Conception et validation des algorithmes systoliques

Abdelhamid Benaini

THESE

présentée par

BENAINI Abdelhamid

pour obtenir le titre de DOCTEUR de

L'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

(arrêté ministériel du 5 juillet 1984)

< Mathématiques Appliquées >

CONCEPTION ET VALIDATION DES ALGORITHMES SYSTOLIQUES

Date de soutenance: 26 Septembre 1988

Composition du jury:

Président:	F. ROBERT
Examinateurs:	M. COSNARD B. PLATEAU P. QUINTON Y. ROBERT M. TCHUENTE

Thèse préparée au sein du laboratoire TIM3, U. A. au CNRS n° 397.

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Président : Georges LESPINARD

Année 1988

Professeurs des Universités

BARIBAUD Michel	ENSERG
BARRAUD Alain	ENSIEG
BAUDELET Bernard	ENSPG
BEAUFILS Jean-Pierre	ENSEEG
BLIMAN Samuel	ENSERG
BLOCH Daniel	ENSPG
BOIS Philippe	ENSHMG
BONNETAIN Lucien	ENSEEG
BOUVARD Maurice	ENSHMG
BRISSONNEAU Pierre	ENSIEG
BRUNET Yves	IUFA
CAILLERIE Denis	ENSHMG
CAVAIGNAC Jean-François	ENSPG
CHARTIER Germain	ENSPG
CHENEVIER Pierre	ENSERG
CHERADAME Hervé	UFR PGP
CHOVET Alain	ENSERG
COHEN Joseph	ENSERG
COUMES André	ENSERG
DARVE Félix	ENSHMG
DELLA-DORA Jean	ENSIMAG
DEPORTES Jacques	ENSPG
DOLMAZON Jean-Marc	ENSERG
DURAND Francis	ENSEEG
DURAND Jean-Louis	ENSIEG
FOGGIA Albert	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 Jean-Claude	ENSPG
JOURDAIN Geneviève	ENSIEG
LACOUME Jean-Louis	ENSIEG
LESIEUR Marcel	ENSHMG
LESPINARD Georges	ENSHMG
LONGEQUEUE Jean-Pierre	ENSPG
LOUCHET François	ENSIEG
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
PERRET René	ENSIEG
PERRET Robert	ENSIEG
PIAU Jean-Michel	ENSHMG
POUPOT Christian	ENSERG
RAMEAU Jean-Jacques	ENSEEG
RENAUD Maurice	UFR PGP
ROBERT André	UFR PGP
ROBERT François	ENSIMAG
SABONNADIERE Jean-Claude	ENSIEG
SAUCIER Gabrielle	ENSIMAG
SCHLENKER Claire	ENSPG
SCHLENKER Michel	ENSPG
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
CHOLLET Jean-Pierre
COEY John
COLINET Catherine
COMMAULT Christian
CORNUEJOLS Gérard
COULOMB Jean- Louis
DALARD Francis
DANES Florin
DEROO Daniel
DIARD Jean-Paul
DION Jean-Michel
DUGARD Luc
DURAND Madeleine
DURAND Robert
GALERIE Alain
GAUTHIER Jean-Paul
GENTIL Sylviane
GHIBAUDO Gérard
HAMAR Sylvaine
HAMAR Roger
LADET Pierre
LATOMBE Claudine
LE GORREC Bernard
MADAR Roland
MULLER Jean
NGUYEN TRONG Bernadette
PASTUREL Alain
PLA Fernand
ROUGER Jean
TCHUENTE Maurice
VINCENT Henri

Chercheurs du C.N.R.S
Directeurs de recherche 1ère Classe

CARRE René
FRUCHART Robert
HOPFINGER Emile
JORRAND Philippe
LANDAU Ioan
VACHAUD Georges
VERJUS Jean-Pierre

Directeurs de recherche 2ème Classe

ALEMANY Antoine
ALLIBERT Colette
ALLIBERT Michel
ANSARA Ibrahim
ARMAND Michel
BERNARD Claude
BINDER Gilbert
BONNET Roland
BORNARD Guy
CAILLET Marcel
CALMET Jacques
COURTOIS Bernard
DAVID René
DRIOLE Jean
ESCUDIER Pierre
EUSTATHOPOULOS Nicolas
GUELIN Pierre
JOUD Jean-Charles
KLEITZ Michel
KOFMAN Walter
KAMARINOS Georges
LEJEUNE Gérard
LE PROVOST Christian
MADAR Roland
MERMET Jean
MICHEL Jean-Marie
MUNIER Jacques
PIAU Monique
SENATEUR Jean-Pierre
SIFAKIS Joseph
SIMON Jean-Paul
SUERY Michel
TEODOSIU Christian
VAUCLIN Michel
WACK Bernard

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

E.N.S.E.E.G

CHATILLON Christian
HAMMOU Abdelkader
MARTIN GARIN Régina
SARRAZIN Pierre
SIMON Jean-Paul

E.N.S.E.R.G

BOREL Joseph

E.N.S.I.E.G

DESCHIZEAUX Pierre
GLANGEAUD François
PERARD Jacques
REINISCH Raymond

E.N.S.H.G

ROWE Alain

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

COURTIN Jacques

E.F.P.

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

DEVINE Rodericq
GERBER Roland
MERCKEL Gérard
PAULEAU Yves

UNIVERSITE Joseph FOURIER (GRENOBLE I)

Président de l'Université :
M. PAYAN Jean Jacques

Année Universitaire 1987 - 1988

MEMBRES DU CORPS ENSEIGNANT DE SCIENCES ET DE GEOGRAPHIE

PROFESSEURS DE 1ère Classe

ARNAUD Paul	Chimie Organique
ARVIEU ROBERT	Physique Nucléaire I.S.N.
AUBERT Guy	Physique C.N.R.S
AURIAULT Jean-Louis	Mécanique
AYANT Yves	Physique Approfondie
BARBIER Marie-Jeanne	Electrochimie
BARJON Robert	Physique Nucléaire ISN
BARNOUD Fernand	Biochimie Macromoléculaire Végétale
BARRA Jean-René	Statistiques-Mathématiques Appliquées
BECKER Pierre	Physique
BEGUIN Claude	Chimie Organique
BELORISKY Elie	Physique
BENZAKEN Claude	Mathématiques Pures
BERARD Pierre	Mathématiques Pures
BERNARD Alain	Mathématiques Pures
BERTRANDIAS Françoise	Mathématiques Pures
BERTRANDIAS Jean-Paul	Mathématiques Pures
BILLET Jean	Géographie
BOELHER Jean-Paul	Mécanique
BONNIER Jane Marie	Chimie Générale
BOUCHEZ Robert	Physique Nucléaire ISN
BRAVARD Yves	Géographie
CARLIER Georges	Biologie Végétale
CAUQUIS Georges	Chimie Organique
CHARDON Michel	Géographie
CHIBON Pierre	Biologie Animale
COHEN ADDAD Jean-Pierre	Physique
COLIN DE VERDIERE Yves	Mathématiques Pures
CYROT Michel	Physique du Solide
DEBELMAS Jacques	Géologie Générale
DEGRANGE Charles	Zoologie
DEMAILLY Jean-Pierre	Mathématiques Pures
DENEUVILLE Alain	Physique
DEPORTES Charles	Chimie Minérale
DOLIQUE Jean-Michel	Physique des Plasmas
DOUCE Roland	Physiologie Végétale
DUCROS Pierre	Cristallographie
FONTAINE Jean-Marc	Mathématiques Pures
GAGNAIRE Didier	Chimie Physique
GERMAIN Jean-Pierre	Mécanique,
GIRAUD Pierre	Géologie
HICTER Pierre	Chimie
IDELMAN Simon	Physiologie Animale
JANIN Bernard	Géographie
JOLY Jean-René	Mathématiques Pures
KAHANE André, détaché	Physique
KAHANE Josette	Physique
KRAKOWIAK Sacha	Mathématiques Appliquées

LAJZEROWICZ Jeanine	Physique
LAJZEROWICZ Joseph	Physique
LAURENT Pierre-Jean	Mathématiques Appliquées
LEBRETON Alain	Mathématiques Appliquées
DE LEIRIS Joël	Biologie
LHOMME Jean	Chimie
LLIBOUTRY Louis	Géophysique
LOISEAUX Jean-Marie	Sciences Nucléaires I.S.N.
LUNA Domingo	Mathématiques Pures
MACHE Régis	Physiologie Végétale
MASCLE Georges	Géologie
MAYNARD Roger	Physique du Solide
OMONT Alain	Astrophysique
OZENDA Paul	Botanique (Biologie Végétale)
PAYAN Jean-Jacques	Mathématiques Pures
PEBAY-PEYROULA Jean-Claude	Physique
PERRIER Guy	Géophysique
PIERRARD Jean-Marie	Mécanique
PIERRE Jean-Louis	Chimie Organique
RENARD Michel	Thermodynamique
RINAUDO Marguerite	Chimie CERMAV
ROSSI André	Biologie
SAXOD Raymond	Biologie Animale
SENGEL Philippe	Biologie Animale
SERGERAERT Francis	Mathématiques Pures
SOUCHIER Bernard	Biologie
SOUTIF Michel	Physique
STUTZ Pierre	Mécanique
TRILLING Laurent	Mathématiques Appliquées
VALENTIN Jacques	Physique Nucléaire I.S.N.
VAN CUTSEM Bernard	Mathématiques Appliquées
VIALON Pierre	Géologie

