[0]?X
                CY[0]?Y
              K:=K+1
          CA[0]!' '
      TRUE
        SKIP:

PROC TRAITER(VALUE J)=
-- programme d'une cellule
  VAR P1[BYTE 13],P2[BYTE 13]:
  VAR OPERAT,NBRE,LLP,LONG:
  VAR B,X,Y,E,K:

  PROC PILE(VAR P[],VALUE OPERAT,DONNEE)=
```

```
-- gestion d'une pile de Guibas
    DEF L=13:
    VAR T[BYTE L]:
    VAR I,NBRECYCLE:
    SEQ
      IF
        OPERAT='E'
          P[BYTE 1]:=DONNEE
        OPERAT='D'
          P[BYTE 2]:=' '
      PAR I=[1 FOR (L-1)]
        T[BYTE I]:=P[BYTE I]
      NBRECYCLE:=1
      WHILE NBRECYCLE<=3
        SEQ
          SEQ I=[1 FOR (L-2)]
            SEQ
              IF
                (P[BYTE (I-1)]<>' ') AND (P[BYTE I]<>' ') AND
                  (P[BYTE (I+1)]=' ')
                  SEQ
                    T[BYTE I]:=' '
                    T[BYTE (I+1)]:= P[BYTE I]
                (P[BYTE (I-1)]=' ') AND (P[BYTE I]=' ') AND
                  (P[BYTE (I+1)]<>' ')
                  SEQ
                    T[BYTE I]:=P[BYTE (I+1)]
                    T[BYTE (I+1)]:=' '
                TRUE
                  SKIP
          PAR I=[1 FOR (L-1)]
            P[BYTE I]:=T[BYTE I]
          NBRECYCLE:=NBRECYCLE+1:

  PROC RECHMAX(VAR MAX)=
    VAR T[3],I:
    SEQ
      PAR
        T[0]:=LLP
        T[1]:=E+X
        T[2]:=Y
      PAR
        MAX:=T[0]
        I:=1
      WHILE I<=2
        SEQ
          IF
            MAX<=T[I]
              SEQ
                MAX:=T[I]
                I:=I+1
            TRUE
              I:=I+1:

  SEQ
    PAR
      LLP:=0
      B:=BB[J]
      NBRE:=1
      OPERAT:='E'
```

```
  SEQ X=[1 FOR 12]
    PAR
      P1[BYTE X]:=' '
      P2[BYTE X]:=' '
WHILE NBRE<=N
  VAR A,LLC:
  SEQ
    PAR
      CA[J-1]?A
      CX[J-1]?X
      CY[J-1]?Y
    IF
      A=B
        E:=1
      A<>B
        E:=0
    RECHMAX(LLC)
    PAR
      CA[J]!A
      CX[J]!LLP
      CY[J]!LLC
    LLP:=LLC
    IF
      (E=1) AND (LLC<>P2[BYTE 2])
        SEQ
          OPERAT:='E'
          PAR
            PILE(P1,OPERAT,NBRE)
            PILE(P2,OPERAT,LLC)
      TRUE
        SKIP
    NBRE:=NBRE+1
CX[J]?LONG
CX[J-1]!LONG
IF
  LONG<>0
    SEQ
      NBRE:=1
      WHILE NBRE<=((M-J)+1)
        VAR DL,W:
        SEQ
          PAR
            CX[J]?DL
            CY[J]?W
          PAR
            CX[J-1]!DL
            CY[J-1]!W
          IF
            (DL='O') AND (W=P2[BYTE 2])
              SEQ
                OPERAT:='D'
                PAR
                  PILE(P1,OPERAT,W)
                  PILE(P2,OPERAT,W)
            TRUE
              SKIP
          NBRE:=NBRE+1
      PAR
        CX[J]?X
        CY[J]?Y
```

```
        IF
          (P1[BYTE 2]>=X) OR (P2[BYTE 2]<>Y)
            DEF CARACT = ' ':
            VAR DL,W:
            SEQ
              PAR
                DL:='N'
                W:=0
              PAR
                CA[J]!CARACT
                CX[J-1]!DL
                CY[J-1]!W
          TRUE
            VAR DL:
            SEQ
              DL:='O'
              PAR
                CX[J-1]!DL
                CY[J-1]!Y
                CA[J]!B
              PAR
                X:=P1[BYTE 2]
                Y:=Y-1
        PAR
          CX[J-1]!X
          CY[J-1]!Y
        NBRE:=1
        WHILE NBRE<=J
          VAR A:
          SEQ
            CA[J-1]?A
            CA[J]!A
            NBRE:=NBRE+1
      TRUE
        SKIP:

PROC SORTIE=
  VAR Y,NBRE:
  SEQ
    NBRE:=1
    WHILE NBRE<=N
      VAR X,A:
      SEQ
        PAR
          CA[M]?A
          CX[M]?X
          CY[M]?Y
        NBRE:=NBRE+1
    CX[M]!Y
    IF
      Y=0
        SEQ
          OUTPUTSTR(Screen,"LES DEUX CHAINES N ONT PAS DE S/SUITE COMMUNE")
          OUTPUTSTR(FILEOUT,"LES DEUX CHAINES N ONT PAS DE S/SUITE COMMUNE")
          WRITELN
          WRITELN
          WRITELNBIS
      Y<>0
        DEF CARACT = ' ',DL='N',W=0:
        VAR X,I,P[100]:
```

```
        SEQ
          OUTPUTSTR(Screen,"LA PLUS LONGUE S/SUITE COMMUNE EST : ")
          Screen ! EndBuffer
          OUTPUTSTR(FILEOUT,"LA PLUS LONGUE S/SUITE COMMUNE EST : ")
          PAR
            CX[M]!DL
            CY[M]!W
          X:=N+1
          PAR
            CX[M]!X
            CY[M]!Y
            NBRE:=1
            I:=0
          WHILE NBRE<=(M+1)
            VAR A:
            SEQ
              CA[M]?A
              IF
                A<>CARACT
                  SEQ
                    I:=I+1
                    P[I]:=A
                TRUE
                  SKIP
              NBRE:=NBRE+1
          WHILE I>=1
            SEQ
              Screen!P[I]
              FILEOUT ! P[I]
              I:=I-1
          WRITELN
          WRITELNBIS :

PROC INIT(VAR A[],VAR P)=
  SEQ
    Keyboard?A[P]
    WHILE A[P]<>'*C'
      SEQ
        Screen!A[P];EndBuffer
        FILEOUT ! A[P]
        P:=P+1
        Keyboard?A[P]
    WRITELN
    WRITELN
    WRITELNBIS
    P:=P-1:

PROC OPENFILE =
  VAR STATUS , X :
  SEQ
    FILEOUT ! OPENFORWRITE
    Keyboard ? X
    WHILE X <> '*C'
      SEQ
        Screen ! X;EndBuffer
        FILEOUT ! X
        Keyboard ? X
    FILEOUT ! ENDNAME
    FILEIN ? STATUS :
```

```
PROC CLOSEFILE =
  VAR ACK :
  SEQ
    FILEOUT ! ENDFILE
    FILEIN ? ACK :

SEQ
  WRITELN
  WRITELN
  OUTPUTSTR(Screen,"ENTRER LE NOM DU FICHIER RESULTAT : ")
  Screen ! EndBuffer
  OPENFILE
  WRITELN
  WRITELN
  OUTPUTSTR(Screen,"ENTRER LA CHAINE A : ")
  Screen ! EndBuffer
  OUTPUTSTR(FILEOUT,"ENTRER LA CHAINE A : ")
  N:=1
  INIT(AA,N)
  OUTPUTSTR(Screen,"ENTRER LA CHAINE B : ")
  Screen ! EndBuffer
  OUTPUTSTR(FILEOUT,"ENTRER LA CHAINE B : ")
  M:=1
  INIT(BB,M)
  PAR
    ENTREE
    PAR J=[1 FOR 100]
      IF
        J<=M
          TRAITER(J)
        TRUE
          SKIP
    SORTIE
  CLOSEFILE
------
```

Exemple d'execution