PROFESSEURS de 2ème Classe

ADIBA Michel	Mathématiques Pures
ANTOINE Pierre	Géologie
ARMAND Gilbert	Géographie
BARET Paul	Chimie
BLANCHI J.Pierre	STAPS
BLUM Jacques	Mathématiques Appliquées
BOITET Christian	Mathématiques Appliquées
BORNAREL Jean	Physique
BRUANDET J.François	Physique
BRUGAL Gérard	Biologie
BRUN Gilbert	Biologie
CASTAING Bernard	Physique
CERFF Rudiger	Biologie
CHIARAMELLA Yves	Mathématiques Appliquées
COURT Jean	Chimie
DUFRESNOY Alain	Mathématiques Pures
GASPARD François	Physique
GAUTRON René	Chimie
GENIES Eugène	Chimie
GIDON Maurice	Géologie
GIGNOUX Claude	Sciences Nucléaires
GILLARD Roland	Mathématiques Pures
GIORNI Alain	Sciences Nucléaires
GONZALEZ SPRINBERG Gérardo	Mathématiques Pures
GUIGO Maryse	Géographie
GUMUCHAIN Hervé	Géographie
GUITTON Jacques	Chimie

HACQUES Gérard	Mathématiques Appliquées
HERBIN Jacky	Géographie
HERAULT Jeanny	Physique
JARDON Pierre	Chimie
JOSELEAU Jean-Paul	Biochimie
KERCKHOVE Claude	Géologie
LONGEQUEUE Nicole	Sciences Nucléaires I.S.N.
LUCAS Robert	Physique
MANDARON Paul	Biologie
MARTINEZ Francis	Mathématiques Appliquées
NEMOZ Alain	Thermodymamique CNRS - CRTBT
OUDET Bruno	Mathématiques Appliquées
PECHER Arnaud	Géologie
PELMONT Jean	Biochimie
PERRIN Claude	Sciences Nucléaires I.S.N.
PFISTER Jean-Claude	Physique du Solide
PIBOULE Michel	Géologie
RAYNAUD Hervé	Mathématiques Appliquées
RICHARD Jean-Marc	Physique
RIEDTMANN Christine	Mathématiques Pures
ROBERT Gilles	Mathématiques Pures
ROBERT Jean-Bernard	Chimie Physique
SARROT-REYNAULD Jean	Géologie
SAYETAT Françoise	Physique
SERVE Denis	Chimie
STOECKEL Frédéric	Physique
SCHOLL Pierre-Claude	Mathématiques Appliquées
SUBRA Robert	Chimie
VALLADE Marcel	Physique
VIDAL Michel	Chimie Organique
VIVIAN Robert	Géographie
VOTTERO Philippe	Chimie

MEMBRES DU CORPS ENSEIGNANT DE L' IUT 1

PROFESSEURS de 1ère Classe

BUISSON Roger	Physique IUT 1
DODU Jacques	Mécanique Appliquée IUT 1
NEGRE Robert	Génie Civil IUT 1
NOUGARET Marcel	Automatique IUT 1
PERARD Jacques	EEA. IUT 1

PROFESSEURS de 2ème classe

BOUTHINON Michel	EEA. IUT 1
CHAMBON René	Génie Mécanique IUT 1
CHEHIKIAN Alain	EEA. IUT 1
CHENAVAS Jean	Physique IUT 1
CHOUTEAU Gérard	Physique IUT 1
CONTE René	Physique IUT 1
GOSSE Jean-Pierre	EEA.IUT 1
GROS Yves	Physique IUT 1
KUHN Gérard, (Détaché)	Physique IUT 1
MAZUER Jean	Physique IUT 1
MICHOULIER Jean	Physique IUT 1
MONLLOR Christian	EEA.IUT 1
PEFFEN René	Métallurgie IUT 1
PERRAUD Robert	Chimie IUT 1
PIERRE Gérard	Chimie IUT 1
TERRIEZ Jean-Michel	Génie Mécanique IUT 1
TOUZAIN Philippe	Chimie IUT 1
VINCENDON Marc	Chimie IUT 1

PROFESSEURS DE PHARMACIE

AGNIUS-DELORD Claudine	Physique	Faculté La Tronche
ALARY Josette	Chimie Analytique	Faculté La Tronche
BERIEL Hélène	Physiologie et Pharmacologie	Faculté La Tronche
CUSSAC Max	Chimie Therapeutique	Faculté La Tronche
DEMENGE Pierre	Pharmacodynamie	Faculté La Tronche
FAVIER Alain	Biochimie	C.H.R.G.
JEANNIN Charles	Pharmacie Galénique	Faculté Meylan
LATURAZE Jean	Biochimie	Faculté La Tronche
LUU DUC Cuong	Chimie Générale	Faculté La Tronche
MARIOTTE Anne-Marie	Pharmacognosie	Faculté La Tronche
MARZIN Daniel	Toxicologie	Faculté Meylan
RENAUDET Jacqueline	Bactériologie	Faculté La Tronche
ROCHAT Jacques	Hygiéne et Hydrologie	Faculté La Tronche
SEIGLE-MURANDI Françoise	Botanique et Cryptogamie	Faculté Meylan
VERAIN Alice	Pharmacie Galénique	Faculté Meylan

MEMBRES DU CORPS ENSEIGNANT DE MEDECINE

PROFESSEURS CLASSE EXEPTIONNELLE ET 1ère CLASSE

AMBLARD Pierre	Dermatologie	C.H.R.G.
AMBROISE-THOMAS Pierre	Parasitologie	C.H.R.G.
BEAUDOING André	Pédiatrie-Puericulture	C.H.R.G.
BEZEZ Henri	Orthopédie-Traumatologie	Hopital SUD
BONNET Jean-Louis	Ophtalmologie	C.H.R.G.
BOUCHET Yves	Anatomie	Faculté La Merci
	Chirurgie Générale et Digestive	C.H.R.G.
BUTEL Jean	Orthopédie-Traumatologie	C.H.R.G.
CHAMBAZ Edmond	Biochimie	C.H.R.G.
CHAMPETIER Jean	Anatomie-Topographique et Appliquée	C.H.R.G.
CHARACHON Robert	O.R.L.	C.H.R.G.
COLOMB Maurice	Immunologie	Hopital sud
COUDERC Pierre	Anatomie-Pathologique	C.H.R.G.
DELORMAS Pierre	Pneumophtisjologie	C.H.R.G.
DENIS Bernard	Cardiologie	C.H.R.G.
GAVEND Michel	Pharmacologie	Faculté La Merci
HOLLARD Daniel	Hématologie	C.H.R.G.
LATREILLE René	Chirurgie Thoracique et Cardiovasculaire	C.H.R.G.
LE NOC Pierre	Bactériologie-Virologie	C.H.R.G.
MALINAS Yves	Gynécologie et Obstétrique	C.H.R.G.
MALLION Jean-Michel	Médecine du Travail	C.H.R.G.
MICOUD Max	Clinique Médicale et Maladies Infectieuses	C.H.R.G.
MOURIQUAND Claude	Histologie	Faculté La Merci
PARAMELLE Bernard	Pneumologie	C.H.R.G.
PERRET Jean	Neurologie	C.H.R.G.
RACHAIL Michel	Hépato-Gastro-Entérologie	C.H.R.G.
DE ROUGEMONT Jacques	Neurochirurgie	C.H.R.G.
SARRAZIN Roger	Clinique Chirurgicale	C.H.R.G.
STIEGLITZ Paul	Anestésiologie	C.H.R.G.
TANCHE Maurice	Physiologie	Faculté La Merci
VIGNAIS Pierre	Biochimie	Faculté La Merci

PROFESSEURS 2ème CLASSE

BACHELOT Yvan	Endocrinologie	C.H.R.G.
BARGE Michel	Neurochirurgie	C.H.R.G.
BENABID Alim Louis	Biophysique	Faculté La Merci
BENSA Jean-Claude	Immunologie	Hopital Sud
BERNARD Pierre	Gynécologie-Obstétrique	C.H.R.G.
BESSARD Germain	Pharmacologie	ABIDJAN
BOLLA Michel	Radiothérapie	C.H.R.G.
BOST Michel	Pédiatrie	C.H.R.G.
BOUCHARLAT Jacques	Psychiatrie Adultes	Hopital Sud
BRAMBILLA Christian	Pneumologie	C.H.R.G.
CHIROSSEL Jean-Paul	Anatomie-Neurochirurgie	C.H.R.G.
COMET Michel	Biophysique	Faculté La Merci
CONTAMIN Charles	Chirurgie Thoracique et Cardiovasculaire	C.H.R.G.
CORDONNIER Daniel	Néphrologie	C.H.R.G.
COULOMB Max	Radiologie	C.H.R.G.
CROUZET Guy	Radiologie	C.H.R.G.
DEBRU Jean-Luc	Médecine Interne et Toxicologie	C.H.R.G.
DEMONGEOT Jacques	Biostatistiques et Informatique Médicale	Faculté La Merci
DUPRE Alain	Chirurgie Générale	C.H.R.G.
DYON Jean-François	Chirurgie Infantile	C.H.R.G.
ETERRADOSSI Jacqueline	Physiologie	Faculté La Merci
FAURE Claude	Anatomie et Organogénèse	C.H.R.G.
FAURE Gilbert	Urologie	C.H.R.G.
FOURNET Jacques	Hépato-Gastro-Entérologie	C.H.R.G.
FRANCO Alain	Médecine Interne	C.H.R.G.
GIRARDET Pierre	Anesthésiologie	C.H.R.G.
GUIDICELLI Henri	Chirurgie Générale et Vasculaire	C.H.R.G.
GUIGNIER Michel	Thérapeutique et Réanimation Médicale	C.H.R.G.
HADJIAN Arthur	Biochimie	Faculté La Merci
HALIMI Serge	Endocrinologie et Maladies Métaboliques	C.H.R.G.
HOSTEIN Jean	Hépato-Gastro-Entérologie	C.H.R.G.
HUGONOT Robert	Médecine Interne	C.H.R.G.
JALBERT Pierre	Histologie-Cytogénétique	C.H.R.G.
JUNIEN-LAVILLAUROY Claude	O.R.L.	C.H.R.G.
KOLODIE Lucien	Hématologie Biologique	C.H.R.G.
LETOUBLON Christian	Chirurgie Générale	C.H.R.G.
MACHECOURT Jacques	Cardiologie et Maladies Vasculaires	C.H.R.G.
MAGNIN Robert	Hygiène	C.H.R.G.
MASSOT Christian	Médecine Interne	C.H.R.G.
MOUILLON Michel	Ophtalmologie	C.H.R.G.
PELLAT Jacques	Neurologie	C.H.R.G.
PHELIP Xavier	Rhumatologie	C.H.R.G.
RACINET Claude	Gynécologie-Obstétrique	Hopital Sud
RAMBAUD Pierre	Pédiatrie	C.H.R.G.
RAPHAEL Bernard	Stomatologie	C.H.R.G.
SCHAERER René	Cancérologie	C.H.R.G.
SEIGNEURIN Jean-Marie	Bactériologie-Virologie	Faculté La Merci
SELE Bernard	Cytogénétique	Faculté La Merci
SOTTO Jean-Jacques	Hématologie	C.H.R.G.
STOEBNER Pierre	Anatomie Pathologique	C.H.R.G.
VROUSOS Constantin	Radiothérapie	C.H.R.G.

Monsieur Maurice TCHUENTE, Professeur à l'Université de Yaoundé, a dirigé mon travail et m'a judicieusement conseillé pendant ces quatres années. J'ai apprécié sa disponibilité, ses qualités scientifiques et humaines. Je lui adresse tous mes remerciements.

Je remercie Monsieur François ROBERT, Professeur à l'INPG, pour l'honneur qu'il me fait en présidant le jury de cette thèse. Je lui exprime ma reconnaissance de m'avoir souvent consacré de son précieux temps.

J'exprime toute ma reconnaissance à Monsieur Michel COSNARD, Professeur à l'ENS de Lyon, d'avoir accepté d'être rapporteur et de m'avoir encouragé, conseillé et aidé durant l'élaboration de cette thèse.

Je remercie Monsieur Patrice QUINTON, Directeur de Recherche à l'IRISA, Rennes, d'avoir accepté d'être rapporteur et de m'avoir conseillé plusieurs fois.

Mes vifs remerciements vont également à:

Monsieur Yves ROBERT, Chargé de Recherche au CNRS, d'avoir accepté de participer à ce jury et pour l'aide qu'il m'a apportée.

Madame Brigitte PLATEAU, Professeur à L'INPG, pour l'honneur qu'elle me fait en participant à ce jury.

Monsieur le Professeur Pierre Jean LAURENT qui m'a aidé à obtenir une bourse du Ministère de la Recherche et de Technologie.

Toute l'équipe d'Algorithmique Parallèle et Calcul Formel du laboratoire TIM3.

Je rèmercie les membres du service de reprographie pour l'excellente qualité de leur travail.

Table des matières

Chapitre 1 3

INTRODUCTION AUX ALGORITHMES
ET ARCHITECTURES SYSTOLIQUES

1.1. Exemple 3
1.2. Principes de base des architectures systoliques 5
1.3. Optimisation des circuits synchrones 6
1.4. Méthode de projection des dépendances 13
1.5. Validation des algorithmes systoliques 16
1.6. Quelques résultats et réalisations 17

Partie 1 19

ETUDE DU RESEAU LINEAIRE

Chapitre 2 20

RESEAUX UNIDIRECTIONNELS SANS
SIGNAUX DE CONTROLE

2.1. Introduction 20
2.2. Notations et définitions de base 22
2.3. Formulation géométrique 26
2.4. Produit de matrices denses: formules analytiques 49
2.5. Résultat de complexité pour $x \geq n+2$ 53
2.6. Produit de matrices triangulaires 56
2.7. Influence du nombre de canaux 62

Chapitre 3 72

RESEAUX LINEAIRES AVEC SIGNAUX
DE CONTROLE

3.1. Introduction 72
3.2. Algorithmes optimaux en dimension 2 73
3.3. Algorithmes unidirectionnels totalement pipelinés 78
3.4. Algorithmes unidirectionnels partiellement pipelinés 84
3.5. Algorithmes bidirectionnels 87

Chapitre 4 91

ELIMINATION DE GAUSS SUR UN RESEAU UNIDIRECTIONNEL

4.1. Introduction 91
4.2. Structure générale du réseau 92
4.3. Solution sur réseau de taille n-1 93
4.4. Solution modulaire 94

Partie 2 98

SIMULATION ET VALIDATION DES ALGORITHMES SYSTOLIQUES

Chapitre 5 100

SIMULATION DES ALGORITHMES SYSTOLIQUES

5.1. Description des algorithmes systoliques 100
5.2. SISYC: SImulation of SYstolic Circuits 106
5.3. Opérations irrégulières 111
5.4. Exemples de simulations de réseaux systoliques 112
5.5. Conclusion 125

Chapitre 6 126

VALIDATION DES ALGORITHMES SYSTOLIQUES

6.1. Introduction 126
Exemple 6.1 128
6.2. Transformation des algorithmes decrits par des SERT 131
Exemple 6.2 134
6.3. Le logiciel SISYC2 137
6.4. Exemples d'utilisation de SISYC2 143

CONCLUSION 152

REFERENCES 153

ANNEXE 1 160
GRAMMAIRE DE SISYC2

ANNEXE 2 165
PROGRAMME PRINCIPAL DE SISYC2

Chapitre 1

INTRODUCTION AUX ALGORITHMES ET ARCHITECTURES SYSTOLIQUES

Le concept d'architecture systolique a été introduit en 1978 par H.T. Kung et C.E. Leiserson [LeK 78], dans le but d'accélérer le traitement de problèmes qui nécessitent beaucoup de calculs. De manière informelle, ils définissent une architecture systolique comme étant un réseau de processeurs spécialisés (dits cellules), localement interconnectés et fonctionnant en mode synchrone. Leur idée est d'utiliser une architecture systolique comme périphérique d'un calculateur hôte de type conventionnel (figure 1.1).

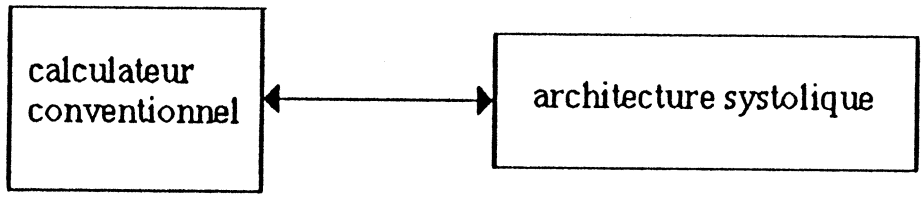


Figure 1.1. Architecture systolique connectée à un calculateur hôte.

Avant d'expliciter les principes de base des réseaux systoliques, nous traitons un exemple simple afin de permettre au lecteur de se familiariser avec ce modèle. Il s'agit d'une architecture systolique destinée à effectuer le produit C = A.B, de deux matrices de taille 3x3 (exemple tiré de [MeT 85]).

1.1. Exemple

La figure 1.2 illustre le réseau conçu pour cette tache et qui doit être relié à une machine hôte via un réseau d'interconnexion. Sur ce schéma, les cellules sont représentées par des carrés et les bus de communication, par des flèches.

A tout instant, chaque cellule reçoit les données de ses voisines en entrée et, après avoir effectué la transformation indiquée par la figure 1.2b, elle les envoie à ses voisines en sortie. Au bout de 8 cycles d'horloge, tous les $c_{i,j}$ sont calculés et se trouvent en dehors du réseau. Notons que les données, une fois lues par le réseau, sont entièrement gérées par celui-ci pendant tout le déroulement de l'algorithme. Il convient aussi de remarquer que, à partir du mode de rangement de données dans la mémoire hôte, le réseau d'interconnexion entre celui-ci et l'architecture systolique doit éventuellement effectuer des réarrangements pour les faire arriver dans les bons ports d'entrée du réseau systolique.