```
ENTRER LA CHAINE A : abcdbb
ENTRER LA CHAINE B : cbacbaaba
LA PLUS LONGUE S/SUITE COMMUNE EST : acbb
```

6 - Conclusion

A la lumière des exemples que nous venons d'étudier , la méthode de partitionnement conduit pour le schéma de résolution séquentielle de départ , à une parallélisation optimale en temps . Le problème d'une allocation optimale en nombre de processeurs reste ouvert . Par rapport aux méthodes de projection , les réseaux obtenus par partitionnement sont meilleurs . En effet nous assistons comme pour l'inversion d'une matrice triangulaire à la réduction de moitié la taille des meilleurs réseaux publiés jusqu'ici dans la littérature .

Références

[1] **A . APOSTOLICO and A . NEGRO :** Systolic algorithms for string manipulation , IEEE , Trans . Computer 4 (1984) .

[2] **A . DAVIS and U . WEISER :** A wavefront notion tool for VLSI arrays , VLSI systems and Computation , H . T . Kung and al eds Computer Science Press pp 226 - 234 , Oct 1981 .

[3] **A . V . KULKARNI and D . W . L . YEN :** Systolic processing and an implementation for signal and image processing , IEEE , Trans . Computer 10 , 1982 .

[4] **A . WAKSMAN :** An optimal solution to the firing squad synchronization problem , Information and Control 9 , pp 66 - 78 , 1966 .

[5] **A . WINKLER and W . L . MIRANKER :** A space-time representation of computational structure , Computing vol . 32 , pp 93 - 114 , 1984 .

[6] **B . W . WAH and G . J . LI :** The design of optimal systolic arrays , IEEE TC , vol . c - 33 , N°10 , pp 66 - 77 , 1985 .

[7] **C . E . LEISERSON and J . B . SAXE :** Optimizing synchronous systems ans proc . 22nd annual symposium on Fondation of Computer Science , IEEE Computer Soc . , pp 23 - 34 , Oct 1981 .

[8] **D . I . MOLDOVAN :** On the design of algorithms for VLSI systolic arrays Proc . of IEEE , vol . 71 , N°1 , pp113 - 120 , 1 (1983) .

[9] **H . M . AHMED , J . M . DELOSME and M . MORF :** Highly concurrent computing structures for matrix arithmetic and signal processing IEEE , Computer Magazine 15 , 1 (1982) .

[10] **H . T . KUNG and C . E . LEISERSON :** Systolic arrays for VLSI , Proc . Symp . on Sparse Matrix Computation , Knoxville Tennessee , I . S . Duff , G . W . Stewart eds 1978 , pp 256 - 282 .

[11] **H . T . KUNG and W . M . GENTLEMAN :** Matrix triangularization by systolic arrays , Proc . SPIE 298 , Real-time signal processing , 4 (1981) .

[12] **H . T . KUNG :** Why systolic architectures , Computer , vol . 15 , N°1 pp 37 - 46 (1982) .

[13] **J . MAZOYER :** Une solution en temps minimal à six états internes au problème de la synchronisation d'une ligne de fusiliers , Thèse de Doctorat Université de Lyon 1 , 1986 .

[14] **L . J . GUIBAS and F . M . LIANG :** Systolic stacks , queues and counters , Proceedings of the 1982 Conference on ad vanced research in VLSI , MIT .

[15] **L . MELKEMI :** Réseaux systoliques pour la résolution de problèmes linéaires , Thèse de Doctorat de troisième cycle , Université de Grenoble 1986 .

[16] **M . TCHUENTE and Y . ROBERT :** An efficient array for the 1D recursive convolution problem , Journal of VLSI and Computer Systems , vol . 1 , N°4 , pp 398 - 407 (1986) .

[17] **M . TCHUENTE and Y . ROBERT :** A systolic array for the longest common subsequence problem , RR IMAG .

[18] **M . TCHUENTE and Y . ROBERT :** Réseaux systoliques pour des problèmes de mots , RAIRO , Informatique Théorique .

[19] **P . QUINTON :** The systematic design of systolic arrays , IRISA Rapport interne N°193 , 1983 .

[20] **T . MUNTEAN :** Introduction à OCCAM langage parallèle issu de CSP pour la programmation des systèmes de transinateurs , Laboratoire de Génie Informatique , Décembre 1983 .

DEUXIEME PARTIE

EVALUATION DES PERFORMANCES D'UN RESEAU DE PROCESSEURS EN PIPELINE DE CYCLES DE TEMPS PERIODIQUES

1- Introduction

Dans les architectures parallèles , un problème clef reste celui de la synchronisation des processeurs . Usuellement , celle-ci est globale et est assurée par une horloge commune qui gouverne tous les processeurs ; c'est le cas des réseaux systoliques [1] Il est cependant des réseaux pour lesquels aucune synchronisation globale n'est nécessaire ; tel est par exemple le cas des processus de synchronisation de certains langages CSP comme OCCAM [2] où chaque processeur travaille à son propre rythme , la synchronisation s'effectuant par des procédures locales.
Dans ce chapitre , nous étudierons la performance d'un tel modèle théorique .

On considère un réseau de m processeurs P(1) , P(2) , . . . P(m) en pipeline dans lequel les données circulent dans la même direction d'un processeur au voisin .

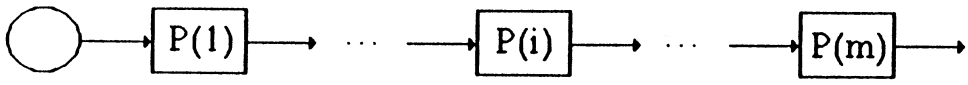


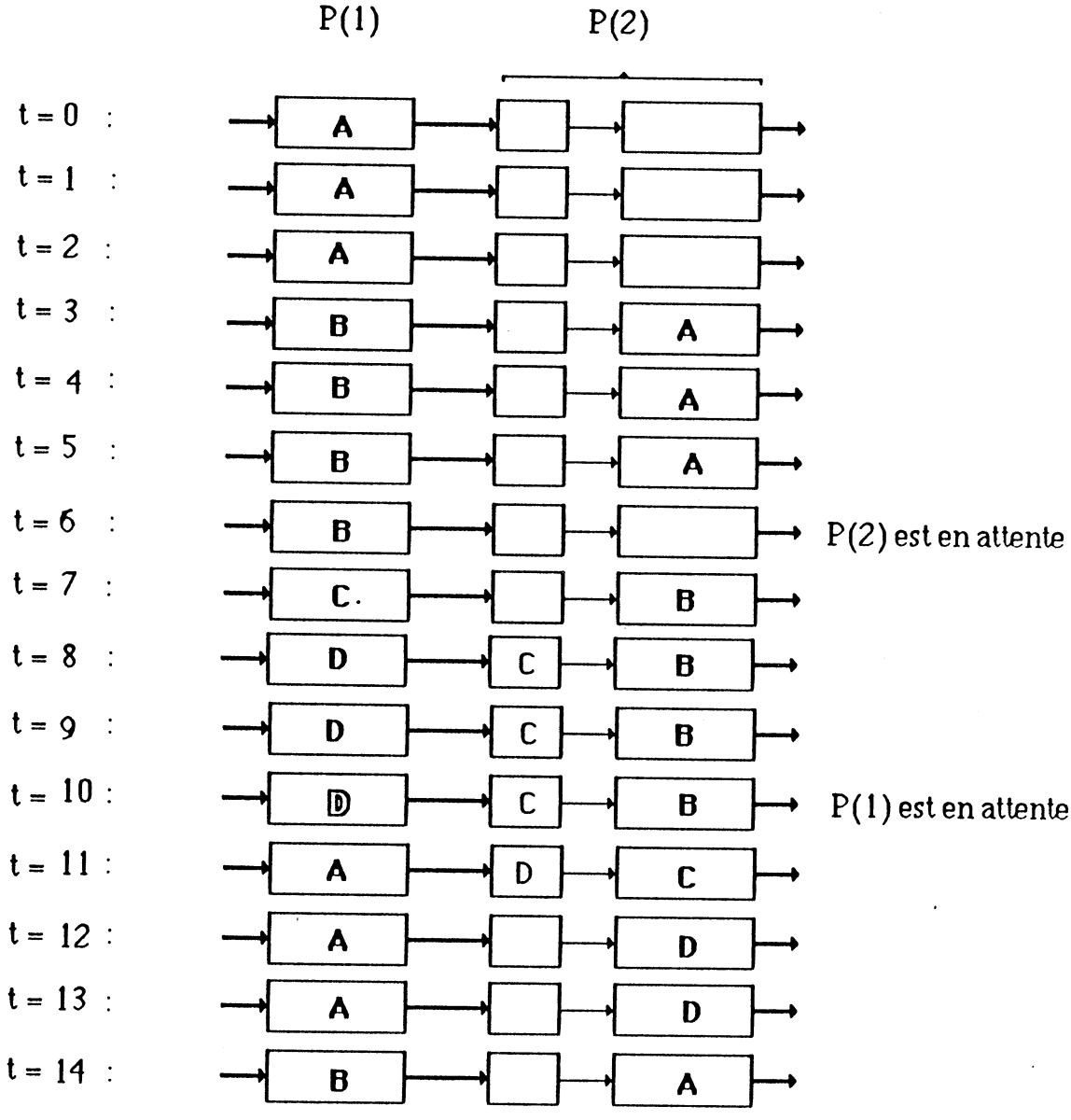
figure 1 .

On suppose que :
- Les données entrent dans P(1) par le moyen d'un dispositif externe et ce sans délai .
- Chaque donnée qui entre dans P(1) est soumis à m transformations réalisées successivement par P(1) , P(2) , . . . , P(m) avant d'être évacuée du réseau par P(m) .
- Chaque processeur dispose d'un buffer de profondeur 1 dans lequel il peut stocker une donnée .
- La communication entre deux processeurs s'effectue selon le principe de "rendez-vous" [2] et est de durée négligeable .
- Le cycle de temps d'un processeur est une séquence périodique s(j) , s(j+1) , . . . , s(n) , s(1) , s(2) , . . . , s(j-1) où s est une permutation fixée de 1 , 2 , . . . , n et j peut varier d'un processeur à un autre .

Du fait de la phase d'initialisation , numérotons k = i , i+1 , . . . les tâches réalisées par le processeur P(i) . Alors :
- Pour $i \geq 2$, $k \geq i$, les données nécessaires à la $k^{ème}$ tâche de P(i) sont fournies par la $(k-1)^{ème}$ tâche de P(i-1) .
- P(i) stocke dans son buffer la donnée nécessaire à sa $k^{ème}$ tâche tandis qu'il réalise sa $(k-1)^{ème}$ tâche .

Aucune synchronisation globale n'est nécessaire pour une évolution correcte de ce système . A chaque moment , un processeur peut se trouver dans l'une des situations suivantes :
- Il peut être en train de réaliser sa $k^{ème}$ tâche ,
- Il peut être dans l'attente de la donnée nécessaire à sa $k^{ème}$ tâche ; dans ce cas , il a fini sa $(k-1)^{ème}$ tâche tandis que P(i-1) réalise sa $(k-1)^{ème}$ tâche (voir sur la figure 2 la situation de P(2) à l'nstant t = 6) ,
- Il peut attendre que P(i+1) vide son buffer ; cette situation apparait quand P(i) a fini sa $k^{ème}$ tâche et P(i+1) est en train de réaliser sa $(k-1)^{ème}$ tâche (voir sur la

Exemple 1 : Le cycle de temps de P(1) et P(2) est $s = (3, 4, 1, 2)$.



Les situations aux instants t = 3 et t = 14 sont identiques.

Figure 2.

figure 2 la situation de P(1) à l'instant t = 10).

2- Notations et définitions

Désignons par (d(i,k) ; k ≥ i) , la séquence correspondant au cycle de temps de P(i) où d(i,k) est le nombre d'unités de temps nécessaires à P(i) pour réaliser sa $k^{ème}$ tâche .
Considérons une période pendant laquelle P(1) est plus rapide que P(2) .
- On dira que P(1) est retardé par P(2) après sa $k^{ème}$ tâche s'il ne peut pas transmettre à P(2) les résultats correspondant à sa $k^{ème}$ tâche .
Cette situation apparait lorsque P(1) a fini sa $k^{ème}$ tâche tandis que P(2) est en train de réaliser sa $(k-1)^{ème}$ tâche et son buffer est occupé par le résultat de la $(k-1)^{ème}$ tâche de P(1) (voir sur la figure 2 la situation à l'instant t = 10) .
Ainsi P(1) est retardé par P(2) pour la première fois après sa $k^{ème}$ tâche si et seulement si

$$\sum_{i=1}^{k} d(1,i) < \sum_{i=2}^{k-1} d(2,i)$$

et

$$\sum_{i=1}^{j} d(1,i) < \sum_{i=2}^{j-1} d(2,i) \quad \text{pour tout } j < k .$$

Dans une telle situation , P(1) doit attendre jusqu'à ce que P(2) ait fini sa $(k-1)^{ème}$ tâche et vidé son buffer pour commencer sa $(k+1)^{ème}$ tâche . Il aura été retardé de

$$r(k) = \sum_{i=2}^{k-1} d(2,i) - \sum_{i=2}^{k} d(1,i)$$

Il est alors naturel de prendre pour durée réelle de la $k^{ème}$ tâche de P(1)

$$d'(1,k) = d(1,k) + r(k) .$$

Du fait du buffer de P(2) , la $(k+1)^{ème}$ tâche de P(1) et la $k^{ème}$ tâche de P(2) commencent simultanément ; il suit que P(1) est retardé par P(2) pour la deuxième fois après sa $k'^{ème}$ tâche si et seulement si

$$\sum_{i=k+1}^{k'} d(1,i) < \sum_{i=k}^{k'-1} d(2,i) \quad \text{pour tout } k+1 \leq j < k' .$$

Identiquement , dans une telle situation , P(1) doit attendre jusqu'à ce que P(2) ait

fini sa $(k'-1)^{ème}$ tâche et vidé son buffer pour commencer sa $(k'+1)^{ème}$ tâche . P(1) est donc retardé de

$$r(k') = \sum_{i=k}^{k'-1} d(2,i) - \sum_{i=k+1}^{k'} d(1,i) .$$

Nous dirons alors que la durée réelle de la $k'^{ème}$ tâche de P(1) est

$$d'(1,k') = d(1,k) + r(k') .$$

D'une manière générale , nous pouvons de la même façon caractériser la situation où P(1) est retardé par P(2) pour la $q^{ème}$ fois et inversément .

Puisque les entrées de P(1) sont fournies sans délai par un système externe , la performance du réseau est entièrement caractérisée par la limite T de t(i)/i où t(i) est l'instant auquel débute la $i^{ème}$ tâche de P(1) . Clairement , T est la durée moyenne d'une tâche de P(1) .
Le cycle de temps de P(1) étant une suite périodique de période s(1) , s(2) , . . . , s(n) où s est une permutation fixée d'ordre n alors :

$$T \geq \sum_{i=1}^{n} i/n = (n+1)/2 .$$

Dans les paragraphes à venir nous préciserons les bornes de T et les cycles de temps associés .

3 - Retard maximal

Dans ce paragraphe , nous exhibons un réseau dont le cycle de temps correspond à la plus mauvaise performance .
Pour cela considérons un réseau de deux processeurs tournant à plein régime c'est à dire que nous négligeons la phase d'initialisation de P(2) .
Une condition nécessaire pour que P(1) soit retardé au maximum est qu'il soit à son rythme le plus rapide ; soit s la permutation associée à un tel cycle de temps de P(1) Comme par hypothèse les cycles de temps des processeurs sont basés sur la même permutation , alors le cycle de temps de P(2) est une certaine permutation circulaire de rang r de celui de P(1) . Une des situations où P(1) est à son rythme le plus rapide est lorsqu'il a un cycle de temps unimodal c'est à dire :

$$d(1,i) = i \qquad 1 \leq i \leq n$$

Mais alors comme la phase d'initialisation de P(2) a été négligée son cycle de temps

est défini comme suit :

$$d(2,i) = \begin{cases} d(1,n-r+i) & \text{si } 1 \leq i \leq r \\ d(1,i-r) & \text{si } r+1 \leq i \leq n . \end{cases}$$

D'après cette remarque on peut donner la borne supérieure de la performance du réseau considéré .

Proposition :

$$T \leq \begin{cases} 3(n-1/n)/4 & \text{si n est impair} \\ 3n/4 - 1/n & \text{si n pair .} \end{cases}$$

Démonstration :

Soit R(r) le retard engendré par une permutation circulaire de rang r de la séquence (1 , 2 , 3 , . . . , n) . On montre que :

$$R(r) = \max (0 , \sum_{i=1}^{r} d(2,i) - \sum_{i=1}^{r+1} d(1,i)) .$$

En effet , au cours des r+1 premières tâches , P(1) a un rythme plus rapide que celui de P(2) alors que pendant les n-(r+1) dernières tâches la situation s'inverse c'est à dire que P(1) adopte un ryhme au moins aussi lent que celui de P(2) et donc ne saurait être retardé .
Considérons donc la grandeur

$$\sum_{i=2}^{r+1} d(2,i) - \sum_{i=1}^{r+1} d(1,i) .$$

(d(1, i) ; $1 \leq i \leq r+1$)) est la suite des r+1 premiers entiers naturels non nuls tandis que (d(2, i) ; $1 \leq i \leq r$)) est une suite arithmétique de raison -1 et de premier terme n-r+1 ; il vient

$$\sum_{i=1}^{r} d(2 , i) - \sum_{i=1}^{r+1} d(1 , i) = r(n-r+1+n)/2-(r+1)(r+2)/2$$

$$= -r^2+r(n-1)-1$$

Cette fonction atteint son maximum pour r = (n-1)/2 ; on prendra r = n/2 pour n pair . Il suit alors :

$$Rmax = \max_{r} R(r) = \begin{cases} \max (0 , (n^2-2n-3)/4) & \text{si n est impair} \\ \max (0 , (n^2-2n-4)/4) & \text{sinon .} \end{cases}$$

Par définition , la plus mauvaise performance pour un processeur est obtenue dès lors qu'il est retardé au maximum ; on en tire que

$$T \leq \begin{cases} 3n/4-1/n & \text{si n est pair} \\ 3(n-1/n)/4 & \text{sinon .} \end{cases}$$

Ceci termine la démonstration de la proposition .

Exemple : n = 12

P(1)	P(2)
d(1,1) = 1	d(2,1) = 7
d'(1,2) = 2 + 4	d(2,2) = 8
d'(1,3) = 3 + 5	d(2,3) = 9
d'(1,4) = 4 + 5	d(2,4) = 10
d'(1,5) = 5 + 5	d(2,5) = 11
d'(1,6) = 6 + 5	d(2,6) = 12
d'(1,7) = 7 + 5	d(2,7) = 1
d(1,8) = 8	d'(2,8) = 2 + 12
d(1,9) = 9	d'(2,9) = 3 + 6
d(1,10) = 10	d'(2,10) = 4 + 6
d(1,11) = 11	d'(2,11) = 5 + 6
d(1,12) = 12	d'(2,12) = 6 + 6
d(1,13) = 1	d(2,13) = 7
d(1,14) = 2 + 4	d(2,14) = 8
.	
.	
.	

Les situations aux dates 2 et 14 sont identiques ; donc T = 107/12 .

Remarque :

Les cycles de temps exhibés dans la démonstration ne sont pas uniques . On vérifie pour n = 2p que si P(1) et P(2) ont pour cycles de temps respectivement :

(p+1 , 1 , 2 , . . . , p , n , n-1 , n-2 , . . . , p+2) (n , n-1 , n-2 , . . . , p+1 , 1 , 2 , . . . , p)

la borne supérieure de T est aussi atteinte .

En effet en tenant compte de la phase d'initialisation de P(2) on vérifie aisément que
- aux étapes numérotées 2 , P(1) et P(2) commencent simultanément leurs tâches ,
- P(1) est retardé par P(2) respectivement de n-3 , n-4 , n-6 , n-8 , . . . , 4 , 2 unités de temps après les étapes numérotées 3 , . . . , p-1 , p , p+1 ,
- durant les étapes numérotées p+2 , p+3 , . . . , n , P(1) a un rythme plus lent que celui de P(2) ,
- les situations aux étapes 2 et n+2 sont identiques .

Ceci montre que si nous négligeons la phase d'initialisation de P(2) , alors le cycle de temps réel de P(1) correspond à la séquence périodique (d'(1,i) ; $i \geq 1$) de période

$$(1 , n-1 , n-1 , n-2 , \ldots , p+2 , n , n-1 , n-2 , \ldots , p+2 , p+1) .$$

Comme conséquence ,

$$T = (1+n-1+n-1+n-2+ \ldots + p+2+n+n-1+\ldots +p+1)/n$$
$$= 3n/4 - 1/n .$$

Notons enfin que dans le cas d'un réseau de plus de deux processeurs ,la situation est la même car P(2) étant retardé au maximum , on ne saurait trouver de cycle de temps de P(3) qui le retarde davantage .

Exemple : n = 12

P(1)		P(2)	
d(1,1)	= 7	-	(phase d'initialisation)
d((1,2)	= 1	d(2,2)	= 12
d'(1,3)	= 2 + 9	d(2,3)	= 11
d'(1,4)	= 3 + 8	d(2,4)	= 10
d'(1,5)	= 4 + 6	d(2,5)	= 9
d'(1,6)	= 5 + 4	d(2,6)	= 8
d'(1,7)	= 6 + 2	d(2,7)	= 7
d'(1,8)	= 12	d'(2,8)	= 1 + 4
d'(1,9)	= 11	d'(2,9)	= 2 + 9
d'(1,10)	= 10	d'(2,10)	= 3 + 7
d'(1,11)	= 9	d'(2,11)	= 4 + 5
d'(1,12)	= 8	d'(2,12)	= 5 + 3
d'(1,13)	= 7	d'(2,13)	= 6 + 1
d(1,14)	= 1	d(2,14)	= 12

.
.
.

Tout calcul fait on obtient T = 107/12 .

3 - Synchronisation sans retard

Dans cette section , nous nous intéressons aux permutations qui n'induisent pas de retard sur le cycle de temps d'un processeur ; c'est à dire que la durée moyenne de réalisation d'une tâche est de (n+1)/2 unités de temps .
Pour ce faire , notons que les retards générés dans le cycle de temps d'un processeur sont dus à un problème de blocage dans les communications inhérent à tout système parallèle dont les processeurs se synchronisent localement . Nous distinguons deux types de blocage .
- Le blocage en sortie : le processeur P(i) a fini sa tâche et attend que P(i+1) ait vidé son buffer (voir la situation de P(1) à l'instant t = 10 sur la figure 2) ; le retard engendré par ce blocage sera dit retard en sortie .
- Le blocage en entrée : P(i) attend que P(i-1) lui transmette ses résultats (voir la situation de P(2) à l'instant t = 6 sur la figure 2) ; le retard engendré par ce blocage sera dit retard en entrée .

Etant donné que notre réseau fonctionne selon un mécanisme de producteur-consommateur , il sera sans retard si et seulement si le processeur P(1) n'est jamais retardé en sortie . Comme P(2) dispose d'un buffer de profondeur 1 , P(1) n'est jamais retardé si et seulement si

$$\sum_{i=1}^{k} d(1,i) \geq \sum_{i=2}^{k-1} d(2,i) \qquad \text{pour tout } k .$$

Par hypothèse , (d(i,k) ; k ≥ i) est une séquence (s(j) ; 1 ≤ j ≤ n) où s est une permutation fixée et j peut varier d'un processeur à un autre alors la relation précédente devient :

$$\sum_{l=0}^{k} s(i+l) \geq \sum_{l=0}^{k-1} s(j+l) \qquad \text{pour tout } k \qquad (1)$$

où les indices sont calculés modulo n .

Remarque : si s = (s(i) ; 1 ≤ i ≤ n) est une solution de (1) alors la séquence

$$s' = (s(n-i+1) ; 1 \leq i \leq n)$$

l'est aussi .
Nous allons à présent expliciter la relation (1) . Pour cela , nous introduirons d'abord quelques notations et établirons quelques résultats préliminaires .

Sans affecter la généralité , nous considèrerons , sauf indication contraire que s est tel que s(1) = n . Ainsi toute permutation solution de (1) peut être étendue à la

séquence infinie (s(i) ; i ≥ 1) de période

$$(n , s(2) , s(3) , \ldots , s(i) , \ldots , s(n)) .$$

- Deux termes s(i) et s(i+1) sont dits voisins ; en particulier , n et s(n) sont voisins .
- Désignons par R(i,k) et L(i,k) la somme de k termes consécutifs de s respectivement dans l'ordre croissant et décroissant des indices à partir de la position i :

$$R(i,k) = \sum_{l=0}^{k-1} s(i+l)$$

$$L(i,k) = \sum_{l=0}^{k-1} s(i-l) .$$

Lemme 4.1 :

Si s = (n , s(1) , . . . , s(n)) est une permutation telle que R(i,2) ≥ R(1,1) = n et L(i,2) ≥ L(1,1) = n pour tout i , alors s est de la forme

$$(n , u(1) , v(1) , \ldots u(i) , v(i) , \ldots)$$

où v(i) = n - u(i) et u(i) < u(i+1) .

Démonstration :

Si s(i) = 1 alors R(i,2) = 1+s(i+1) et L(i,2) = 1+s(i-1) ; ainsi s(i-1) ≥ n et s(i+1) ≥ n .
On est dans l'une des deux situations suivantes :
1) s(i+1) = n-1 et s(i-1) = n ce qui implique que i = 1 càd

$$s = (n , 1 , n-1 , \ldots) \text{ donc } u(1) = 1 \text{ et } v(1) = n-1 ;$$

2) s(i+1) = n et s(i-1) = n-1 c'est à dire que s est de la forme

$$s = (n , \ldots , n-1 , 1) \text{ donc } u(j) = n-1 \text{ et } v(j) = 1$$

$$\text{où } j = \begin{cases} 2p & \text{si } n = 4p+1 \text{ ou } 4p+2 \\ 2p+1 & \text{si } n = 4p+3 \\ 2p-1 & \text{si } n = 4p . \end{cases}$$

Dans toute la suite , nous supposerons que nous sommes dans la première situation ; à cause de la symétrie , ceci ne restreint en rien la généralité .

Maintenant si s(i) = 2 alors :

R(i,2) = 2+s(i+1) $\geq$ n ce qui implique s(i+1) $\geq$ n-2
L(i,2) = 2+s(i-1) $\geq$ n ce qui implique s(i-1) $\geq$ n-2

c'est à dire {s(i-1) , s(i+1)} $\subset$ {n-2 , n-1 , n} ; en fait comme les voisins de 2 ne sauraient être n et n-1 , on a {s(i-1) , s(i+1)} inclus soit dans {n-2 , n-1} soit dans {n , n-2} . Il suit

$$s = (n , 1 , n-1 , \ldots , n-2 , 2) \quad \text{ou } s = (n , 1 , 2 , n-2 , \ldots , s(n)) .$$

Si s(i) = 3 alors comme précédemment , on a :

R(i,2) = 3+s(i+1) $\geq$ n d'où s(i+1) $\geq$ n-3
L(i,2) = 3+s(i-1) $\geq$ n d'où s(i-1) $\geq$ n-3

c'est à dire {s(i-1) , s(i+1)} $\subset$ {n-3 , n-2 , n-1 , n} . Comme deux éléments distincts ne peuvent pas avoir les mêmes voisins , {s(i-1) , s(i+1)} est inclus dans l'une des trois paires {n-3 , n-2 } , {n-3 , n-1} , {n-3 , n}... Selon que s = (n , 1 , n-1 , . . . , n-2 , 2) ou s = (n , 1 , n-1 , 2 , n-2 , . . .) , on se retouve dans l'une des quatre situations suivantes :

$$s = (n , 1 , n-1 , 3 , n-3 , \ldots , n-2 , 2)$$
$$s = (n , 1 , n-1 , \ldots , 3 , n-3 , n-2 , 2)$$
$$s = (n , 1 , n-1 , 2 , n-2 , 3 , n-3 , \ldots)$$
$$s = (n , 1 , n-1 , 2 , n-2 , \ldots , n-3 , 3) .$$

Clairement , il s'agit à chaque étape i , de placer le couple (i , n-i) à la position inoccupée la plus à gauche ou le couple (n-i , i) à la position inoccupée la plus à droite .
On montre ainsi de proche en proche que les éléments i et n-i pour i $\geq$ 1 sont toujours voisins d'où la structure suivante de s

$$s = (n , u(1) , v(1) , u(2) , v(2) , \ldots) \text{ où } v(i) = n-u(i) .$$

Il faut tout de même remarquer que si n = 2p , à la dernière étape du processus ci-dessus décrit , l'élément p sera le seul à placer car s est une permutation de n éléments distincts ; il représente donc le couple (p , n-p) .

Soit (i,j) tel que s(j) = y(i) .

$$R(i,2) = v(i) + u(i+1) \geq n$$

donc

$$u(i+1) \geq n-v(i) = u(i) .$$

Cette inégalité est en fait stricte car s est une permutation d'éléments distincts .

Ce lemme revêt une grande importance car outre la solution formelle , il indique la façon dont une solution peut être obtenue . En effet , par rapport à la position de s(1) = n le lemme dit qu'une solution peut être obtenue en plaçant successivement tantôt à gauche , tantôt à droite respectivement les couples (i , n-i) et (n-i , i) i = 1 , 2 , ...
s pourrait être réécrit sous la forme

$$s = (\ldots , y(-i) , x(-i) , \ldots , y(-1) , x(-1) , n , x(1) , y(1) , \ldots , x(i) , y(i) , \ldots)$$

où x(j) + y(j) = n , x(-i) < x(-(i+1)) et x(i) < x(i+1) .

On dira que n est la racine et que les couples (x(i) , y(i)) et (y(-i) , x(-i)) sont de rang i ou encore sont à une distance i de la racine .
Dans le cas où n est pair , en comparaison des autres termes regroupés par couple , n/2 peut être considéré comme une dégénérescence du couple (n/2 , n/2) ; aussi , l'appelerons nous couple dégénéré .

Lemme 4.2 :

Si s est une solution alors

$$x(\pm i) = \begin{cases} 2i \\ \\ 2i-1 \end{cases} \quad \text{pour tout } i \geq 1 .$$

Démonstration :

Au cours de cette démonstration , nous choisissons de représenter s sous la forme

$$s = (\ldots , y(-i) , x(-i) , \ldots , y(-1) , x(-1) , n , x(1) , y(1) , \ldots , x(i) , y(i) , \ldots)$$

et numérotons ses éléments de sorte que s(0) = n .
Rappelons que nous avons choisi x(1) = 1 . Nous commencerons donc par démontrer que x(-1) = 2 . Supposons x(-1) > 2 . Alors , d'après le processus de construction d'une solution décrit par le lemme 4.1 on a la situation suivante :

$$s = (\ldots y(-1) , x(-1) , n , 1 , n-1 , x(2) , y(2) , \ldots)$$

où x(2) = 2 . Mais alors :

$$R(-1,2) = x(-1) + n \text{ et } R(1,3) = n + 2$$

ce qui est une contradiction ; donc x(-1) ≤ 2 et nécessairement x(-1) = 2 . Par suite :

$$s = (\ldots, n-2, 2, n, 1, n-1, \ldots).$$

D'après la construction exhibée dans la démonstration du lemme 4.1 , le couple défini par la paire {3 , n-3} doit être placé à la gauche ou à la droite de la racine . Ainsi deux situations se présentent :

1) $s = (\ldots, y(-2), x(-2), n-2, 2, n, 1, n-1, 3, n-3, x(3), y(3), \ldots)$
2) $s = (\ldots, n-3, 3, n-2, 2, n, 1, n-1, x(2), y(2), \ldots)$.

Considérons le premier cas et supposons $x(-2) > 4$ alors d'après le lemme 4.1 $x(3) = 4$; ce qui génère une contradiction car :

$$R(-3, 4) = 2n + x(-2) \text{ et } L(5, 5) = 2n + 4 .$$

Dans la deuxième situation si $x(2) > 4$ on aboutit à la contradiction

$$R(-2, 3) = n + 3 \text{ et } L(2, 2) = n + x(2) - 1 > n + 3 .$$

Le lemme est donc vrai pour $i = 1$ et 2 . Supposons le vrai pour $i = k-1$ et montrons qu'il l'est pour $i = k$. Usant de la construction exhibée dans la démonstration du lemme 4.1 , on distingue deux situations :

1) $s = (\ldots, y(-k), x(-k), \ldots, n-2, 2, n, 1, n-1, x(k), y(k), x(k+1), y(k+1), \ldots)$

Si $x(-k) > 2k$ alors nécessairement $x(k+1) = 2k$ et on a la contradiction

$$R(-2k+1, 2k) = nk + x(-k) \text{ et } L(2k+1, 2k+1) = nk + 2k .$$

Il suit $x(-k) = 2k$.

2) $s = (\ldots n-2k+1, 2k-1, \ldots, n-2, 2, n, 1, n-1, \ldots, x(k), y(k), \ldots)$

dont on tire

$$L(2k-1, 2k-2) = n(k-1) + x(k) - 1 \text{ et } R(-2k+1, 2k-1) = n(k-1) + 2k - 1 .$$

La relation (1) implique $x(k) \leq 2k$; d'où $x(k) = 2k$.

Commentaire :

Ce lemme précise la nature des permutations qui sont solution de (1) . En effet , il indique que les couples $(y(-i), x(-i))$ et $(x(i), y(i))$ $i \geq 1$ équidistants de la racine n sont permutables : $x(-i)$ permute avec $x(i)$ et $y(-i)$ avec $y(i)$; nous dirons que ces couples sont duaux .