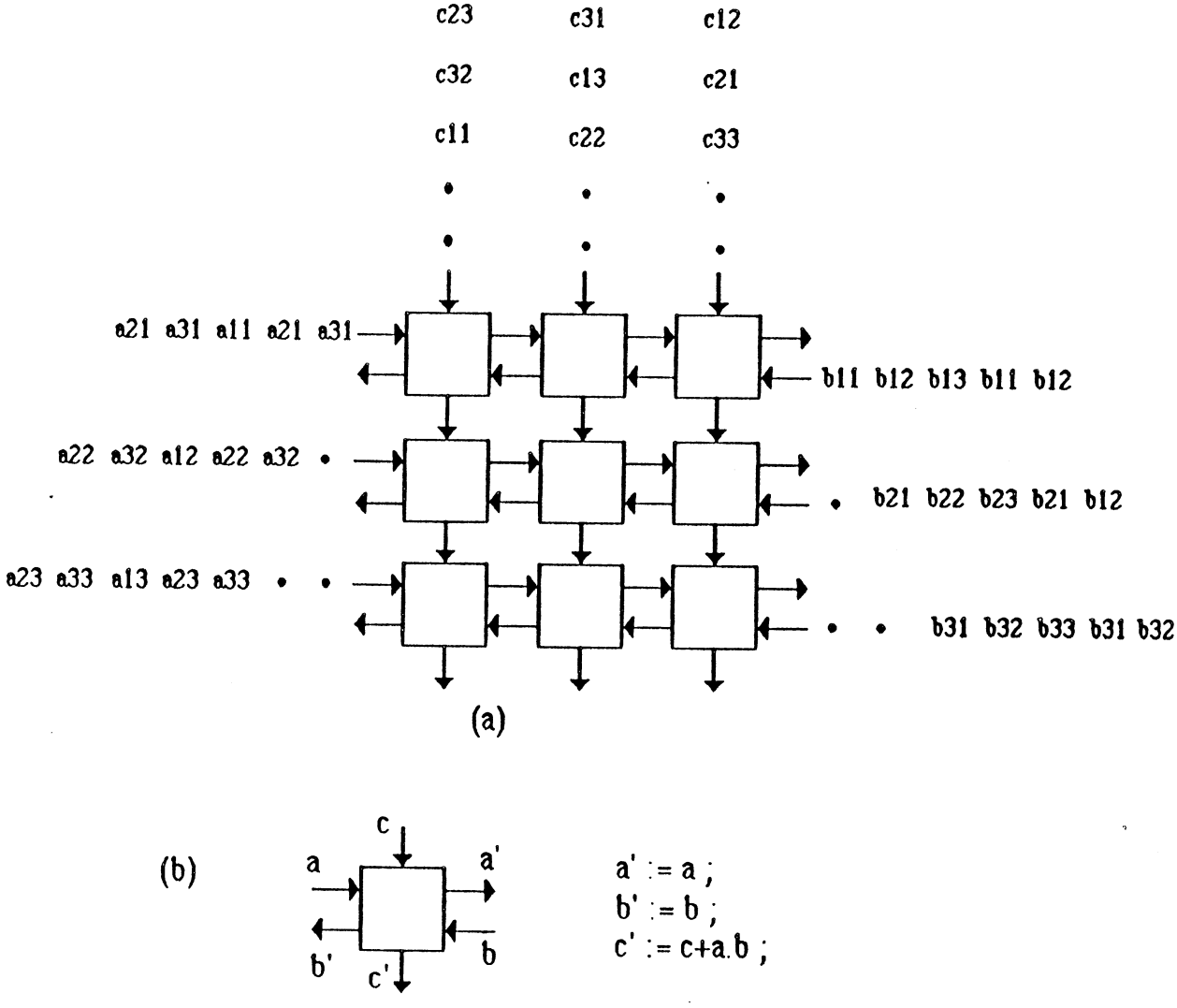


Figure 1.2: (a) réseau systolique pour le produit matriciel
(b) cellule de base

1.2. Principes de base des architectures systoliques

Malgré l'absence d'une définition exacte du mot systolique, nous essayons dans ce qui suit de présenter très brièvement les principes de base des architectures systoliques. En d'autre termes, nous expliquons pourquoi un réseau de cellules spécialisées, régulièrement interconnectées et fonctionnant en mode synchrone est intéressant.

1.2.1. Architectures spécialisées

Les technologies VLSI offrent des possibilités de réalisation de circuits intégrés à faible coût: d'où l'idée de concevoir une architecture systolique destinée à une application bien définie. Ainsi, contrairement aux machines parallèles universelles, elle tire partie des caractéristiques du problème pour lequel elle a été conçue. Par conséquent, de bonnes performances peuvent être atteintes.

1.2.2. Synchronisme global

Selon la définition de Kung et Leiserson, une machine systolique fonctionne en mode synchrone, en ce sens que les cellules évoluent en parallèle sous le contrôle d'une horloge globale. Ceci facilite énormément la mise en oeuvre des protocoles de communication. Le synchronisme global est une spécificité de tous les réseaux systoliques (même le warp [AAG 86, BCC 87] qui est une machine systolique universelle fonctionne en mode synchrone).

1.2.3. Pipeline et accès mémoire

Une même donnée est transformée plusieurs fois à l'intérieur du réseau sans être lue autant de fois par ce dernier. Par exemple dans la figure 2, l'élément $c_{1,1}$ accumule $a_{1,1}.b_{1,1}$ dans la cellule (1,1) et passe, au top suivant, à la cellule (2,1) où il accumule $a_{1,2}.b_{2,1}$ sans être lu une deuxième fois. Ce mode de fonctionnement minimise les accès mémoire, qui constituent un goulot d'étranglement pour les machines basées sur le principe de Von Neumann.

Par ailleurs, les problèmes qui se prêtent bien à l'approche systolique sont ceux qui nécessitent des traitements répétitifs sur la même variable. De tels problèmes sont baptisés en anglais ' compute-bound ', car ils induisent des réseaux où le nombre des calculs prime largement sur celui des communications avec l'hôte.

1.2.4. Modularité et localité

Le réseau de la figure 1.2 peut être étendu afin de calculer le produit de deux matrices de taille supérieure (4x4 par exemple). Pour cela, il suffit d'adjoindre au réseau initial une ligne et une colonne de cellules et de disposer les données de façon à réaliser toutes les rencontres ($c_{i,j}$, $a_{i,k}$, $b_{k,j}$).

Ce caractère de modularité est très important car en conservant la même structure pour la cellule de base, on peut construire des réseaux pour le traitement de problèmes de taille quelconque. D'autre part, les cellules étant interconnectées de manière locale, la longueur des bus de communication entre cellules reste constante.

Nous allons maintenant présenter les deux principales méthodes développées dans la littérature, pour la synthèse automatique ou semi automatique des algorithmes et architectures systoliques.

1.3. Optimisation des circuits synchrones

L'idée de base de la méthode de Leiserson et Saxe [LeS 78], est d'optimiser les circuits synchrones en minimisant le cycle d'horloge. Cette méthode est issue de la théorie des graphe. Elle prend comme point de départ, un circuit décrit en terme de graphes, et le transforme, sans modifier sa topologie, en un circuit systolique .

1.3.1. Représentation abstraite des circuits

Un circuit est formellement représenté par un multigraphe noté
$G = (V, E, f_+, f_-, d, w, v_h)$, où

- L'ensemble des sommets V représente les éléments fonctionnels (de nature combinatoire) du circuit
- Le noeud v_h représente la structure hôte.
- L'ensemble des arcs E représente les connexions entre les éléments fonctionnels du circuit.
- f_+ et f_- sont deux fonctions de E dans V représentant respectivement la source et la destination d'un arc.
- d est une fonction de V à valeurs positives: d(v) représente le delai de propogation (ou le temps de cycle) de la cellule v.

- w est une fonction de E à valeurs entières représentant le nombre de registres (ou latches) sur chaque connexion: w(e) est le nombre de bascules que doit traverser une donnée pour aller du sommet $f_-(e)$ au sommet $f_+(e)$.

Un circuit est modelisé par un multigraphe, et non par un graphe car si on a deux éléments fonctionnels u , v , ayant entre eux deux canaux de communication de même direction (u,v) , portant des signaux différents, alors ces canaux seront représentés par deux arcs e, e' ayant la même origine u et la même destination v.

Les auteurs considérent uniquement les circuits synchrones. Un circuit est synchrone si tout cycle élémentaire du graphe qui le représente a au moins un registre.

Entre deux tops d'horloge, chaque donnée se propage le long de connexions, jusqu'à ce qu'elle soit bloquée par un registre.

Nous illustrons la méthode de Leiserson et Saxe sur le problème du produit de convolution non récursive. Etant donnés k coefficients reéls (a_i, $1 \leq i \leq k$) , et une suite réelle (x_i, $i \geq 0$), la convolution consiste à calculer la suite (y_i, $i \geq k$), où les y_i sont définies par les relations:

$$y_i = a_1.x_i + a_2.x_{i-1} + \ \ldots \ + a_k.x_{i-k+1}.$$

Pour $k = 4$, un circuit synchrone effectuant cette opération est donné par la figure 1.3.

La figure 1.4 est la représentation en terme de graphe du circuit de convolution de la figure 1.3. Les noeuds sont les v_i, $1 \leq i \leq 4$, et le sommet v_h représente la structure hôte. Les arcs sont représentés par des flèches, et les entiers sur les arcs représentent les poids de ces derniers, c'est à dire le nombre de registres qu'ils portent. Sur les arcs du bas (cf. figure 1.3) , il y avait un registre, donc le poids de chaque arc du bas est 1. Par contre les arcs du haut ont des poids nuls.Par conséquent les données qui circulent sur les arcs du haut, vont de v_h à v_4 en un seul cycle de temps. En conséquence, le temps de cycle doit être supérieur au temps de la traversée d'une donnée du chemin joignant v_h à v_4 , c'est-à-dire $d(v_1) + d(v_2) + d(v_3) + d(v_4)$.

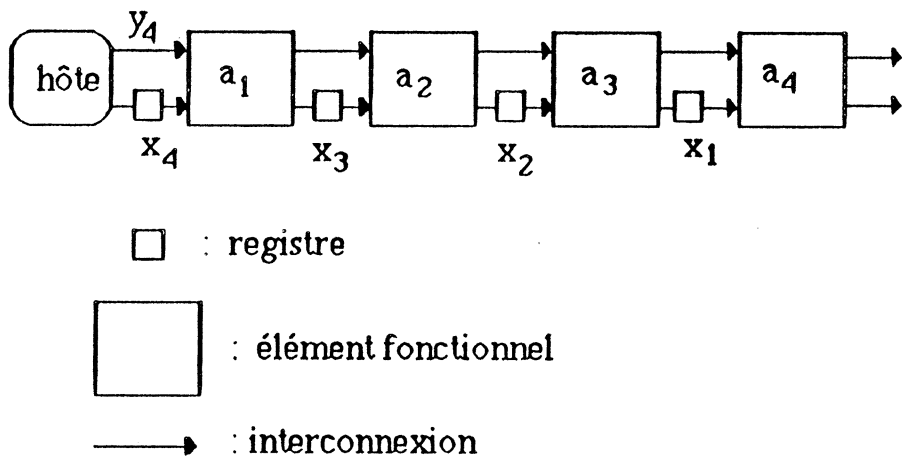


Figure 1.3: circuit synchrone pour la convolution

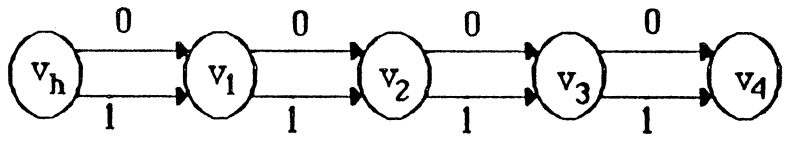


Figure 1.4: multigraphe représentant le circuit de convolution

Le but de la méthode de Leiserson et Saxe est d'effectuer une nouvelle répartition des registres afin d'optimiser le temps de cycle du cicuit consideré. Par exemple, partant du circuit de la figure 1.3, elle permet de trouver le circuit représenté par figure 1.5. Chaque interconnexion de ce circuit a au moins un registre. Un tel circuit est dit systolique. On vérifie aisément que ce circuit réalise aussi la convolution:

- y_4 est délivré après 4 cycles.
- les y_i sont espacés de deux cycles

Comme le temps de cycle de ce réseau systolique est $d = \max (d(v_i))$, on voit qu'en moyenne ce réseau systolique est 4 fois plus rapide que le circuit synchronne de la figure 1.2.

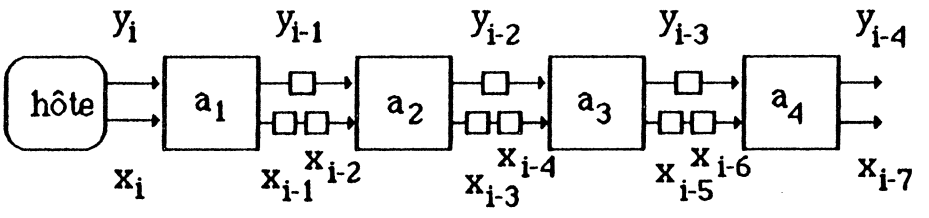


Figure 1.5: circuit systolique pour la convolution.

1.3.2. Méthodologie de systolisation

Cette technique due à Leiserson et Saxe, pemet de systoliser un circuit synchrone. L'idée de base est de réallouer les registres. En effet on peut , sans modifier le fonctionnement du système, enlever un registre à chacun des arcs entrants, et en rajouter un sur chacun des arcs sortants de l'un des sommets du circuit (voir figure 1.4).

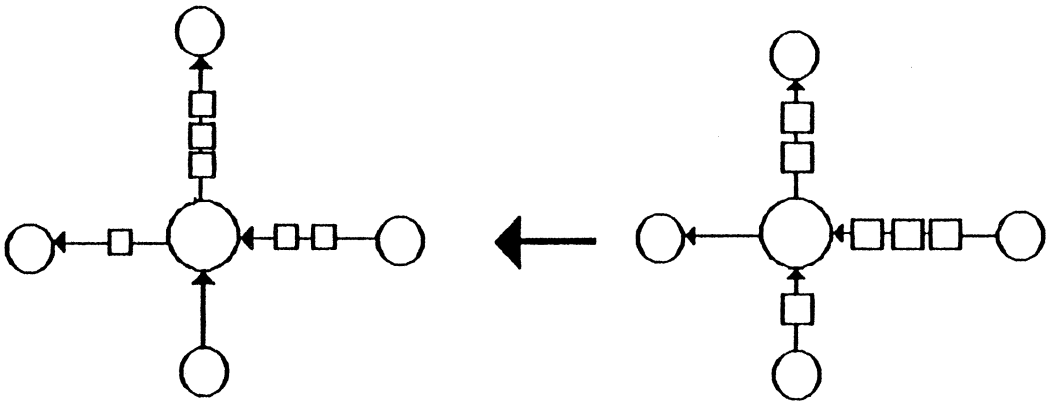


Figure 1.6: une transformation du graphe

Cette transformation ne modifie pas le nombre total de registres sur les cycles élémentaires. En conséquence, une condition nécessaire pour que de telles transformations converstissent un circuit synchrone représenté par un multigraphe G, en un circuit systolique est la suivante:

> Pour chaque cycle élémentaire de G, le nombre de registres doit être supérieur ou égal au nombre d'arcs .

En effet, chaque arc du réseau systolique doit porter au moins un registre. Dans tout cycle c du graphe associé au réseau systolique, le nombre w(c) de registres est supérieur ou égal au nombre d'arcs. Le résultat vient donc du fait que la transformation de base laisse invariant w(c).

Une autre formulation de cette condition est la suivante: Pour convertir un système synchrone représenté par le multigraphe G en un système systolique, il est nécessaire que

(C) il n'existe pas de circuit élémentaire dans G-1 dont le poids soit négatif.

G-1 désigne simplement le multigraphe obtenu à partir de G en soustrayant un registre sur chaque arc. Les poids de G-1 peuvent donc être négatifs.

Par exemple le multigraphe ci-dessous ne vérifie pas la condition (C). En effet, le poids du circuit élémentaire v_1, v_2, v_1 , dans G-1, est égal à -1.

multigraphe G multigraphe G-1.

Figure 1.7: exemple de circuits avec un poids négatif.

Lorsque G présente des circuits élémentaires de poids négatifs, on cherche un entier positif k tel qu'on puisse systoliser le multigraphe kG. Ce dernier est obtenu à partir de G en multipliant les poids de G par k. Il représente un circuit synchrone rélisant la même fonction que G, mais avec une vitesse divisée par k.

Nous exposons maintenant les étapes du procédé de systolisation:

Etape 1: Détermination du plus petit entier k tel que le multigraphe kG-1 vérifie la condition C.

Etape 2: Calcul de la fonction r de V à valeurs entières définie par

r(v) = Poids minimun w(c) d'un chemin c allant du sommet v au sommet v_h dans le multigraphe kG-1.

Etape 3: Le graphe du circuit systolique est $G_r = (V, E, f_+, f_-, d, w_r, v_h)$, où

$$w_r(e) = w(e) + r(f_+(e)) - r(f_-(e)).$$

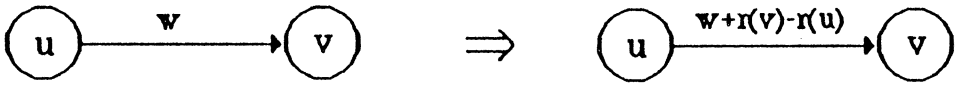


Figure 1.8

La figure 1.9, montre les étapes successives pour la systolisation du circuit de la convolution. Il convient de signaler que nous avons volontairement représenté l'hôte deux fois, pour éviter les arcs qui retournent de v_4 à v_h. D'autre part, le nombre de registres entre v_4 et v_h n'influe pas sur le résultat.

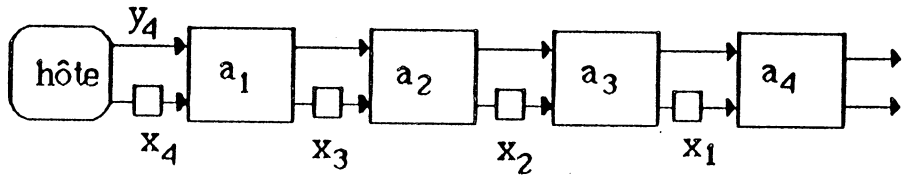
Le circuit systolique obtenu est le même que le circuit systolique de départ, à la différence qu'il comporte un registre supplémentaire sur chacun de ses arcs.

C'est un résultat généralisable aux circuits unidirectionnels, c'est à dire où tous les données circulent dans le même sens:

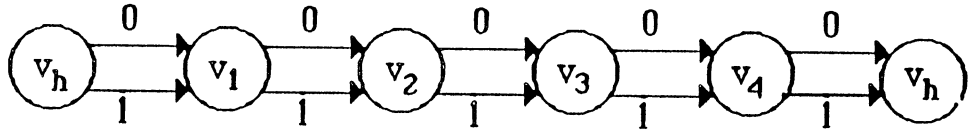
> On peut ajouter ou retrancher le même nombre de registres à tous les arcs d'un circuit unidirectionnel sans modidfier son fonctionnement. Cependant, le temps d'exécution du nouveau circuit est :
>
> $$T' = T + x.S$$
>
> où
>
> T = le temps d'exécution du circuit de départ,
> x = le nombre de registres qu'on ajoute sur chaque arc (x peut être positif ou négatif).
> S = le nombre d'éléments fonctionnels du circuit.