A propos de couples duaux , notons que :
- si n = 4p+1 , chaque couple possède un dual propre ,
- si n = 4p +3 , le couple de rang p+1 est son propre dual ; la permutation avec lui même correspond à une permutation de ses éléments ,
- si n = 2p , le couple dégénéré n/2 est son propre dual ou celui d'un couple non dégénéré selon que respectivement p est pair ou impair .

Exemples :

Dans ces exemples les nombre portés au bas de chaque "couple" indique son rang ou encore sa position par rapport à la racine n .

n = 9 : (5 , 4) (7 , 2) (9) (1 , 8) (3 , 6)
-2- -1- -1- -2-

n = 11 : (7 , 4) (9 , 2) (11) (1 , 10) (3 , 8) (5 , 6)
-2- -1- -1- -2- -3-

n = 10 : (6 , 4) (8 , 2) (10) (1 , 9) (3 , 7) (5)
-2- -1- -1- -2- -3-

n = 12 : (6) (8 , 4) (10 , 2) (12) (1 , 11) (3 , 9) (5 , 7) .
-3- -2- -1- -1- -2- -3-

Proposition 4.1 :

Pour tout n , la permutation

$$s^* = (\ldots, n-2i, 2i, \ldots, n-2, 2, n, 1, n-1, \ldots, 2i-1, n-2i+1, \ldots)$$

obtenue en plaçant systématiquement les couples (2i-1 , n-2i+1) à droite de la racine et le couple (n-2i , 2i) à gauche de la racine est une solution .

Avant de démontrer cette proposition , nous allons d'abord introduire quelques notations et établir des lemmes préliminaires .

Pour plus de clarté , nous représenterons souvent , chaque fois que besoin sera , une somme de termes de s* par une flêche reliant ses termes extrêmes ; la flêche sera orientée de la gauche vers la droite ou inversément selon que respectivement on calcule R(i,k) ou L(i,k) .

Lemme 4.3 :
Si n est impair , alors pour tout i on a :

$$n(k+1) \leq R(i,2k+1) \leq (n+2)k+1 \qquad (a)$$
$$nk \leq R(i,2k) \leq (n+2)k\,. \qquad (b)$$

Démonstration :

Considérons $s^* = (n\ ,\ 1\ ,\ n-1\ ,\ \ldots\ 2i-1\ ,\ n-2i+1\ ,\ \ldots\ .)$ et numérotons ses termes de sorte que $s^*(0) = n$.
Alors pour tout i

$$s^*(i) = \begin{cases} s^*(i \bmod n) & \text{si } i > n \\ i & \text{si i est impair} \\ n-i+1 & \text{si i est pair .} \end{cases}$$

Considérons la relation (a) .

a.1) On suppose que les sommes ne comportent pas n .

a.1.1) i est impair :

$$(s^*(i)\ ,\ n-s^*(i))\text{----------}(s^*(i+2k)\ ,\ n-s^*(i+2k))$$
$$\text{----------}2k+1\text{---------->}$$

$$R(i,2k+1) = nk+s^*(i+2k)$$

On a d'une part :

$$R(i,2k+1) = nk+2k+i$$
$$\geq nk+2k+1 \text{ car } i \geq 1$$

et d'autre part , comme la somme ne comporte pas n alors $s^*(i+2k) < n$ d'où :

$$R(i,2k+1) = nk+s^*(i+2k)$$
$$< n(k+1)\,.$$

a.1.2) i est pair :

$$(n-s^*(i)\ ,\ s^*(i))\text{----------}(n-s^*(i+2k)\ ,\ s^*(i+2k))$$
$$\text{----------}2k+1\text{---------->}$$

$$R(i,2k+1) = s^*(i)+nk$$
$$= n-i+1+nk$$

Ajoutons et retranchons 2k au second membre , on obtient :

$$R(i,2k+1) = nk+2k+1+n-(i+2k) .$$

Comme la somme ne comporte pas n alors $i+2k < n$ d'où

$$R(i,2k+1) > nk+2k+1.$$
$$R(i,2k+1) = n(k+1)-i+1$$

comme $i \geq 2$ alors $R(i,2k+1) < n(k+1)$.

a.2) On suppose que la somme comporte n .

a.2.1) i est impair :

$$(s^*(i) , n-s^*(i))\text{---------}(n)\text{----------}(n-s^*(i+2k) , s^*(i+2k))$$
$$\text{----------}2k+1\text{----------}>$$

$$R(i,2k+1) = nk+n$$

comme la somme porte sur n termes au plus alors $2k+1 \leq n$; par suite

$$R(i,2k+1) = n+2k+1+n-(2k+1)$$
$$\geq nk+2k+1 .$$

a.2.2) i est pair :

$$(n-s^*(i) , s^*(i))\text{----------}(n)\text{--------}(s^*(i+2k) , n-s^*(i+2k))$$
$$\text{----------}2k+1\text{----------}>$$

$$R(i,2k+1) = n(k+1)-i+1+(i+2k) \bmod n$$
$$= n(k+1)-i+1+i+2k-n$$
$$= nk+2k+1 .$$

Comme la somme porte sur moins de n termes alors $2k+1 < n$ d'où

$$R(i,2k+1) < (n+1)k .$$

Ceci termine la démonstration de la relation (a) .

Examinons la relation (b) .

b.1) La somme ne comporte pas n .

b.1.1) i est impair :

(s*(i) , n-s*(i))----------(n-s*(i+2k-1) , s*(i+2k-1))
----------2k---------->

$$R(i,2k) = nk .$$

b.1.2) i est pair :

(n-s*(i) , s*(i))----------(s*(i+2k-1) , n-s*(i+2k-1))
----------2k---------->

$$R(i,2k) = n-i+1+n(k-1)+i+2k-1$$
$$= nk+2k .$$

b.2) La somme comporte n .

b.2.1) i est impair :

(s*(i) , n-s*(i))----------(n)----------(s*(i+2k-1) , n-s*(i+2k-1))
----------2k---------->

$$R(i,2k) = n(k-1)+n+(i+2k-1) \bmod n$$
$$= nk+2k+i-(n+1) .$$

On a d'une part $i \leq n$ d'où $R(i,2k) < (n+2)k$ et d'autre part comme la somme traverse n alors $i+2k-1 \geq n$ et par suite $R(i,2k) \geq nk$.

b.2.2) i est pair :

(n-s*(i) , s*(i))----------(n)---------(n-s*(i+2k-1) , s*(i+2k-1))
----------2k---------->

$$R(i,2k) = n-i+1+n+n(k-1)$$
$$= n(k+1)-i+1 ;$$

comme $i \leq n-1$ alors $R(i,2k) \geq nk$.
En ajoutant et retranchant 2k au second membre , on obtient :

$$R(i,2k) = nk+2k+n-(i+2k-1) .$$

Comme la somme traverse n alors $i+2k-1 > n$ d'où $R(i,2k) < nk+2k$.

Ceci termine la démonstration de la relation (b) et du lemme .

On démontre un lemme identique pour le cas où n est pair .

Lemme 4.4 : Si n est pair et si R(i,l) est tel que $l \leq n/2$ alors :

$$(n+2)k+1 \leq R(i,2k+1) \leq n(k+1) \quad (a)$$
$$nk \leq R(i,2k) \leq (n+2)k . \quad (b)$$

Démonstration :

Nous utiliserons la même technique que pour le cas n impair .
Commençons par faire remarquer que s* ayant la même structure , toutes les sommes qui ne font pas intervenir n/2 restent égales à celles obtenues pour le cas où n est impair . Aussi , il ne nous reste qu'à examiner les cas qui font intervenir le couple dégénéré (n/2) .
Rappelons aussi que s* est de la forme

$$s^* = (1 , n-1 , 3 , n-3 , \ldots , n/2 , \ldots , n-4 , 4 , n-2 , 2 , n) .$$

$$s^*(i) = \begin{cases} i & \text{si i est impair et } s^*(i) < n/2 \text{ ou i est pair et } s^*(i) > n/2 \\ n/2 & \text{si } i = n/2 \text{ et } n = 4p+2 \text{ ou } i = n/2+1 \text{ et } n = 4p . \end{cases}$$

Considérons la relation (a) .

a.1) i est impair et $s^*(i) < n/2$:

$$(s^*(i) , n-s^*(i))\text{----------}(n/2)$$
$$\text{----------}2k+1\text{---------->}$$

$$R(i,2k+1) = nk+n/2$$

avec $2k+1 \leq n/2$ d'où :

$$n(k+1) < R(i,2k+1) \leq n+2k+1 .$$

Ce résultat est aussi celui du cas suivant :

$$(s^*(i) , n-s^*(i))\text{-------}(n/2)\text{-------}(n-s^*(i+2k) , s^*(i+2k))$$
$$\text{----------}2k+1\text{---------->}$$

a.2) i est pair et $s^*(i) < n/2$:

(n-s*(i) , s*(i))-------(n/2)-------(s*(i+2k) , n-s*(i+2k))
----------2k+1---------->

$$R(i,2k) = nk+2k+n/2+1 .$$

Comme la somme porte sur un nombre de termes au plus égal à n/2 alors :

$$nk+2k+1 \leq R(i,2k+1) \leq (n+1)k .$$

Ceci termine la démonstration de la relation (a) .

a.3) $s^*(i) = n/2$:

(n/2)----------(n-s*(i+2k) , s*(i+2k))
----------2k+1---------->

Le résultat est identique à celui du cas a.1) .
Ceci termine la démonstration de la relation (a) .

Examinons la relation (b) .

b.1) i est impair et $s^*(i) < n/2$:

(s*(i) , n-s*(i))-------(n/2)-------(s*(i+2k-1) , n-s*(i+2k-1))
----------2k---------->

$$R(i,2k) = n(k-1)+n/2+i+2k-1 .$$

Comme la somme traverse n/2 alors $i+2k-1 > n/2$ d'où $R(i,2k) \geq nk$.
Pour la majoration , il suffit de remarquer que :

$$R(i,2k) = nk+2k+i-(n/2+1) .$$

Comme $i \leq n/2$, il suit $R(i,2k) < nk+2k$.

b.2) i est pair et $s^*(i) < n/2$:

(n-s*(i) , s*(i))-------(n/2)-------(n-s*(i+2k-1) , s*(i+2k-1))
----------2k---------->

$$R(i,2k) = nk+n/2-i+1 .$$

D'une part $R(i,2k) = nk+(n/2+1)-i$; comme $i+1 < n/2$ alors $R(i,2k) > nk$.
D'autre part comme la somme traverse n/2 $i+2k-1 > n/2$ alors :

$$R(i,2k) = nk+2k+n/2-(i+2k-1)$$
$$< nk+2k .$$

b.3) $s^*(i) = n/2$:

$$(n/2)\text{----------}(s^*(i+2k-1)\ ,\ n-s^*(i+2k-1))$$
$$\text{----------}2k\text{---------->}$$

$$R(i,2k) = nk-n/2+i+2k-1 .$$

R(i,2k) est minoré par nk car , comme la somme traverse n/2 , $i+2k-1 > n/2$.
R(i,2k) est majoré par nk+2k parce que :
$R(i,2k) = nk+2k+i-(n/2+1)$ où $i = n/2$ ou $n/2+1$ selon que respectivement $n = 4p+2$ ou $n = 4p$.
Ceci termine la démonstration du lemme .

Lemme 4.5 : Si pour tout (i,j,k) tel que $l \leq n/2$ on a la relation

$$R(i\ ,\ l) \geq R(i\ ,\ l-1)$$

alors , elle reste vraie pour tout $l \geq n/2$.

Démonstration :

Posons $R = n(n+1)/2$.
Dire que $R(i,l-1) \leq R(j,l)$ pour tout (i,j,l) tel que $l \leq n/2$ signifie que

$$\max_i R(i,l-1) \leq \min_j R(j,l)$$

par suite

$$R\text{-}\min_j R(j,l) \leq R\text{-}\max_i R(i,l-1) \qquad (*)$$

or pour tout i on a :

$$R(i,n-l) = R-R(i+n-l,l)$$
$$R(i+n-l,l) \geq \min_j Rj,l) .$$

Il suit

$$R-R(i+n-l,l) \leq R\text{-}\min_j R(j,l)$$

et

$$R(i,n-l) \leq \underset{j}{\text{R-min}}\, R(j,l)$$

c'est à dire

$$\max_{i} R(i,n-l) = \underset{j}{\text{R-min}}\, R(j,l)$$

La relation $(*)$ devient

$$\max_{j} R(j,n-l) \leq \min_{i} R(i,n-l+1)$$

c'est à dire , en posant n-l = m , pour tout (i,j,m) tel $m \geq n/2$ on a :

$$R(j,m) \leq R(i,m+1) \ .$$

Avec ce lemme la proposition 4.1 devient évidente .

Dans toute la suite s* sera dite solution ou permutation standard .

Il est évident que toute autre solution , en vertu du lemme 4.2 , est obtenue de la solution standard par permutation de couples duaux de rang supérieur à 1 .
Dans les paragraphes à venir , nous étudions les propriétés des permutations susceptibles de générer d'autres solutions .

Lemme 4.6 :

Soit s une solution obtenue de s* par permutation de couples duaux . Soit Λ , un sous-ensemble de $\{1 , 2 , 3 , \ldots , n\}$, l'ensemble des rangs des couples permutés et k son plus petit élément .
A est l'ensemble de tous les multiples de k .

Démonstration :

Nous montrerons d'abord par récurrence que tout couple de rang multiple de k est permuté avec son dual par rapport à s* .
Pour ce faire rappelons que par rapport à s* , s est de la forme

$$s = (y(-m),x(-m),\ldots,y(-i),x(-i),\ldots,n-2,2,n,1,n-1,\ldots,x(i),y(i),\ldots,x(m),y(m))$$

$$m = \begin{cases} p & \text{si } n = 4p+1 \\ p+1 & \text{si } n = 4p \text{ ou } 4p+2 \text{ ou } 4p+3 , \end{cases}$$

$x(j)+y(j) = n$ et $x(i) = 2i$ ou $2i-1$ selon que le couple auquel il appartient a été respectivement permuté avec son dual ou pas .
Rappelons aussi que :
- Si $n = 4p+3$, les couples de rang $m = p+1$ sont confondus et constituent le couple qui est son propre dual ; aussi aura -t-on $x(-m) = 2m-1$ ou $2m$ selon que respectivement il est permuté ou non .
- Si $n = 4p+2$, les couples de rang $m = p+1$ sont dégénérés et confondus ; l'élément $x(-m)$ auquel ils sont réduits est son propre dual et a pour valeur $2m-1$.
- Si $n = 4p$ l'un des couples de rang $m = p$ est dégénéré .
Considérons les sommes

$$L(-2k,2k) = y(-k)+n(k-1)+x(-2k)$$
$$R(-2k+1,2k+1) = x(-k)+n(k-1)+n+1 .$$

Par hypothèse , le couple de rang k est permuté donc $x(-k) = 2k-1$ et $y(-k) = n-2k+1$; il suit :

$$R(-2k+1,2k+1) = nk+2k$$
$$L(-2k,2k) = nk-2k+x(-2k)+1 .$$

s étant une solution , alors $R(-2k+1,2k+1) \geq L(-2k,2k)$ c'est à dire $x(-2k) \leq 4k-1$.
Or d'après le lemme 4.2 , $x(-2k) = 4k-1$ ou $4k$; nécessairement , $x(-2k) = 4k-1$. Les couples de rang $2k$ sont permutés .
Supposons que la propriété est vraie pour le couple de rang $i = lk$ et montrons qu'elle l'est pour le couple de rang $i = (l+1)k$.
Il suffit de comparer $R(-2k+1,2k+1)$ avec $L(-2lk,2k)$.

$$L(-2lk,2k) = y(-lk)+n(k-1)+x(-(l+1)k) .$$

Comme , la propriété est vraie pour les couples de rang lk alors

$$y(-lk) = n-2lk+1$$
$$L(-2lk,2k) = nk-2k+x(-(l+1)k)+1 .$$

s étant une solution alors $R(-2k+1,2k+1) \geq L(-2lk,2k)$ c'est à dire que

$$x(-(l+1)k) \leq 2(l+1)k-1 .$$

Deux situations se présentent :

1) $(l+1)k \neq m$ ou $(l+1)k = m$ et $n = 4p+3$ ou $4p+2$ ou $4p+1$: c'est le cas où les couples de rang $(l+1)k$ sont non dégénérés ou possèdent des duaux propres .D'après le lemme 4.2 , $x(-(l+1)k) = 2(l+1)k-1$ ou $2(l+1)k$; nécessairement

$$x(-(l+1)k) = 2(l+1)k-1;$$

c'est à dire que les couples de rang $(l+1)k$ sont permutés .
2) $n = 4p$ et $(l+1)k = m$: c' est le cas où le couple de rang m est le couple dégénéré . Il n'y a aucune ambiguité car $x(-(l+1)k) = 2(l+1)k-1$.

Il reste à montrer que tout élément de A est multiple de k .
Soit k' un élément de A ; on a la situation suivante :

$$s = (\ldots , y(-k') , x(-k') , \ldots , y(-k) , x(-k) , \ldots , n-2 , 2 , n , 1 , n-1 , \ldots)$$

où $x(k') = 2k'-1$.

Considérons la somme

$$\begin{aligned} R(-2k'+1,2k) &= x(-k')+n(k-1)+y(-(k'-k)) \\ &= n(k-1)+2k'-1+y(-(k'-k)) . \end{aligned}$$

Comme s est une solution alors , $R(-2k+1,2k+1) \geq R(-2k'+1,2k)$ d'où :

$$y(-(k'-k)) \leq n-2(k'-k)+1 ;$$

ce qui signifie que le couple de rang $(k'-k)$ est permuté avec son dual .
Identiquement , à partir du couple de rang $(k'-k)$, on montre que le couple de rang $k'-2k$ est permuté avec son dual . Ainsi de proche en proche , on montre que les couples de rang $k'-lk$ sont permutés avec leurs duaux .
Soit donc r le maximum de tous les l tel que les couples de rang $k'-kl$ sont permutés avec leurs duaux , alors nécessairement $k'-rk \leq k$. k étant le plus petit élément de Λ , on a plutôt $k'-rk = k$ et donc k' est un multiple de k .

Commentaire :

Ce lemme caractérise toute solution non standard s comme une séquence obtenue de s* par permutation des seuls couples duaux placés à une distance multiple de k de la racine n . Nous dirons que s est une k-permutation .
Une conséquence de la première partie de la démonstration est que pour n impair , si r est le rang le plus élevé des couples permutés , les couples de rang r sont à une distance k l'un de l'autre sauf dans le cas particulier $n = 4p+3$ et $r = p+1$ où le couple de rang r est à une distance k des couples de rang $r-k$.
En effet , si tel n'était pas le cas , k ne serait pas le plus petit des rangs des couples permutés .
Clairement , dans la séquence infinie de période s , deux couples permutés et consécutifs sont à une distance k l'un de l'autre d'où :
a) si $n = 4p+1$ ou $n = 4p+3$ et $r \neq p+1$, on a :

$$s = (n\ ,\ 1\ ,\ n\text{-}1\ ,\ \ldots\ ,\ x(r)\ ,\ y(r)\ ,\ \ldots\ ,\ y(\text{-}r)\ ,\ x(\text{-}r)\ ,\ \ldots\ ,\ n\text{-}2\ ,\ 2)$$
$$\text{<--------2k----->}$$

par suite

$$n = 4r+2(k\text{-}1)+1 \text{ où } r = lk$$
$$= (4l+2)k\text{-}1\ .$$

a.1) si $n = 4p+1$, il vient $2p+1 = (2l+1)k$ c'est à dire k est impair ; donc $k \geq 3$ car $k > 1$.

a.2) si $n = 4p+3$, il vient $2(p+1) = (2l+1)k$ c'est à dire k est pair , différent de $p+1$ et diviseur de $2(p+1)$; en particulier $k = 2$.

b) si $n = 4p+3$ et $r = p+1$, on a :

$$s = (\ \ldots\ ,\ x(r\text{-}k)\ ,\ y(r\text{-}k)\ ,\ \ldots\ ,\ 2p+2\ ,\ 2p+1\ ,\ \ldots\ ,\ y(\text{-}(r\text{-}k))\ ,\ x(\text{-}(r\text{-}k))\ ,\ \ldots\ )$$
$$\text{<---------2k------> <---------2k-------->}$$

par suite $n = 2r+2(k\text{-}1)+2(r\text{-}k)+1$ avec $r = lk$ d'où $p+1 = lk$; k est donc un diviseur de $p+1$ et en particulier $k = p+1$.

A l' issue de ce commentaire , nous pouvons répondre à la question de savoir si toute k-permutation est une solution .

Proposition 4.2 : Pour tout $n = 4p+1$, s^* est l'unique solution .

Démonstration :

Elle est une conséquence du commentaire . En effet , d'après le commetaire , si s est une k-permutation alors $k \geq 3$. Considérons alors les sommes $R(\text{-}2k+1,2k+3)$ et $L(\text{-}2k,2k+2)$.

$$s = (\ \ldots\ ,\ n\text{-}z\ ,\ z\ ,\ \ldots\ ,\ y(\text{-}k)\ ,\ x(\text{-}k)\ ,\ \ldots\ ,\ n\text{-}2\ ,\ 2\ ,\ n\ ,\ 1\ ,\ n\text{-}1\ ,\ 3\ ,\ n\text{-}3\ ,\ \ldots\ )$$
$$\text{<----2k+2--- ----------2k+3------->}$$

$$R(\text{-}2k+1,2k+3) = x(\text{-}k)+nk+n+3 \quad \text{avec } x(\text{-}k) = 2k\text{-}1$$
$$= n(k+1)+2k+2$$
$$L(\text{-}2k,2k+2) = y(\text{-}k)+nk+z$$
$$= n(k+1)\text{-}2k+1+z$$

où z appartient à un couple permuté et

$$z = \begin{cases} x(\text{-}2k+1)) = 4k+2 & \text{si la somme ne traverse pas le couple de rang p} \\ y((n\text{-}1\text{-}4k)/2) = 4k+2 & \text{sinon .} \end{cases}$$

Démonstration :

C'est une conséquence de la démonstration de la proposition 4.2 .
En effet si $k \geq 3$ et si la somme $L(-2k,2k+2)$ ne traverse pas le couple de rang p , la contradiction $R(-2k+1,2k+3) < L(-2k,2k+2)$ subsiste ; nécessairement $k = 2$ ou k est tel que $L(-2k,2k+2)$ doit traverser le couple de rang p c'est à dire :

$$2(k+1) > (2p-1)-2k+1$$
$$k > (2p+2)/4$$

qu'on peut écrire $k > (n/2+2)/4$ ou $k > (n/2+1)/4$ selon que respectivement $n = 4p$ ou $n = 4p+2$.

A l'issue de cette étude , qui donne les conditions nécessaires pour qu'une k-permutation soit une solution , on est à même de caractériser les solutions obtenues de s* . En fait , nous vérifierons que les valeurs de k précédemment exhibées définissent bien des solutions . Pour ce faire , en vertu du lemme 4.4 , nous ne considérerons , sauf indication contraire , dans les démonstrations à venir que les sommes d'au plus n/2 termes .

Lemme 4.9 :

Soit s une k-permutation . Pour tout n et k tels que $n = 4p+3$ et $k = p+1$ on a :

1) si $2l+1 \leq 2p+1$ alors $nl+2l+1 \leq R(i,2l+1) \leq n(l+1)$ et $nl \leq R(i,2l) \leq nl+2l+1$
2) si $2l+1 > 2p+1$ alors $nl+2l \leq R(i,2l+1) \leq n(l+1)$ et $nl \leq R(i,2l) \leq nl+2l$.

Démonstration :

Rappelons que s est obtenue de s* en permutant les éléments du couple de rang p+1 ; en numérotant les termes de s tel que $s(0) = n$ on a :

$$s = (n , 1 , n-1 , \ldots , \mathbf{2p+2} , \mathbf{2p+1} , \ldots , n-2 , 2)$$

$$s(i) = \begin{cases} s(i \bmod n) & \text{si } i \geq n \\ i & \text{si i est impair et différent de } 2p+1 \\ 2p+2 & \text{si } i = 2p+1 \\ n-i+1 & \text{si i est pair et différent de } 2p+2 . \end{cases}$$

Par rapport au lemme 4.3 , nous n'aurons à examiner que les sommes qui font intervenir isolément les éléments du couple permuté (2p+2 , 2p+1) . Il faut donc rajouter à la démonstration du lemme 4.3 les cas suivants .

1) On suppose $2l+1 \leq 2p+1$:

Si i est impair , on a :

(s(i) n-s(i))----------->**(2p+2 2p+1)**
----------2l+1---------->

$$R(i,2l+1) = nl+2p+2 > nl+2l+1 \text{ car } 2l+1 \leq 2p+1$$

$$R(i,2l+1) = nl+2p+2 < nl+n .$$

Si i est pair on a :

(n-s(i) s(i))----------**(2p+2 2p+1)**
----------2l---------->

$R(i,2l) = n-i+1+n(l-1)+2p+2$ avec $2l = 2p+1-i+1$ d'où $R(i,2l) = nl+2l+1$.

Si $i = 2p+2$, on distingue les situations suivantes :

(2p+2 2p+1)----------(n-s(i+2l) s(i+2l))
----------2l+1---------->

$$R(2p+2,2l+1) = 2p+1+nl \geq nl+2l+1 \text{ car } 2l+1 \leq 2p+1$$
$$R(2p+2,2l+1) < nl+n .$$

(2p+2 2p+1)----------(s(i+2l-1) n-s(i+2l-1))
----------2l---------->

$$R(i,2l) = 2p+1+n(l-1)+i+2l-1 \text{ où } i=2p+2 = nl+2l-1 ;$$

ceci termine la démonstration de la première partie du lemme .
Pour la deuxième partie on distingue les cas où $i = 2p+2$:

(2p+2 2p+1)----------(n)---------(s(i+2l) n-s(i+2l))
----------2l+1---------->

$$R(2p+2,2l+1) = 2p+1+n+s(i+2l)+n(l-1)$$

mais $s(i+2l) = 2p+2+2l-n$; d'où $R(2p+2,2l+1) = nl+2l$.

(2p+2 2p+1)----------(n)---------(n-s(i+2l-1) s(i+2l-1))
----------2l---------->

$$R(2p+2,2l) = 2p+1+n+n(l-1)$$
$$= nl+2p+1$$
$$< nl+2l \quad \text{car } 2l > 2p+1 .$$

Lemme 4.10 :

Soit s une k-permutation . Si n est pair alors pour tout (k,l) tel que $2l+1 \leq n/2$ et

$$k > \begin{cases} (n/2+1)/4 & \text{si } n = 4p \\ (n/2+2)/4 & \text{si } n = 4p+2 \end{cases}$$

on a :

1) Si $2l+1 \leq 2k-1$ alors :

$$nl+2l+1 \leq R(i,2l+1) \leq n(l+1)$$
$$nl \leq R(i,2l) \leq nl+2l+1$$

2) Si $2k \leq 2l \leq n/2$ alors :

$$nl+2l \leq R(i,2l+1) \leq n(l+1)$$
$$nl \leq R(i,2l) \leq nl+2l .$$

Démonstration :

Considérons la séquence s obtenue de s* par permutation des couples duaux de rang k .

s = (n,1,n-1, . . . , **2k,n-2k**, . . . , n/2 , . . . , **n-2k+1,2k-1**, . . . , n-2,2)

où les termes en gras sont les éléments des couples permutés et numérotons les termes de s tel que s(0) = n . Clairement , pour tout i tel que s(i) n'appartient pas à un couple permuté on a :

$$s(i) = \begin{cases} s(i \bmod n) & \text{si } i \geq n \\ i & \text{si } 1 \leq i \leq j \text{ et impair ou } j \leq i \leq n \text{ et pair} \\ n-i+1 & \text{sinon} \end{cases}$$

où j désigne la position du couple dégénéré n/2 .

Par rapport au lemme 4.4 , il nous reste à examiner les sommes faisant intervenir

isolément les éléments des couples permutés .

Considérons la première partie du lemme .
Si i est impair et la somme ne traverse pas n/2 , on distingue les cas suivants :

(s(i) n-s(i))----------(**2k n-2k**)
----------2l+1---------->

R(i,2l+1) = nl+2k ; comme par hypothèse 2l+1 ≤ 2k-1 et 2k < n on a :
nl+2k+1 < R(i,2l+1) < n(l+1) .

Si i = 2k et la somme ne traverse pas n/2 on a :

(**2k n-2k**)----------(n-s(i+2l) s(i+2l))
----------2l+1---------->

R(i,2l+1) = n-2k+nl .

Ajoutons et retranchons 2l+1 au second membre ; on obtient :

R(i,2l+1) = nl+2l+1+n-(2l+2k+1) .

Mais 2l+2k+1 ≤ 4k d'où R(i,2l+1) ≥ nl+2l+1+n-4k .

Etant donné l'hypothèse faite sur k on a R(i,2l+1) > nl+2l+1 .

D'autre part R(i,2l+1) = nl+n-2k < nl+n .

Si i = 2k et la somme traverse n/2 on a :

R(i,2l+1) = n-2k+n/2+i+2l+n(l-1)
= nl+2l+n/2 .

Comme 2l < n/2 on obtient la relation désirée .

(**2k n-2k**)---------(n/2)----------(**n-2k+1 2k-1**)
----------2l+1---------->

R(2k,2l+1) = n-2k+n/2+n-2k+1+n(l-1)
= nl+n-(4k-1)+n/2 avec 2l+1 = n-4k+1
= nl+2l+1+n/2 ;

comme 2l+1 ≤ n/2 , on a la relation désirée .

Supposons maintenant qu'on est placé en i >j et pair ; on a :

(s(i) n-s(i))----------**(n-2k+1 2k-1)**
----------2l+1---------->

$$R(i,2l+1) = nl+n-2k+1 < nl+n$$

Par ailleurs comme $k \leq n/4$ on a :

$$\begin{aligned} R(i,2l+1) &\geq nl+n-n/2+1 \\ &\geq nl+n/2+1 \\ &> nl+2l+2 . \end{aligned}$$

Si i est tel que s(i) = 2k-1 on a :

(n-2k+1 2k-1)----------(n-s(i+2l) s(i+2l))
----------2l+1---------->

$$R(i,2l+1) = 2k-1+nl .$$

Par hypothèse $2l+1 \leq 2k-1 \leq n/2$ d'où :

$$nl+2l+1 \leq R(i,2l+1) \leq nl+n/2 < n(l+1) .$$

Ceci termine la démonstration de la première relation de la première partie du lemme .
Considérons les sommes qui ne traversent pas n/2 .

(n-s(i) s(i))----------**(2k n-2k)**
----------2l---------- >

$$\begin{aligned} R(i,2l) &= n-i+1+2k+n(l-1) \quad \text{où } 2k-1-i+1 = 2l \\ &= nl+2l+1 . \end{aligned}$$

Si i = 2k on a :

(2k n-2k)----------(s(i+2l-1) n-s(i+2l-1))
----------2l---------->

$$\begin{aligned} R(2k,2l) &= n-2k+i+2l-1+n(l-1) \\ &= nl+2l-1 . \end{aligned}$$

Si i = n-2k+1 on a :

(n-2k+1 2k-1)----------(s(i+2l) n-s(i+2l))
----------2l---------->

$$R(i,2l) = 2k-1+i+2l-1+n(l-1)+1$$
$$= nl+2l .$$

Considérons les sommes qui traversent n/2 .

Si i = 2k on a :

(2k n-2k)----------(n/2)---------(n-s(i+2l-1) s(i+2l-1))
----------2l---------->

$$R(i,2l) = n-2k+n/2+n(l-1) \text{ où par hypothèse } k \leq n/4$$
$$> nl .$$

Par ailleurs , en ajoutant et retranchant 2l+1 du second membre on a :

$$R(i,2l) = nl+2l+1+n/2-(2k+2l+1) .$$

Comme la somme traverse n/2 alors 2l+2k > n/2 d'où R(i,2l) < nl+2l+1 .

Pour terminer la démonstration de la deuxième relation de la première partie du lemme , il nous reste à examiner le cas suivant :

(2k n-2k)---------(n/2)----------**(n-2k+1 2k-1)**
----------2l---------->

$$R(2k-1, 2l) = n/2+n-2k+1+n(l-1)$$
$$= nl+n/2-2k+1 .$$

Comme 2k < n/2 alors R(2k-1, 2l) > nl .
Ajoutons et retranchons 2l au second membre ; il vient :

$$R(2k-1,2l) = nl+2l+1+n/2-(2l+2k) ;$$

et comme la somme traverse n/2 on a 2l+2k ≥ n/2 ; d'où R(2k-1,2l) ≤ nl+2l+1 .

Ceci termine la démonstration de la première partie du lemme .

La deuxième partie se démontre en considérant essentiellement des sommes qui contiennent n .
Pour la première relation on distingue les cas suivants où i = n-2(k-1) :

(n-2k+1 2k-1)----------(n)----------(s(i+2l) n-s(i+2l))
----------2l+1---------->

$$R(i,2l+1) = 2k-1+n+n(l-1)+i+2l-n$$
$$= nl+2l .$$

(n-2k+1 2k-1)----------(n)----------**(2k n-2k)**
----------2l+1---------->

$$R(i,2l+1) = 2k-1+n+2k+n(l-1)$$
$$= nl+4k-1 .$$

On a d'une part $4k-1 < n$ d'où $R(i,2l+1) < n(l+1)$.
D'autre part on a :

$$R(i,2l+1) = nl+2l+4k-2l .$$

Or $k > (n/2+e)/4$ avec $e = 1$ ou 2 selon que $n = 4p$ ou $n = 4p+2$; d'où

$$4k > n/2+e > 2l$$
$$R(i,2l+1) > nl+2l .$$

Pour la deuxième relation , on distingue les cas suivants :

(n-2k+1 2k-1)----------(n)----------**(2k n-2k)**
----------2l---------->

$$R(i,2l) = nl+2k > nl .$$

Par hypothèse , $2k < 2l$ et par suite $R(i,2l) < nl+2l$.

<u>Lemme 4.11</u> :

Soit s une k-permutation obtenue de s* . Pour tout (l,n,k) tel que $n \neq 4p+1$, $k = 2$, $2l+1 \leq n/2$, on a :

1) $$n(l+1) \geq R(i,2l+1) \geq \begin{cases} nl+2l+1 & \text{si l est impair} \\ nl+2l & \text{si l est pair .} \end{cases}$$

2) $$nl \leq R(i,2l) \leq \begin{cases} nl+2l & \text{si l est pair} \\ nl+2l+1 & \text{si l est impair .} \end{cases}$$

Démonstration :

Considérons la séquence s obtenue de s* en permutant les couples de rang pair avec leurs duaux et numérotons ses termes de sorte que s(0) = n .

s = (n , 1 , n-1 , **4 , n-4** , 5 ,n-5 , . . . , n-6 , 6 , **n-3 , 3** , n-2 , 2)

où les termes en gras sont les éléments des couples permutés .

Supposons n = 4p+1 ; alors s est tel que :

$$s(i) = \begin{cases} s(i \bmod n) & \text{si } i \geq n \\ i & \text{si i est impair et n'appartient pas à un couple permuté} \\ i+1 & \text{si i est impair et appartient à un couple permuté} \\ n-s(i-1) & \text{si i est pair .} \end{cases}$$

Comme dans les lemmes précédents , nous ne nous intéresserons qu'aux sommes qui font intervenir isolément des éléments de couples permutés .

Supposons que la somme est calculée sur un nombre impair de termes .

1) 2l+1 est de la forme 4r+1 :

1.a) Si la somme ne contient pas n on a :

(s(i) n-s(i))----------(s(i+2l) n-s(i+2l))
----------2l+1---------->

R(i,2l+1) = nl+i+2l+1 > nl+2l+1.

Comme s(i+2l) < n on a R(i,2l+1) < n(l+1) .

. **(n-s(i) s(i))----------(n-s(i+2l) s(i+2l))**
----------2l+1---------->

R(i,2l+1) = n-i+nl < n(l+1) .

Ajoutons et retranchons 2l+1 du second membre :

R(i,2l+1) = nl+2l+1+n-(i+2l+1) ;

comme la somme ne traverse pas n alors n-(i+2l+1) > 0 d'où R(i,2l+1) > nl+2l+1 .

1.b) Si la somme contient n on distingue les cas suivants :

(n-s(i) s(i))----------(n)---------(s(i+2l) n-s(i+2l))
----------2l+1---------->

R(i,2l+1) = n-i+1+n+i+2l-n+1+n(l-1)
= nl+2l+2 .

(n-s(i) s(i))----------(n)----------(s(i+2l) n-s(i+2l))
----------2l+1---------->

R(i,2l+1) = n-i+n+i+2l-n+n(l-1)
= nl+2l .

2) 2l+1 est de la forme 4r+3 :

2.a) Si la somme ne contient pas n , on a :

(n-s(i) s(i))----------(n-s(i+2l) s(i+2l))
----------2l+1---------->

d'une part :

R(i,2l+1) = n-i+1+nl où i > 1
< n(l+1) .

D'autre part en ajoutant et retranchant 2l au second membre il vient :

R(i,2l+1) = nl+2l+1+n(i+2l) .

Comme la somme ne traverse pas n alors i+2l < n d'où R(i,2l+1) > nl+2l+1 .

(s(i) n-s(i))----------(s(i+2l) n-s(i+2l))
----------2l+1---------->

R(i,2l+1) = i+2l+1+nl < n(l+1)

car la somme ne traverse pas n .

Supposons à présent que la somme porte sur un nombre pair de termes .

1) Le nombre de termes est de la forme 4r :

1.a) Si la somme ne contient pas n on a :

(n-s(i) s(i))----------(s(i+2l-1) n-s(i+2l-1))
----------2l---------->

$R(i,2l) = n-i+n(l-1)+i+2l-1+1 = nl+2l$.

1.b) Si la somme contient n on a :

(n-s(i) s(i))----------(n)
----------2l--------->

$R(i,2l) = n-i+n+n(l-1)$
$= nl+n-i$ où $n-i+1 = 2l$
$= nl+2l-1$.

2) Le nombre de termes est de la forme 4r+2 :

2.a) Si la somme ne contient pas n on distingue les cas suivants :

(n-s(i) s(i))----------**(s(i+2l-1) n-s(i+2l-1))**
----------2l---------->

$R(i,2l) = n-i+1+n(l-1)+i+2l-1+1$
$= nl+2l+1$.

(n-s(i) s(i))----------(s(i+2l-1) n-s(i+2l-1))
----------2l---------->

$R(i,2l) = n-i+n(l-1)+i+2l-1$
$= nl+2l-1$.

2.b) Si la somme contient n on distingue les cas suivants :

(s(i) n-s(i))---------(n)----------**(s(i+2l-1) n-s(i+2l-1))**
----------2l---------->

$R(i,2l) = n+i+2l-1-n+n(l-1)+1$.

Comme la somme traverse n alors $i+2l-1-n > 0$ et par suite $R(i,2l) > nl$.

Par ailleurs :

$R(i,2l) = nl+2l+i-n$ avec $i < n$ d'où : $R(i,2l) < nl+2l$.

(n-s(i) s(i))----------(n)---------**(n-s(i+2l-1) s(i+2l-1))**
----------2l---------->

$$R(i,2l) = n-i+n+n(l-1)$$
$$= nl+n-i \quad \text{avec } n > i$$
$$> nl .$$

R(i,2l) est majorée par nl+2l car R(i,2l) = nl+2l+n-(i+2l) . Et comme la somme traverse n , i+2l > n et par suite R(i,2l) < nl+2l .

Ceci termine la démonstration pour le cas n = 4p+3 .

Pour le cas n pair , il suffit d'examiner les sommes qui font intervenir le couple dégénéré n/2 .

Supposons n = 4p+2 .

1) Si le nombre de termes est de la forme 4r+1 on a :

(n-s(i) s(i))----------(n/2)--------(s(i+2l) n-s(i+2l))
----------2l+1---------->

$$R(i,2l+1) = n-i+n/2+i+2l+n(l-1)$$
$$= nl+2l+n/2 .$$

Comme le nombre de termes est au plus égal à n/2 il suit :

$$nl+2l < R(i,2l+1) < n(l+1) .$$

2) Si le nombre de termes est de la forme 4r+3 on distingue les cas suivants :

(n-s(i) s(i))----------(n/2)---------**(s(i+2l) n-s(i+2l))**
----------2l+1---------->

$$R(i,2l+1) = n-i+n/2+i+2l+1+n(l-1)$$
$$= nl+2l+1+n/2 .$$

Comme le nombre de termes est au plus égal à n/2 il suit :

$$nl+2l+1 < R(i,2l+1) \leq n(l+1) .$$

Supposons que la somme est calculée sur un nombre pair de termes .

1) Si le nombre de termes est de la forme 4r on a les cas suivants :

(n-s(i) s(i))----------(n/2)
----------2l--------->

$$R(i,2l) = n-i+n/2+n(l-1)$$
$$= nl+n/2-i \text{ avec } i < n/2$$
$$> nl .$$

Ajoutons et retranchons 2l au second membre ; on obtient :

$$R(i,2l) = nl+2l+n/2-(i+2l) \text{ avec } i+2l > n/2$$
$$< nl+2l .$$

(s(i) n-s(i))----------(n/2)---------**(s(i+2l-1) n-s(i+2l-1))**
----------2l---------->

D'une part on a :

$$R(i,2l) = n/2+i+2l+n(l-1)$$
$$= nl+i+2l-n/2 \text{ avec } i+2l > n/2$$
$$> nl .$$

D'autre part ,

$$R(i,2l) = nl+2l+i-n/2 \text{ avec } i < n/2$$
$$< nl+2l .$$

(n/2)----------**(s(i+2l-1) n-s(i+2l-1))** où i = n/2
----------2l---------->

$$R(i,2l) = n/2+i+2l+n(l+1)$$
$$= nl+2l .$$

2) Si le nombre de termes est de la forme 4r+2 on a :

(n-s(i) s(i))----------(n/2)---------**(n-s(i+2l-1) s(i+2l-1))**
----------2l---------->

$$R(i,2l) = n-i+n/2+n(l-1)$$
$$= nl+n/2-i$$
$$> nl \text{ car } i < n/2 .$$

Ajoutons et retranchons 2l au second membre ; il vient :

$$R(i,2l) = nl+2l+n/2-(i+2l)$$
$$< nl+2l \text{ car } i+2l > n/2 .$$

Ceci termine la démonstration du lemme pour $n = 4p+2$.

Le cas $n = 4p$ n'est guère différent . En effet la particularité que ce cas introduit apparait lorsque le couple dégénéré est permuté avec son dual ; or cela ne modifie que la position de celui-ci et par suite la forme du nombre de termes des sommes qui le font intervenir et qui du reste coincide avec au moins une des situations du cas $n = 4p+2$.

Nous venons de montrer que pour tout $n \neq 4p+1$, si s est une k-permutation tel que $k = 2$ ou $k > (n/2+e)/4$ où $e = 1$ ou 2 alors , pour tout (i,j,l) tel que $l \leq n/2$, $R(i,l+1) \geq R(i,l)$. Ce résultat se prolonge assez aisément lorsque $l \geq n/2$ grâce au lemme 4.3 . Aussi peut-on énoncer :

Proposition 4.3 :

Pour tout $n \neq 4p+1$, une condition nécessaire et suffisante pour qu'une k-permutation soit solution est
$k = 2$ pour tout n
$k = p+1$ si n est impair
$k > (n/2+e)/4$ avec $e = 1$ ou 2 selon que respectivement $n = 4p$ ou $n = 4p+2$.

5 - Conclusion

L'analyse que nous venons de faire concerne la performance d'un modèle théorique de synchronisation locale dans un réseau de processeurs en pipeline disposant chacun d'un buffer de profondeur 1 et dont le cycle des temps est une séquence périodique dont la période est une permutation de $1 , 2 , \ldots , n$.
Un problème intéressant peut être l'extension de cette étude au cas où les cycles de temps sont des variables aléatoires obéissant à une certaine loi de probabilité et où chaque processeur peut disposer d'un buffer de profondeur supérieure à 1 .

Références

[1] **H . T . KUNG** : Why systolic architecture , Computer Magazine 15 , pp 37 - 46 (1986) .

[2] **T . MUNTEAN** : Introduction à OCCAM langage parallèle issu de CSP pour la programmation des systèmes de transinateurs , Laboratoire de Génie Informatique , Décembre 1983 .

ANNEXE

UNE CONJECTURE SUR LES CYCLES LIMITES DES RESEAUX BOOLEENS MONOTONES

1 - Introduction

L'étude du comportement dynamique des systèmes discrets est un domaine où les résultats généraux sont très rares . La difficulté de l'étude est due principalement à l'aspect combinatoire du problème . Des études approfondies ont été menées par de nombreux auteurs pour l'analyse de certaines classes de réseaux . On peut citer en particulier A . GILL pour les réseaux linéaires , E . GOLES et al pour les réseaux à seuil , E . GOLES , Y . ROBERT et M . TCHUENTE pour les réseaux neuronaux , F . ROBERT , ...
Nous nous intéressons ici aux cycles limites engendrés par les réseaux booléens . Le problème peut se formuler comme suit .
Considérons un réseau d'automates composé de n cellules binaires c'est à dire dont l'ensemble des états possibles $E = \{0,1\}$.
- Une configuration du réseau à un instant t est l'ensemble des états de ses cellules ; on la représente par un vecteur $x(t) = (x_i(t) ; 1 \leq i \leq n)$ où $x_i(t) \in E$.
Soit $F = (f_i ; 1 \leq i \leq n)$ une fonction globale de transition du réseau c'est à dire :

$$x(t+1) = F((x(t)) \quad \forall t .$$

- On dit que F est un opérateur parallèle si à partir d'une configuration x(t) , chaque cellule i prend conscience de son environnement et évolue selon sa loi locale de transition f_i :

$$x_i(t+1) = f_i(x_1(t) , x_2(t) , \ldots , x_n(t)) .$$

- On appelle opérateur de GAUSS-SEIDEL associé à un opérateur parallèle $F = (f_i ; 1 \leq i \leq n)$ un opérateur $G = (g_i ; 1 \leq i \leq n)$ tel que :

$$g_i(x(t)) = f_i(g_1(x(t)) , g_2(x(t)) , \ldots , g_{i-1}(x(t)) , x_i(t) , x_{i+1}(t) , \ldots , x_n(t))) .$$

Clairement , G est un opérateur série : à partir d'une configuration x(t) du réseau , on active les cellules dans l'ordre périodique 1 , 2 , . . . , n selon le schéma suivant :

$$x_i(t+1) = f_i(x_1(t+1) , x_2(t+1) , \ldots , x_{i-1}(t+1) , x_i(t) , x_{i+1}(t) , \ldots , x_n(t)) .$$

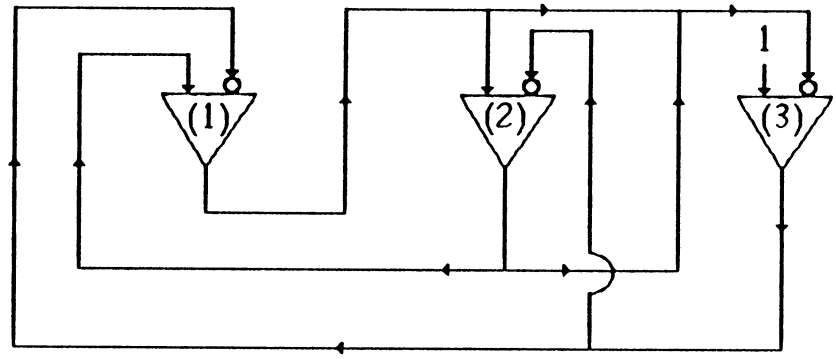
- Un opérateur $F = (f_i ; 1 \leq i \leq n)$ est dit monotone si et seulement si

$$(\forall i , x_i \leq y_i) \Rightarrow (\forall i , f_i(x) \leq f_i(y)) ;$$

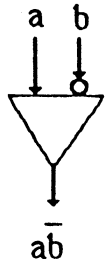
on note aussi $(x \leq y) \Rightarrow (F(x) \leq F(y))$. Il est bien connu que E étant fini , pour un opérateur quelconque H , la suite $x(t+1) = F(x(t))$ n'a que deux comportements limites possibles :

- ou elle finit par rester stationnaire sur la même configuration x^* : $F(x^*) = x^*$; on dit que x^* est une configuration stable du réseau ,
- ou elle finit par décrire indéfiniment le même ensemble ordonné $C = (x^{(i)} ; 1 \leq i \leq p)$ de configurations $x^{(i)}$; on dit que C est un cycle de longueur p .

Exemple [3] : Considérons un réseau de trois cellules connectées comme l'indique la figure ci-dessous .

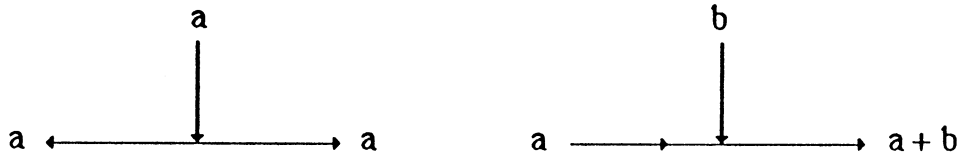


où chaque cellule est supposée être un modèle , très rudimentaire , de neurone :



$$\bar{b} = \begin{cases} 1 & \text{si } b = 0 \\ 0 & \text{sinon .} \end{cases}$$

a et b sont des valeurs booléennes dites respectivement excitatrice et inhibitrice du neurone . Avec les conventions



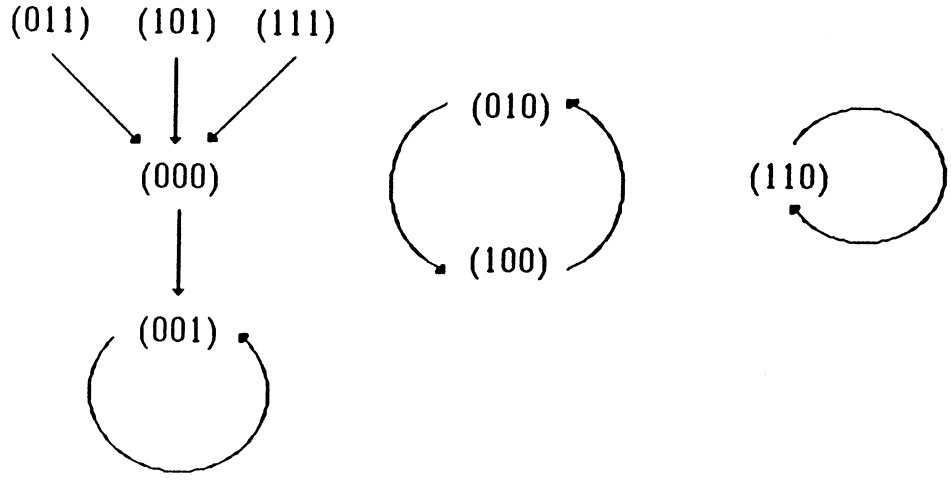
la fonction globale de transition F de ce réseau est défini par

$$f_1(x_1 , x_2 , x_3) = x_2\bar{x}_3$$

$$f_2(x_1 , x_2 , x_3) = x_1\bar{x}_3$$

$$f_3(x_1 , x_2 , x_3) = \overline{x_1 + x_2} = \bar{x}_1\bar{x}_2$$

Le schéma ci-dessous indique le graphe d'itération du réseau .



Ce réseau admet 001 et 110 comme configurations stables tandis que le couple (010 , 100) constitue un cycle .

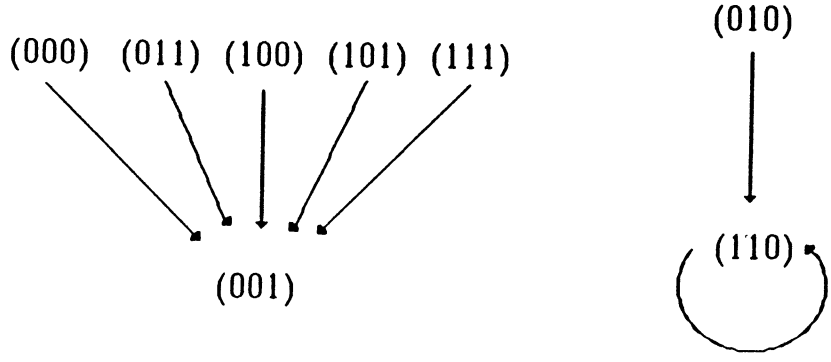
L'opérateur de GAUSS-SEIDEL associé à F est défini par

$$g_1(x_1 , x_2 , x_3) = x_2\bar{x}_3$$

$$g_2(x_1 , x_2 , x_3) = x_2\bar{x}_3$$

$$g_3(x_1 , x_2 , x_3) = \bar{x}_2 + \bar{x}_3 .$$

Son graphe d'itération comporte les mêmes configurations stables que F par contre le cycle a disparu .



Dans ce chapitre , nous nous intéressons à la longueur du plus long cycle qu'un

réseau booléen monotone d'automates , évoluant selon un opérateur de GAUSS-SEIDEL associé à un opérateur parallèle monotone peut engendrer .

2 - Quelques définitions et résultats préliminaires

Il est bien connu que dans un cycle engendré par un opérateur monotone deux éléments distincts sont incomparables .
Considérons l'application f de l'ensemble P(A) des parties d'un ensemble $A = \{a_i ;\ 1 \leq i \leq n\}$ dans l'ensemble des vecteurs à n composantes booléennes $x = (x_j \ 1 \leq j \leq n)$ tels que

$$x_j = \begin{cases} 1 \quad \text{si} \quad a_j \in A_i \\ 0 \quad \text{sinon} . \end{cases}$$

f est évidemment bijectif et en plus :

$$(A_i \subseteq A_j) \Leftrightarrow (f(A_i) \leq f(A_j)) .$$

Cette bijection identifie tout système $X = (x_{ij} ; 1 \leq i \leq p ; 1 \leq j \leq n)$ de configurations $x_{i.} = (x_{ij} ; 1 \leq j \leq n)$ deux à deux incomparables à un système de SPERNER c'est à dire un système S de sous-ensemble A_i d'un ensemble fini A tel que :

$$\forall (i \neq j) , A_i \not\subset A_j \quad \text{et} \quad A_j \not\subset A_i .$$

Depuis 1928 , il a été établi un ensemble de résultats connus sous le nom de théorèmes d'intersection de systèmes d'ensembles finis [1] , [2] , [4] , [8] qui ne sont que des cas particuliers d'un résultat plus général dit théorème de SPERNER .

Théorème de SPERNER :

Le nombre maximum de sous-ensembles d'un système de SPERNER sur un ensemble de n éléments est égal à $C_n^{\lfloor n/2 \rfloor}$.

Comme tout cycle d'un opérateur monotone F a ses configurations deux à deux incomparables , le théorème de SPERNER peut être interprété comme suit :
La longueur du plus long cycle qu'un réseau d'automates évoluant selon un opérateur monotone est susceptible de décrire est égale à $C_n^{\lfloor n/2 \rfloor}$.
Si un tel cycle existe , il correspond évidemment au système de tous les sous ensemble à $\lfloor n/2 \rfloor$ éléments d'un ensemble de n éléments .

Tout opérateur de GAUSS-SEIDEL associé à un opéarteur monotone étant monotone [3], $C_n^{\lfloor n/2 \rfloor}$ donne une borne de la longueur de ses cycles .
Comme nous l'allons voir , cette majoration est grossière .

3 - Quelques propriétés des cycles d'OGSM

Soit $X = (x_{ij} ; 1 \leq i \leq p ; 1 \leq j \leq n)$ un cycle d'un OGSM G sur E^n .
Nous avons vu que chaque pas de l'itération de G correspond à n itérations locales sur les cellules i = 1 , 2 , . . . , n . Par rapport à X , chaque transition locale se traduit par la modification des composantes de la colonne dont le numéro est celui de la cellule qu'on fait évoluer .
Désignant par $X(k) = (x_{ij}^{(k)} ; 1 \leq i \leq p ; 1 \leq j \leq n)$ le tableau des configurations obtenues après l'évolution de la $k^{ème}$ cellule du réseau , on a évidemment :

$$x_{ij}^{(k)} = \begin{cases} x_{i+1j} & \text{si} \quad 1 \leq j \leq k \\ x_{ij} & \text{sinon} \end{cases}$$

où les indices de lignes sont calculés modulo p . Par exmple pour n = 5 et p = 4 on a

$$X(3) = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ x_{21} & x_{22} & x_{23} & x_{24} & x_{25} \\ x_{31} & x_{32} & x_{33} & x_{34} & x_{35} \\ x_{41} & x_{42} & x_{43} & x_{44} & x_{45} \\ x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \end{bmatrix}$$

Les éléments $y \in \{0 , 1\}^5$ de X(3) sont reliés par des traits .

Lemme 3.1 :

Si $X = (x_{ij} ; 1 \leq i \leq p ; 1 \leq j \leq n)$ est un cycle d'un OGSM $G = (g_i)$, alors pour tout k , X(k) est un système de SPERNER .

Démonstration :

X(k) est un tableau dont chaque ligne représente l'état du réseau après l'activation des cellules k+1 , k+2 , . . . , n-1 , n , 1 , 2 , . . . , k dans cet ordre . Comme le tableau X représentant les états du réseau au cours de la même évolution et après l'activation des cellules 1 , 2 , . . . , n dans cet ordre , correspond à un cycle , il en est forcément de même pour X(k) ; et d'après le lemme 3.1 , les lignes de X(k) forment un système de SPERNER .

Remarque : Soit X un tableau booléen d'ordre pxn . Notons :
X' le tableau obtenu de X en renversant l'ordre de balayage en lignes et en colonne

$\bar{X}$ le tableau obtenu en complémentant les éléments de X .

Si X est un cycle d'un OGSM alors X' et $\bar{X}$ le sont .

Notons C_n l'ensemble des cycles générés par un OGSM d'ordre n .

Lemme 3.2 :

La longueur maximum d'un élément de C_n est au moins égale à $C_{\lfloor n/2 \rfloor}^{\lfloor n/4 \rfloor}$.

En effet si $F : E^n \dashrightarrow E^n$ est une fonction booléenne monotone qui génère en parallèle $q \geq C_n^{\lfloor n/2 \rfloor}$ alors on vérifie aisément que la fonction

$$H : E^{2n} \dashrightarrow E^{2n}$$
$$(X_1 , X_2) \dashrightarrow (F(X_2) , F(X_1))$$

génère séquentiellement un cycle de longueur q .

Soit $s(k) = \sum_{j>k} e_j$ où e_j désigne le $j^{ème}$ vecteur de la base canonique . On a :

Lemme 3.3 : Si X appartient à C_n alors il ne contient aucun s(k) .

En effet soit (i,k) tel que $x_{i.} = s(k)$ et considérons X(k) . On observe que

$$x_{i.}^{(k)} \leq x_{i+1.}^{(k)}$$

ce qui contredit l'appartenance de X à C_n en vertu du lemme 3.1 .

De manière analogue on vérifie que , si X appartient à C_n il ne contient aucun des $\bar{s}(k)$.

Deux conséquences immédiates de ces lemmes sont :
- Aucun OGSM sur E^2 n'admet de cycle .
- Si un OGSM sur E^3 admet un cycle X alors ce cycle est unique et on a :

$$X = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

On vérifie aisément que le passage de X(k-1) à X(k) pour tout k , s'effectue par une permutation circulaire d'ordre 1 de la $k^{ème}$ colonne de X(k-1) . La contrainte qui veut que X(k) soit un système de SPERNER pourrait nous amener à croire , du fait que X(k-1) est lui même un système de SPERNER, que les informations apportées par la colonne permutée sont redondantes et donc que le tableau X'(k) déduit de X(k) par suppression de sa $k^{ème}$ colonne est un système de SPERNER . En fait ce n'est pas toujours le cas comme le montre l'exemple sur E^5 ci-dessous

$$X = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

X'(2) , X'(4) , X'(5) sont des systèmes de SPERNER par contre X'(1) , X'(3) ne le sont pas .

Lemme 3.4 :

Si X appartient à C_n alors pour tout k , deux lignes voisines de X'(k) sont incomparables .

Démonstration :

Supposons que deux lignes voisines du tableau X'(k) soient comparables ; on a par exemple pour k = 3 la situation suivante :

$$\begin{array}{cccccc} & & x & a_4 & a_5 & a_6 \\ a_1 & a_2 & a & b_4 & b_5 & b_6 \\ b_1 & b_2 & b & & & \end{array}$$

avec $a_i \leq b_i$ pour tout $i \neq 3$.
Comme $(a_1 , a_2 , a , a_4 , a_5 , a_6)$ et $(b_1 , b_2 , b , b_4 , b_5 , b_6)$ sont deux lignes incomparables de X'(k) , on déduit que a = 1 et b = 0 . Par ailleurs , $(a_1 , a_2 , x , a_4 , a_5 , a_6)$ et $(b_1 , b_2 , a , b_4 , b_5 , b_6)$ sont deux lignes incomparables de X(k-1) d'où x = 1 , a = 0 ; ce qui est une contradiction .

Une conséquence immédiate de ce lemme est que si X est un cycle de longueur au plus égale à 3 alors pour tout k , X(k) est un système de SPERNER car deux configurations quelconques sont voisines . L'exemple précédent montre que ce résultat n'est pas nécessairement vrai pour des longueurs supérieures à 3 . En fait nous allons nous intéresser ici à la conjecture plus forte qui s'énonce comme suit :

Conjecture :

Pour tout X appartenant à C_n il existe k , $1 \leq k \leq n$, tel que X'(k) soit un système de SPERNER .

Dans la suite nous noterons C_n^* l'ensemble de tels X .

Commentaire :

Cette conjecture est étroitement liée à la conjecture précédente . En effet , supposons la vraie . Si X appartient à C_n^* alors , il existe k tel que X(k) a ses lignes deux à deux incomparables donc est un système de SPERNER ; le nombre de ses lignes est au plus égal à $C_{n-1}^{\lfloor (n-1)/2 \rfloor}$.

Lemme 3 . 5 :

(i) Si $X \in C_n$ contient une colonne égale à 0 ou à 1 alors $X \in C_n^*$.
(ii) Si $X \in C_n$ est tel que dans sa $k^{ème}$ colonne tous les 1 sont isolés alors $X \in C_n^*$.

Démonstration :

(i) Il est évident que si X contient une colonne k identiquement nulle ou égale à 1 alors X'(k) est un système de SPERNER .
(ii) Considérons un X dont la $k^{ème}$ colonne a tous ses 1 isolés . Si X'(k) n'est pas un système de SPERNER alors il existe au moins r et s non consécutifs d'après le lemme 3 . 4 tels que $x'_r{}^{(k)} \leq x'_s{}^{(k)}$. Comme X(k) et X(k-1) sont des systèmes de SPERNER nécessairement on a le schéma suivant par exemple pour k = 3 :

$$\begin{array}{cccc} & c & a_4 & a_5 & a_6 \\ a_1 & a_2 & a & & \\ & c' & b_4 & b_5 & b_6 \\ b_1 & b_2 & b & & \end{array}$$

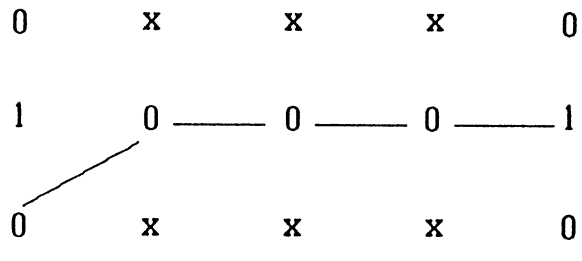
où $x'_{r.}(3) = (a_i ; i \neq 3)$ et $x'_{s.}(3) = (b_i ; i \neq 3)$. Comme $(a_1 , a_2 , a , a_4 , a_5 , a_6)$ et $(b_1 , b_2 , b , b_4 , b_5 , b_6)$ sont incomparables alors a = 1 et b = 0 .
Par ailleurs $(a_1 , a_2 , c , a_4 , a_5 , a_6)$ et $(b_1 , b_2 , c' , b_4 , b_5 , b_6)$ sont incomparables alors c = 1 et c' = 0 ; ce qui contredit l'hypothèse car c et a sont voisins et égaux à 1 .

Corollaire : Tout X de C_n contenant une configuration dont le cardinal égale 1 (respectivement égale n -1) est un élément de C_n^* .
En effet , si $x_{i.} = e_k$ alors la $k^{ème}$ colonne de X comporte un seul élément égal à 1 et qui est donc forcément isolé ; il suffit alors d'appliquer le lemme précédent .

Lemme 3 . 6 :

Si dans un tableau $X \in C_n$ l'une des lignes est égale à (10...01) alors il existe k tel que X'(k) est un système de SPERNER .

En effet si X contient une telle configuration alors nécessairement en vertu du lemme 3.4 pour k = 1 et pour k = n on a le schéma suivant :



Comme X(1) est un système de SPERNER et qu'il contient la configuration (0. . . 01) reliée sur le schéma par un trait alors la $n^{ème}$ colonne de X contient un et un seul 1 . Le raisonnement est identique pour la configuration duale .

Proposition : $C_4 = C_4*$.

Démonstration :

Soit $X \in C_n$. Supposons que X ne vérifie pas les lemmes 3 . 5 et 3 . 6 alors il comporte nécessairement les lignes (0101) et (1010) ce qui est une contradiction .

Remarquons que pour tout n si X est de longueur 4 alors il appartient à C_{n^*} .
Soit X de longueur 4 ne vérifiant pas le lemme 3.5 . X est un système dont les colonnes sont nécessairement

$$\begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Alors pour tout $n \geq 5$, X contient au moins deux colonnes identiques dont on vérifie aisément que l'une est redondante quant à l'information qu'elle apporte sur l'incomparabilité deux à deux des configurations des X(k) . C'est à dire qu'il existe k pour lequel X'(k) est un système de SPERNER .

Nous donnons dans les pages suivantes le programme PASCAL de recherche exhaustive de tous les cycles que peut générer un réseau booléen monotone de taille n et à titre d'exemple les résultats de la recherche des cycles de longueur supérieure ou égale à 5 pour n = 5 .
Chaque cycle X est suivi des X(k) sous la forme :

X X(1) X(2) X(3) X(4) ;

la première colonne correspond à l'écriture décimale des lignes de X .