Nous nous sommes servi de ce résultat pour la construction des algorithmes de produit matriciel développés dans les chapitres 2 et 3.

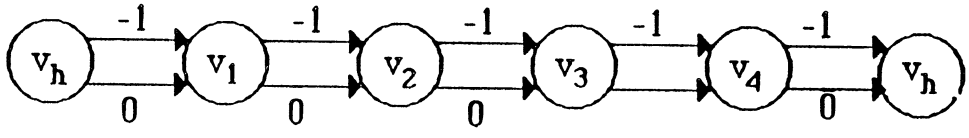
Pour plus de détails sur cette méthode, nous renvoyons le lecteur intéressé au polycopié de Y. Robert [Rob 87]. On y trouve une extension de la méthode à l'aide de la technique du partitionnement due à Y. Robert et M. Tchuente [RoT86].



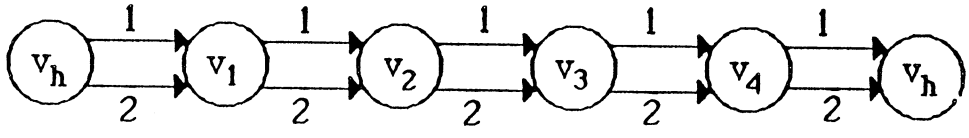
(a) Circuit synchrone pour la convolution.



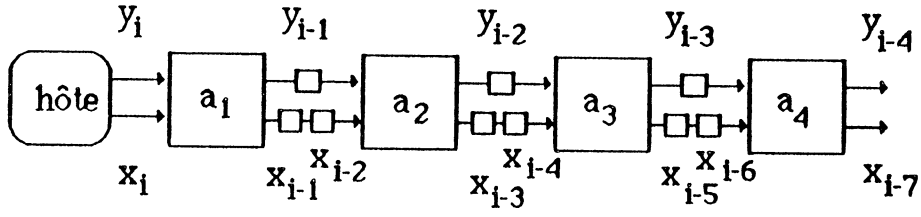
(b) Ce multigraphe G ne comporte aucun circuit élémentaire. Par conséquent, la condition (C) est automatiquement vérifiée pour G-1. Donc c'est le graphe G qu'on systolise (k=1).



(c) Le multigraphe G-1 permettant de calculer les r(v):
$r(v_h) = 0$, $r(v_4) = -1$, $r(v_3) = -2$, $r(v_2) = -3$, $r(v_4) = -4$.



(d) G_r = le résultat de la méthode de systolisation = 2G - 1



(e) Circuit systolique associé à G_r.

Figure 1.9

1.4. Méthode de projection des dépendances

Nous présentons dans ce pragraphe, la méthode de projection des dépendances développée notamment par P. Quinton et son équipe [Qui 84, Qui 85]. Nous nous sommes servis de cette méthode dans les chapitres sur le produit matriciel.

La méthode prend comme point de départ une solution décrite sous forme d'un système d'équations récurrentes uniformes (SERU en abrégé , voir [Quin 85] pour la définition), et délivre en sortie des architectures systoliques capables d'exécuter ce SERU. Nous l'illustrons sur l'exemple du produit de deux matrices denses. On procéde alors en trois étapes comme suit:

Etape 1: On associe les calculs aux points d'un domaine de Z^3.

Partant des équations usuelles pour le produit matriciel

$$c_{i,j} = a_{i,1}.b_{1,j} + ... + a_{i,n}.b_{n,j}$$
$$1 \leq i,j,k \leq n$$

On déduit une formulation sous forme d'équations récurrentes uniformes:

$$c(i,j,k) = c(i,j,k-1) + a(i,j,k).b(i,j,k)$$
$$a(i,j,k) = a(i,j-1,k)$$
$$b(i,j,k) = b(i-1,j,k)$$
$$1 \leq i,j,k \leq n.$$

Avec les conditions aux limites:

$$c(i,j,0) = 0 \; ; \quad c(i,j,n) = c_{i,j}$$
$$a(i,0,k) = a_{i,k} \; ;$$
$$b(0,j,k) = b_{k,j}$$

Etape 2: Construction du graphe des dépendances.

Les équations récurrentes ci-dessus font apparaître un domaine de calcul

$$D_n = \{ (i,j,k) \mid 1 \leq i , j , k \leq n \}.$$

Par ailleurs, l'ensemble des vecteurs de dépendances est

$$U = \{ u_a = (0,1,0), u_b = (1,0,0), u_c = (0,0,1) \}.$$

En effet,

- la valeur de c au point (i,j,k) dépend de la valeur de c au point (i,j,k-1)
- la valeur de a au point (i,j,k) dépend de la valeur de a au point (i,j-1,k)
- la valeur de b au point (i,j,k) dépend de la valeur de b au point (i-1,j,k)

On obtient ainsi le graphe $G_n = (D_n,U)$ illustré dans la figure 1.10, pour n = 3.

Etape 3: Pour obtenir un réseau bidimensionnel il suffit d'appliquer à G une transformation spatio-temporelle qui respecte les relations de dépendance entre les calculs. Cette transformation est projection du domaine D_n selon une direction déterminée par un vecteur u.

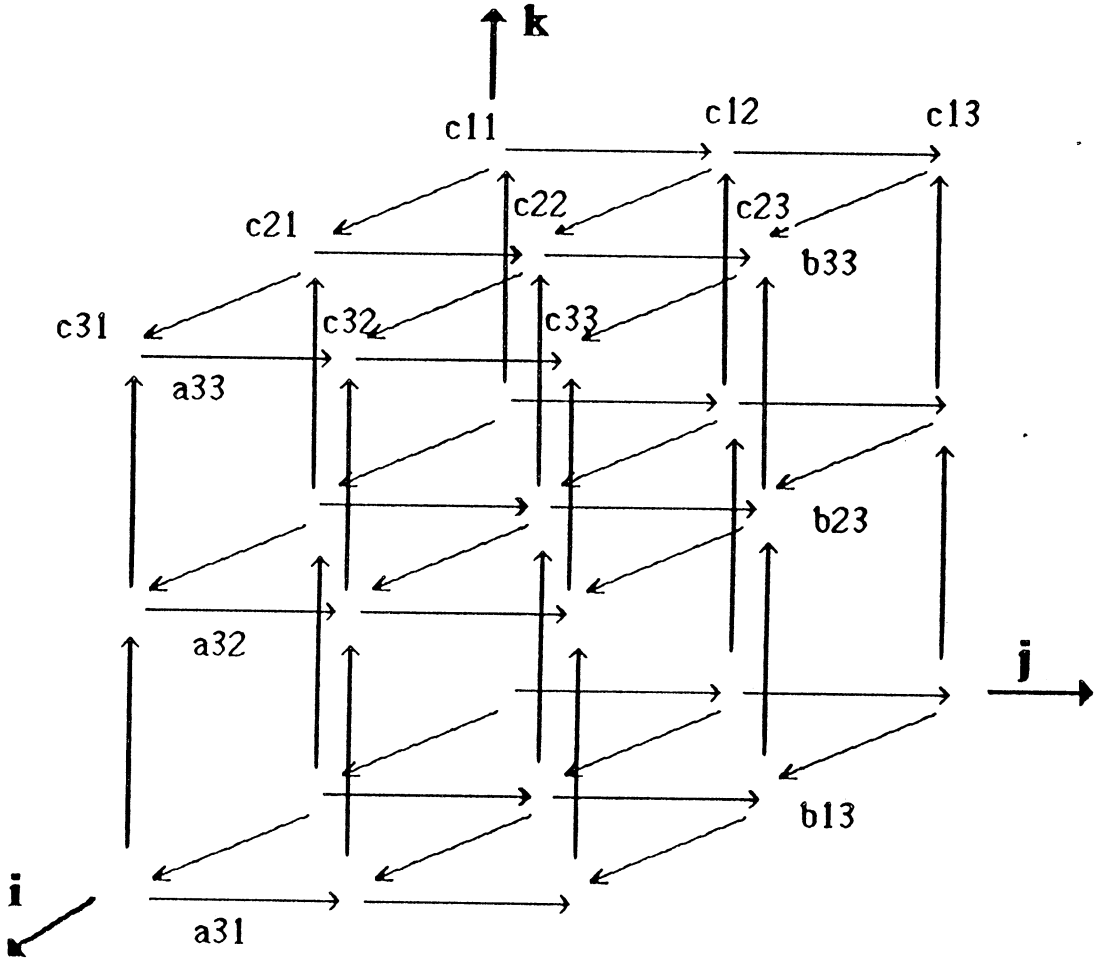


Figure 1.10: Domaine D_n , n = 3.

Solution 1: u = (0, 0, 1): On obtient le réseau de la figure 1.11.