```
program gauss_seideld ;
type vect=array[0..10] of integer;
     mat=array[1..100,0..10] of integer;
var val : vect;lst1:text;
    aa,a1,m : mat;
    npermut,nvar,nconfig,lcycle,pointer,i,j,l : integer;
    compa:boolean;
function fact(p:integer):integer;
(*calcule le factoriel de p*)
begin if p=0 then fact:=1 else fact:=p*fact(p-1);end;
procedure ecribin(x,i:integer;var a:mat);
(*ecrit l entier x en base 2*)
var j,y:integer;
begin
  j:=nvar;y:=x;
  while j>=1 do
  begin
       a[i,j]:=y mod 2;
       y:=y div 2;
       j:=j-1;
  end;
end;
procedure init(var q:integer;var a:mat);
(*recherche les entiers dont l ecriture
   binaire peut apparaitre dans un cycle*)
var i,j,c,k1,k2,tranche:integer;
function puissce(p:integer):integer;
( *calcule la puissance p eme de 2*)
begin
   if p=0 then puissce :=1
   else puissce :=2*puissce(p-1);
end;
begin
  tranche:=1;k1:=0;k2:=0;j:=0;
  while tranche<=nvar-2 do
    begin
      c:=puissce(tranche);k1:=k2+1;k2:=c+k1-2;
      for i:=k1 to k2 do a[i,j]:=c+i-k1;
      tranche:=tranche+1;
    end;
  c:=puissce(nvar)-1;k1:=k2+1;k2:=2*k2;q:=k2;
  for i:=k1 to k2 do a[i,j]:=c-a[k2-i+1,j];
  for i:=1 to q do begin c:=a[i,j];ecribin(c,i,a);end;
end;
procedure generer(var l:integer;var t:vect);
(*genere les permutations de n elements*)
var n,p,q,r:integer;pos:vect;
begin
  n:=lcycle;
  while n>2 do
    begin
      p:=fact(n-1);q:=l div p;r:=l mod p;
      if q=0 then begin pos[n]:=n;l:=r;end
      else if r=0 then begin pos[n]:=n-q+1;l:=p;end
           else begin pos[n]:=n-q;l:=r;end;n:=n-1;
    end;
  if l=2 then begin pos[2]:=1;pos[1]:=2;end
  else begin pos[1]:=1;pos[2]:=2;end;
  for i:=1 to lcycle do t[i]:=0;
  for i:=1 to lcycle do
```

```
    begin
      if t[pos[i]]<>0 then
      for p:=i downto pos[i]+1 do t[p]:=t[p-1];
      t[pos[i]]:=i;
    end;
    for i:=1 to lcycle do write(t[i]:4);writeln;writeln;
end;
procedure comparer(i,j:integer;a:mat);
(*compare deux vecteurs de booleens*)
var c1,c2,k:integer;
begin
  c1:=0;c2:=0;k:=1;compa:=true;
  while k<=nvar do
  begin
    if a[i,k]<a[j,k] then
    begin
       c1:=1;if c1=c2 then
       begin compa:=false;k:=nvar+1;end
       else k:=k+1;
    end
    else if a[i,k]>a[j,k] then
    begin
       c2:=1;if c1=c2 then
       begin compa:=false;k:=nvar+1;end
       else k:=k+1;
    end
    else k:=k+1;
  end;
end;
procedure tester(a : mat);
(*teste si a est un systeme de sperner*)
var i,l : integer;
begin
  i:=1;
  while i<=lcycle-1 do
    begin
      l:=i+1;comparer(i,l,a);
      while (compa=false) and (l<lcycle) do
        begin l:=l+1;comparer(i,l,a);end;
      if compa=false then i:=i+1 else i:=lcycle;
    end
end;
procedure sperner(a : mat);
(*genere a(k) et teste s il est systeme de sperner*)
type matridim = array[0..10,1..20,1..20] of integer;
var i,j,l,s : integer;
        x : matridim;
        b : mat;
begin
  for i:=1 to lcycle do
    begin
      x[0,i,0]:=a[i,0];
      for j:=1 to nvar do
         begin x[0,i,j]:=a[i,j];b[i,j]:=a[i,j];end;
    end;j:=0;compa:=false;
  while (compa=false) and (j<nvar-1) do
    begin
      j:=j+1;s:=b[1,j];
      for i:=1 to lcycle-1 do b[i,j]:=b[i+1,j];
      b[lcycle,j]:=s;tester(b);
```