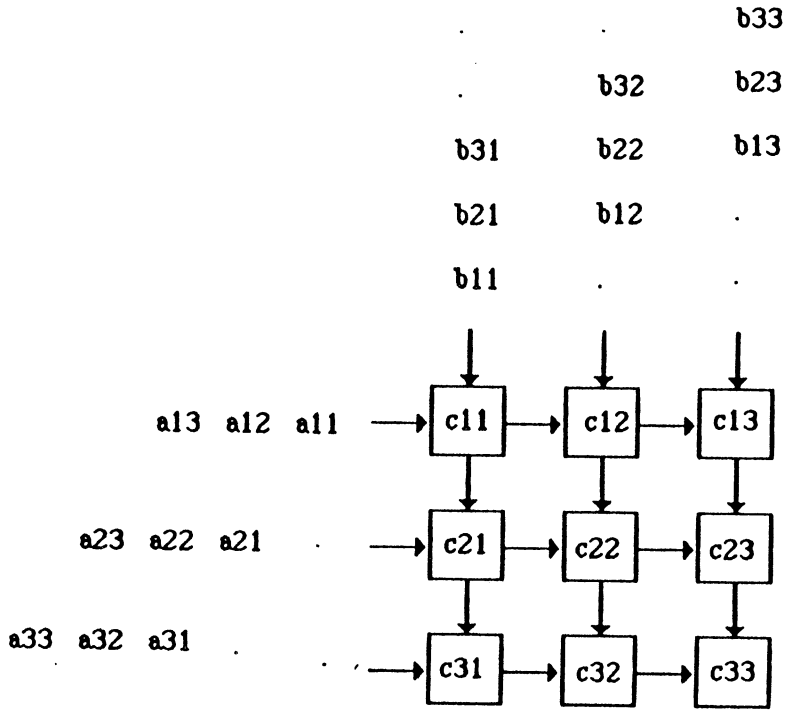


Figure 1.11: projection suivant (0, 0, 1)

Solution 2: u = (1, 1, 0): On obtient le réseau de la figure 1.12.

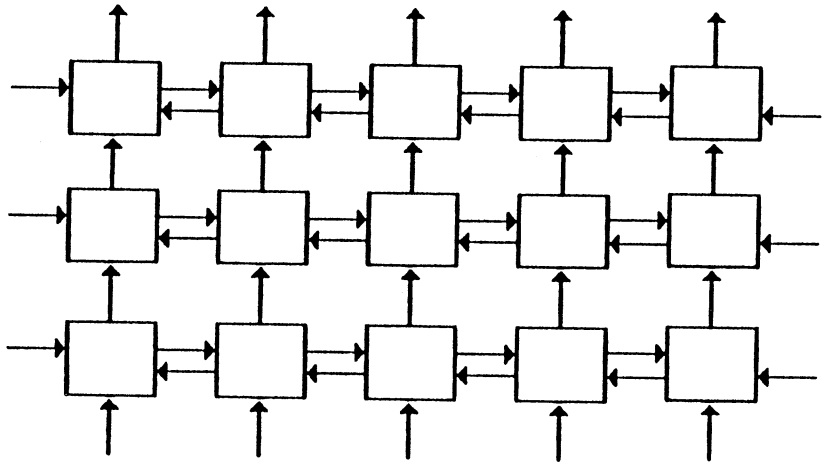


Figure 1.12: projection suivant (1, 1, 0).

Le lecteur interessé trouvera tous les détails de cette méthode dans la thèse de B. Joinnault et P. Gachet [JoG 87], dans laquelle ils généralisent la méthode de projection des dépendances. Ils proposent en plus une méthode systématique pour la synthèse des SERU.

1.5. Validation des algorithmes systoliques

Malgré les nombreuses méthodologies, la plupart des algorithmes systoliques publiés ont été conçus de manière intuitive. Par conséquent, ils doivent faire l'objet d'une analyse détaillée, afin de s'assurer de leur validité. Il est donc nécessaire d'avoir les outils qui permettent de vérifier si un réseau systolique résoud correctement, un problème donné. Parmi le peu de techniques de vérification proposées dans la littérature citons:

- E.Tiden [Tid 84], qui a démontré que le réseau proposé par Heller et Ipsen [Hel 83], réalise correctement la décomposition QR d'une matrice A. La preuve est faite par récurrence sur la taille du réseau. Cette technique (si elle est généralisable) est valable uniquement pour les réseaux modulaires. Elle serait toutefois, intéressante si elle pouvait être automatisée.

- D'autres auteurs [AFE 88 , MiC 81], proposent une méthode basée sur la logique temporelle. Cette méthode consiste à définir un système axiomatique permettant de prouver formellement des propriétés de programmes parallèles. Ces techniques n'exploitent pas les spécificités des architectures systoliques.

- Récemment, Probst et Li [PrL 88] ont proposé une méthode de preuve utilisant les assertions pour la spécification abstraite des calculs synchrones. Elle consiste à dériver l'invariant du réseau et à dresser une table dans laquelle sont résumées ses opérations. Cette technique appliquée à la preuve de la pile systolique [GuL 82], ne montre ni son intérêt ni sa puissance, d'autant plus qu'elle ne conduit pas à des réalisations automatiques.

- Melhem et Reinboldt [MeR 84, Mel 85], proposent de représenter les flots de données par des suites et ensuite, de modéliser les opérations des cellules à l'aide d'opérateurs algébriques. Ainsi, le fonctionnement du réseau en question se résume par un système d'équations, sur les suites, qu'il faut résoudre pour exprimer les sorties du réseau en fonction de ses entrées. Cependant, il n'existe pas de méthode générale pour la résolution de tels systèmes. On peut montrer qu'il est impossible de résoudre un système , par substitution, lorsque son graphe de calcul est cyclique. Ceci restreint beaucoup la classe des réseaux systoliques étudiés.

Ce modèle a conduit naturellement à un simulateur des réseaux systoliques SCE que Melhem a développé [Mel 83, Mel 85]. Ce langage n'autorise aucune forme de conditionnel. Nous l'avons simplifié et généralisé en un simulateur (numérique) baptisé SISYC. Nous exposons ses principes dans le chapitre 5.
Signalons que des langages temps réels beaucoup plus élaborés que SISYC existent. Il s'agit de LUSTRE [Ber 86] développé à Grenoble et de SIGNAL [Gau 84, GuB 86] développé à Rennes. Ces deux langages jumeaux sont fondés sur le principe de la représentation des données par des suites. SISYC se distingue d'eux par sa simplicité et par la façon dont il a été conçu.

Notre approche pour la validation des algorithmes systoliques (qui fait l'objet du chapitre5) consiste à séquentialiser l'algorithme systolique. Nous procédons en deux étapes. D'abord, nous décrivons le réseau à l'aide des notations introduites dans le modèle de Melhem. Ensuite, nous condensons cette description. L'interêt de cette approche est qu'elle tire partie du fait qu'une variable subit plusieurs transformations sans être lue, à chaque fois, par le réseau.

Dans cette approche, la validation d'un algorithme systolique revient à montrer qu'il réalise la même opération qu'un algorithme séquentiel donné. Il suffit donc, d'établir l'équivalence entre l'algorithme fourni par cette technique et celui visé pour pouvoir se prononcer sur la validité de l'algorithme systolique en question. On ramène ainsi la preuve de correction d'un algorithme systolique à celle d'un algorithme séquentiel.

Son automatisation nécessite des calculs symboliques sur les indices. Pour contourner ce problème, nous avons écrit le logiciel SISYC2, qui est basé sur le principe de séquentialisation et qui calcule la trace d'un réseau systolique. Cette trace peut être directement interprétée par un système de calcul formel tel que Reduce, afin d'obtenir des expressions formelles qui expriment les sorties du réseau en fonction de ses entrées.

1.6. Quelques résultats et réalisations

Nous terminons ce chapitre, par la présentation de quelques résultats obtenus dans le domaine du systolique. Nous commençons par les réalisations de circuits (ou machines) systoliques, ensuite nous passons aux résultats obtenus en algorithmique systolique.

1.6.1. Circuits systoliques

Nous passons en revue quelques circuits systoliques prototypes ou opérationnels à l'heure actuelle [Rob 87].

- Le processeur systolique GAPP (pour Geometric Arithmetic Parallel Processor) de NCR apparu sur le marché en 1984. Il contient une grille rectangulaire de 6 x 12 microprocesseurs 1 bit, dont chaque élément peut communiquer avec ses voisins.
- Le réseau de convolution linéaire de ESL
- Le processeur systolique pour le FFT de TWR
- Le correlateur de Marconi Electronic Devices
- Le réseau bidimentionnel 64 x 64 de GEC

- La cellule systolique programmable pour l'algébre linéaire (SCALA) conçue au Laboratoire TIM3 (Grenoble).
- Le PSC (Programmable Systolic Chip) conçu à l'université de Carnegie Mellon
- Le WARP conçu à l'université de Carnegie Mellon. Il se présente comme un réseau linéaire de quelque dizaines de cellules non spécialisées.

1.6.2. Quelques références en algorithmique systoliques.

De nombreux problèmes ont fait l'objet d'une étude systolique, comme en témoigne le nombre d'articles portant sur les algorithmes systoliques. Plusieurs domaines sont concernés, dont le traitement d'images, le calcul matriciel, et la combinatoire.

- Convolution:
  [DoV 87 , KuL 80, KuS 85 , Qui 84, RoT 81 , RoT 85]
- Produit matricel:
  [FuV 84, GKT 79, Kat 80, MeT 85, MeT 88, PRV 80, QJG 86, RaV 84, Var 85]
- Factorisation des matrices:
  [BBK 84, BPS 88, CDM 86, Del 82, GeK 81, HeI 83 , LoT 88, RoT 85, RoT 85b , Sam 82]
- Inversion d'une matrice:
  [LiW 84, Sak 87]
- Le problème de mots:
  [MeT 86, NKY 84, RoT 85a]
- Calcul des chemins dans un graphe:
  [KuL 85, RoT 85 , RoT 86]
- Pile systolique:
  [GuL 82]
- Programmation dynamique :
  [GKT 79, LoT 88].