```
      if compa = false then
        for i:=1 to lcycle do
          for l:=1 to nvar do x[j,i,l]:=b[i,l];
    end;
  if compa = false then
    begin
      for i:=1 to lcycle do
        begin
          write(lst1,x[0,i,0]:3,'    ');
          write(x[0,i,0]:3,'    ');
          for s:=0 to nvar-1 do
            begin
              for j:=1 to nvar do
              begin write(lst1,x[s,i,j]);write(x[s,i,j]);end;
              write(lst1,'    ');write('    ');
            end;
          writeln(lst1);writeln;
        end;
      writeln(lst1);writeln;writeln(lst1);writeln;
    end;
end;
procedure traiter(var a : mat);
(*verifie si a est un cycle de gauss_seidel*)
var permut : vect;
     j,k,l : integer;
begin
  compa:=true;tester(a);
  if compa = false then
    begin
      sperner(a);
      for l:=2 to npermut do
        begin
          k:=1;generer(k,permut);
          for k:=1 to lcycle do
            for j:=0 to nvar do a1[k,j]:=a[permut[k],j];
          sperner(a1);
        end
    end
end;
begin
  assign(lst1,'B:RESULT.PAS');
  rewrite(lst1);writeln(lst1);writeln;
  write(lst1,'ENTRER LE NOMBRE DE VARIABLES ET LA LONGUEUR DU CYCLE : ');
  write('ENTRER LE NOMBRE DE VARIABLES ET LA LONGUEUR DU CYCLE : ');
  read(nvar,lcycle);write(lst1,nvar,'   ',lcycle);
  writeln(lst1);writeln(lst1);writeln;writeln;
  npermut:=fact(lcycle-1);nconfig:=0;init(nconfig,m);
  for i:=1 to lcycle do
    begin
      val[i]:=i;
      for j:=0 to nvar do aa[i,j]:=m[i,j]
    end;
  val[0]:=nconfig-lcycle;traiter(aa);pointer:=lcycle;
  while pointer >=1 do
    begin
      pointer:=lcycle;
      while (val[pointer]=nconfig-lcycle+pointer)
      and (pointer>=1) do pointer:=pointer-1;
      if val[pointer]<>nconfig-lcycle+pointer then
        begin
```

```
          val[pointer]:=val[pointer]+1;
          if pointer<>lcycle then
          for l:=pointer+1 to lcycle do val[l]:=val[l-1]+1;
          for l:=pointer to lcycle do
            for j:=0 to nvar do aa[l,j]:=m[val[l],j];
          traiter(aa);
        end
   end;close(lst1);
end.
```

ENTRER LE NOMBRE DE VARIABLES ET LA LONGUEUR DU CYCLE : 5 5

19	10011	00011	00011	00111	00101
5	00101	00101	01101	01001	01011
10	01010	01010	01010	01110	01100
12	01100	11100	11100	11000	11000
25	11001	11001	10001	10001	10011

19	10011	00011	00011	00111	00101
5	00101	00101	01101	01001	01011
10	01010	11010	10010	10110	10100
20	10100	10100	11100	11000	11000
25	11001	11001	10001	10001	10011

9	01001	11001	10001	10001	10011
19	10011	00011	00011	00111	00111
6	00110	00110	01110	01010	01010
10	01010	11010	10010	10110	10100
21	10101	00101	01101	01001	01001

9	01001	11001	10001	10001	10011
19	10011	00011	00011	00111	00111
6	00110	00110	01110	01110	01100
12	01100	11100	10100	10100	10100
21	10101	00101	01101	01001	01001

10	01010	11010	10010	10010	10010
19	10011	00011	00011	00111	00111
6	00110	00110	01110	01110	01100
12	01100	11100	10100	10100	10100
21	10101	00101	01101	01001	01011

10	01010	11010	10010	10010	10010
19	10011	00011	00011	00111	00111
6	00110	00110	01110	01110	01100
12	01100	11100	11100	11000	11000
25	11001	01001	01001	01001	01011

10	01010	11010	10010	10110	10100
21	10101	00101	00101	00101	00111
6	00110	00110	01110	01110	01100
12	01100	11100	11100	11000	11000
25	11001	01001	01001	01001	01011

19	10011	00011	00011	00111	00111
6	00110	00110	01110	01110	01100
13	01101	01101	01101	01001	01011
10	01010	11010	10010	10110	10100
21	10101	10101	10101	10001	10011

19	10011	00011	00011	00111	00111

6	00110	00110	01110	01110	01100
13	01101	01101	01101	01001	01011
10	01010	11010	11010	11010	11000
25	11001	11001	10001	10001	10011
19	10011	00011	00011	00111	00111
6	00110	00110	01110	01010	01010
10	01010	11010	10010	10110	10100
20	10100	10100	11100	11000	11000
25	11001	11001	10001	10001	10011
19	10011	00011	00011	00111	00111
6	00110	10110	10110	10110	10100
21	10101	00101	01101	01001	01011
10	01010	11010	11010	11010	11000
25	11001	11001	10001	10001	10011
21	10101	00101	01101	01001	01011
11	01011	01011	00011	00111	00111
6	00110	00110	01110	01110	01100
12	01100	11100	11100	11000	11000
25	11001	11001	10001	10101	10101
21	10101	00101	01101	01001	01011
11	01011	01011	00011	00111	00111
6	00110	00110	01110	01110	01100
12	01100	11100	11100	11000	11010
26	11010	11010	10010	10110	10100
18	10010	10010	10010	10110	10100
21	10101	00101	00101	00101	00111
6	00110	00110	01110	01110	01100
12	01100	11100	11100	11000	11000
25	11001	11001	10001	10001	10011
19	10011	00011	00011	00111	00111
6	00110	10110	10110	10110	10100
21	10101	00101	01101	01101	01101
12	01100	11100	11100	11000	11000
25	11001	11001	10001	10001	10011
21	10101	10101	10101	10001	10011
19	10011	00011	00011	00111	00111
6	00110	00110	01110	01110	01100
12	01100	11100	11100	11000	11010
26	11010	11010	10010	10110	10100
18	10010	10010	10010	10110	10100
21	10101	00101	01101	01001	01011
10	01010	01010	01010	01110	01100
12	01100	11100	11100	11000	11000
25	11001	11001	10001	10001	10011

19	10011	00011	01011	01011	01011
10	01010	11010	10010	10110	10100
21	10101	00101	01101	01101	01101
12	01100	11100	11100	11000	11000
25	11001	11001	10001	10001	10011

19	10011	00011	01011	01011	01011
10	01010	11010	10010	10110	10110
22	10110	00110	01110	01110	01100
12	01100	11100	11100	11000	11000
25	11001	11001	10001	10001	10011

21	10101	00101	01101	01001	01011
10	01010	11010	10010	10110	10110
22	10110	00110	01110	01110	01100
12	01100	11100	11100	11000	11000
25	11001	11001	10001	10101	10101

ENTRER LE NOMBRE DE VARIABLES ET LA LONGUEUR DU CYCLE : 5 6

19	10011	00011	00011	00111	00101
5	00101	00101	01101	01001	01011
10	01010	11010	10010	10110	10110
22	10110	00110	01110	01110	01100
12	01100	11100	11100	11000	11000
25	11001	11001	10001	10001	10011

21	10101	00101	01101	01001	01001
9	01001	11001	10001	10001	10011
19	10011	00011	00011	00111	00111
6	00110	00110	01110	01110	01100
12	01100	11100	11100	11000	11010
26	11010	11010	10010	10110	10100

19	10011	00011	00011	00111	00111
6	00110	00110	01110	01010	01010
10	01010	11010	10010	10110	10100
21	10101	00101	01101	01101	01101
12	01100	11100	11100	11000	11000
25	11001	11001	10001	10001	10011

19	10011	00011	00011	00111	00111
6	00110	00110	01110	01110	01100
12	01100	11100	10100	10100	10100
21	10101	00101	01101	01001	01011
10	01010	11010	11010	11010	11000
25	11001	11001	10001	10001	10011

19	10011	00011	00011	00111	00111
6	00110	10110	10110	10110	10100
21	10101	00101	01101	01001	01011
10	01010	01010	01010	01110	01100
12	01100	11100	11100	11000	11000
25	11001	11001	10001	10001	10011

19	10011	00011	01011	01011	01011
10	01010	11010	10010	10110	10100
21	10101	00101	00101	00101	00111
6	00110	00110	01110	01110	01100
12	01100	11100	11100	11000	11000
25	11001	11001	10001	10001	10011

10	01010	11010	10010	10110	10100
21	10101	10101	10101	10001	10011
19	10011	00011	00011	00111	00111
6	00110	00110	01110	01110	01100
12	01100	11100	11100	11000	11000
25	11001	01001	01001	01001	01011

21	10101	00101	01101	01001	01011

10	01010	11010	10010	10010	10010
19	10011	00011	00011	00111	00111
6	00110	00110	01110	01110	01100
12	01100	11100	11100	11000	11000
25	11001	11001	10001	10101	10101
19	10011	00011	00011	00111	00111
6	00110	00110	01110	01110	01100
13	01101	01101	01101	01001	01011
10	01010	11010	10010	10110	10100
20	10100	10100	11100	11000	11000
25	11001	11001	10001	10001	10011
18	10010	10010	10010	10110	10100
21	10101	00101	01101	01001	01011
11	01011	01011	00011	00111	00111
6	00110	00110	01110	01110	01100
12	01100	11100	11100	11000	11000
25	11001	11001	10001	10001	10011

Références

[1] **A . J . W . HILTON and E . C . MILNER :** Some intersection theorems for systems of finite sets , Quart . J . M . Oxford 18 (1967) 542 - 8 .

[2] **E . C . MILNER :** A combinatorial theorem on systems of sets , J . London M . S , 43 (1968) 204 - 6 .

[3] **F . ROBERT :** Itérations discrètes , A paraître dans North Holland .

[4] **GY . KATONA :** On a conjecture of Erdös and a stronger form of Sperner's theorem , Studia Sci . M . Hung . 1 (1966) 59 - 63 .

[5] **Louis COMTET :** Advanced combinatorics , D Reidel Publishing Company Dordrecht - Holland ; Boston USA .

[6] **M . TCHUENTE :** Sequential simulation of parallel iterations and applications , A paraître dans Theoritical Computer Science .

[7] **M . TCHUENTE :** On cycles generated by sequential transformations , A paraître dans Discrete Mathematics .

[8] **P . ERDOS , Chao KO , R . RADO :** Intersection theorems for systems of finite sets , Quart . J . M . Oxford , 12 (1961) 313 - 20 .

CONCLUSION GENERALE

Dans ce travail , nous avons étudié quelques problèmes soulevés tant par la conception que le fonctionnement de systèmes modélisés par des réseaux d'automates.

Grâce à une exploitation du caractère combinatoire du diagramme espace-temps , la synthèse d'algorithmes systoliques par la méthode de partitionnement conduit pour le shéma de résolution séquentiel de départ à une optimisation parallèle en temps et dans certains cas à une optimisation en nombre de processeurs . Quand bien même le problème d'une allocation optimale en nombre de processeurs reste ouvert , la méthode est bien plus efficace .
Toutefois , en règle générale , les méthodes de conception de réseaux systoliques posent des problèmes qui vont au delà de la validation des algorihmes associés . Pour un problème donné , il aurait été intéressant de pouvoir énoncer et établir des résultats d'optimalité non restrictifs . Mais ceci implique par exemple un examen exhaustif des schémas possibles de réalisation de l'algorithme séquentiel . Aussi faut-il être capable de définir un meilleur algorithme séquentiel dans le sens de celui qui conduit à une meilleure parallélisation .

Si nous avons pu dégager des outils tels la notion de couples duaux , de k-permutation , . . . pour pousser jusqu'au bout une analyse d'un problème de synchronisation dans un réseau d'automates , nous sommes loin d'avoir abordé toute la complexité de ce problème . Dans le contexte d'un réseau dont les processeurs disposent d'un buffer de profondeur p supérieure à 1 et d'un cycle de temps obéissant à une certaine loi de probabilité , une telle étude nécessite plus d'outils . En effet , le fonctionnement d'un tel système étant fondé par la relation

$$d(i,j) = \max (f(i-1,j) , f(i-(p+2),j+1)$$

où d(i,j) , f(i,j) respectivement désignent le début et la fin de la $i^{ème}$ tâche du processeur j , il faudra être capable , entre autres , de définir ce qu'est le maximum de deux variables aléatoires .

Nous avons montré que pour des réseaux de petite taille n , évoluant selon un opérateur de Gauss-Seidel monotone , la longueur maximum d'un cycle est $C_{n-1}^{\lfloor (n-1)2 \rfloor}$. Il reste toutefois à généraliser ce résultat que du reste bien d'observations concourent à confirmer . Cette étude semble être un élément de réponse à la question de savoir quelle relation existerait-il entre opérateurs séquentiels et opérateurs parallèles . En effet , montrer que tout cycle d'un opérateur série sur $\{0 , 1\}^n$ correspond à un système de Sperner sur $\{0 , 1\}^{n-1}$ suggère une certaine correspondance entre opérateurs parallèles sur n-1 variables et opérateurs séquentiels sur n variables .

En somme , outre les difficultés inhérentes à l'étude des réseaux d'automates en tant que systèmes dynamiques discrets évoluant dans un temps discret , bien de choses restent encore à faire pour asseoir , avec des outils mathématiques adaptés , un fondement formel pour l'étude des réseaux d'automates en tant que structures pour le calcul parallèle .

AUTORISATION de SOUTENANCE

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

VU le rapport de présentation de Monsieur Maurice TCHUENTE

Monsieur SAKHO Ibrahima

est autorisé à présenter une thèse en soutenance en vue de l'obtention du titre de DOCTEUR de TROISIEME CYCLE, spécialité "Mathématiques appliquées".

Fait à Grenoble, le 20 mars 1987

D. BLOCH
Président
de l'Institut National Polytechnique
de Grenoble

P.O. le Vice-Président,