Pour plus de détails sur ces algorithmes et sur d'autres algorithmes , le tri, ..., nous renvoyons le lecteur interessé aux excellents ouvrages de M. Tchuente [Tch 88] et de P. Quinton et Y. Robert [QuR 88].

Dans ce cadre d'algorithmique systolique, nous proposons de nouveaux algorithmes pour le produit matriciel sur les réseaux linéaires. Ces algorithmes sont conçus à l'aide d'une formulation combinatoire, de ce problème, similaire à celle due à L. Melkemi et M. Tchuente [MeT 85] pour les réseaux réctangulaires. Nous présentons ces résultats dans les chapitres 2 et 3.

Chapitre 2

RESEAUX UNIDIRECTIONNELS SANS SIGNAUX DE CONTROLE

2.1. Introduction

Dans ce chapitre, nous étudions les réseaux linéaires, composés de cellules identiques interconnectées comme l'indique la figure 2.1. Nous dirons que c'est un réseau systolique unidirectionnel puisque tous les flots de données ont le même sens.

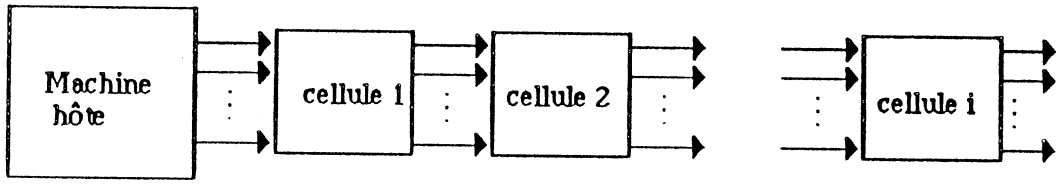


Figure 2.1. réseau unidirectionnel

L'intérêt de tels réseaux réside dans leur régularité et leur nombre constant d'entrées/sorties qui limite les coûts de communication avec la machine hôte. En effet, seules les cellules extrêmes communiquent avec l'hôte. D'autre part, ces réseaux sont adaptés à la tolérance aux fautes de conception [KuM 84, HuA 84].

Nous proposons des algorithmes pour le produit de deux matrices denses d'ordre n. La difficulté du problème provient notamment du fait que les méthodologies basées sur le principe des projections des dépendances, ne peuvent pas aboutir directement à des algorithmes systoliques sur des réseaux linéaires. En effet, la projection d'un graphe des dépendances de dimension k donne un réseau de dimension k-1, qui n'est linéaire que si k=2, alors que le graphe des dépendances du produit matriciel est de dimension 3.

Dans cette partie A, B et C désignent trois matrices d'ordre nxn et α un entier quelconque satisfaisant $1 \leq \alpha \leq n$.

Nous exhibons des algorithmes pour effectuer le produit C= A.B, et qui s'exécutent sur un réseau unidirectionnel, composé de cellules combinatoires, ayant chacune un registre local de longueur x = $O(n/\alpha)$. Les performances atteintes par ces algorithmes sont en $O((\alpha^2+\alpha+1/\alpha)n)$ pour la surface du réseau et en $O((\alpha+1+1/\alpha^2)n^2)$ pour le temps de calcul. Si les matrices A et B sont triangulaires inférieures, alors la surface requise pour le réseau est de l'ordre de $O(\alpha^2 n)$ et le temps d'exécution est en $O(\alpha n^2)$. Lorsque x = n+2, nous montrons que l'algorithme obtenu est optimal en temps et en surface, si on s'astreint à lire chaque donnée une seule fois.
Nous étudions aussi l'influence du nombre d'entrées/sorties simultanées sur la complexité du problème.

Soit donc à calculer le produit C = A.B, de deux matrices carrées d'ordre n, sur un réseau linéaire unidirectionnel. Nous nous imposons trois contraintes:

- Le réseau doit être uniforme, composé de cellules élémentaires ayant la même structure. Une telle cellule peut effectuer une seule accumulation c := c + a.b entre deux tops d'horloge.

- Le réseau doit être modulaire, cela signifie que la structure de la cellule de base doit être indépendante de la taille des matrices à multiplier. Ceci permet donc à partir d'un petit réseau, de construire un réseau pour le produit de grandes matrices en augmentant uniquement le nombre de cellules sans modifier leur structure. Comme seules les cellules extrêmes communiquent avec l'extérieur, la modularité entraîne que le nombre d'entrées/sorties simultanées que peut effectuer le réseau est indépendant de n.

- Pour $1 \leq i,j,k \leq n$, chaque donnée $a_{i,k}$ ou $b_{k,j}$ est lue une seule fois par le réseau. Par ailleurs, chaque $c_{i,j}$ se calcule en passant une seule fois à travers les cellules du réseau. Cette contrainte facilite l'étude des méthodes systématiques de conception d'algorithmes systoliques pour le produit matriciel sur cette classe de réseaux. En effet, dans une telle approche, chaque entrée des matrices à multiplier est représentée par un seul flot de variables. En revanche elle est restrictive et elle augmente la complexité du problème en temps et en nombre de cellules. Cette situation est analogue à ce qui se passe en dimension 2. En effet, tous les algorithmes optimaux connus pour le produit de deux matrices denses sur un réseau systolique bidimensionnel requiérent que certaines données $a_{i,k}$ ou $b_{k,j}$ soient lues plusieurs fois par le réseau [MeT 85, MeT 88].

Les algorithmes présentés ici sont plus performants que ceux de Varman, Ramakrishnan et Fussell [VaR 84, VaF 83]

2.2. Notations et Définitions de base

Dans un premier temps, nous étudions les réseaux où la cellule de base est la plus élémentaire possible comme le schématise la figure 2.2 ci-dessous.

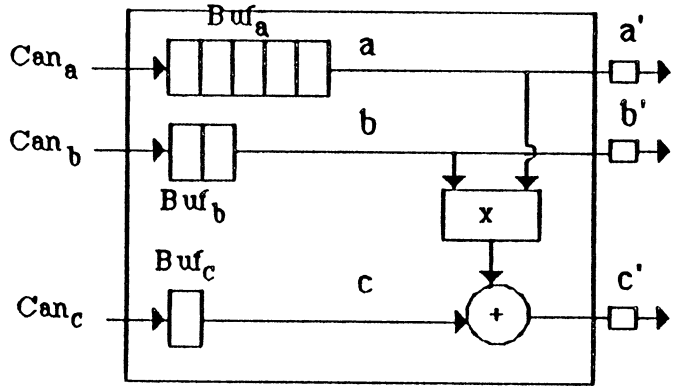


Figure 2.2 : cellule de base

Can_a, Can_b et Can_c désignent les canaux empruntés respectivement par les éléments des matrices A, B et C. Buf_a, Buf_b et Buf_c sont des buffers de tailles respectives x , 2 et 1. Ces buffers servent à retarder les élèments des matrices A , B et C, afin de les faire arriver, au bon moment, dans les unités de calcul. Une unité d' un buffer doit donc être capable de stocker une donnée du même type que les éléments des matrices à multiplier. Avant de passer au cas général, donnons un exemple avec n=2 et x=4.

Exemple 2.1. n=2, x=4

Un algorithme est défini par les temps d'entrée des données dans le réseau. On note

- $T_a[i,j]$ est le temps d'entrée de $a_{i,j}$ dans le réseau,
- $T_b[i,j]$ est le temps d'entrée de $b_{i,j}$ dans le réseau,
- $T_c[i,j]$ est le temps d'entrée de $c_{i,j}$ dans le réseau.

Les matrices T_a, T_b et T_c ci-dessous définissent un algorithme pour le produit de deux matrices de taille 2x2, sur un réseau composé de 4 cellules, et qui s'exécute

en temps T= 17. Dans cet exemple comme pour toute la suite l'unité de temps est égale au délai nécessaire pour effectuer une accumulation c:=c+a.b.

$$T_a = \begin{pmatrix} 6 & 3 \\ 4 & 1 \end{pmatrix} \qquad T_b = \begin{pmatrix} 8 & 10 \\ 7 & 9 \end{pmatrix} \qquad T_c = \begin{pmatrix} 9 & 12 \\ 10 & 13 \end{pmatrix}$$

Vérifions par exemple que cet algorithme effectue l'accumulation $c_{21} := c_{21} + a_{21}.b_{11}$.

- La variable $a_{2,1}$ entre dans le réseau à l'instant $T_a[2,1] = 4$. A l'instant 4+x = 8, elle se trouve à l'entrée du multiplieur de la cellule 1, et à l'instant 12, elle se trouve à l'entrée du multiplieur de la cellule 2.

- L'élément $b_{1,1}$ entre dans le réseau à l'instant $T_b[1,1] = 8$. A l'instant 10 = 8+2, il se trouve à l'entrée du multiplieur de la cellule numéro1 et à l'instant 12 = 10+2, il se trouve à l'entrée du multiplieur de la cellule 2.

- L'élément $c_{2,1}$ entre dans le réseau à l'instant $T_c[2,1] = 10$. A l'instant 11, il se trouve à l'entrée de l'additionneur de la cellule 1 et à l'instant 12 il se trouve à l'entrée de l'additionneur de la cellule 2.

Ceci montre que $c_{2,1}$, $a_{2,1}$ et $b_{1,1}$ se trouvent simultanément à l'entrée des éléments fonctionnels de la cellule 2 à l'instant 12. L'accumulation

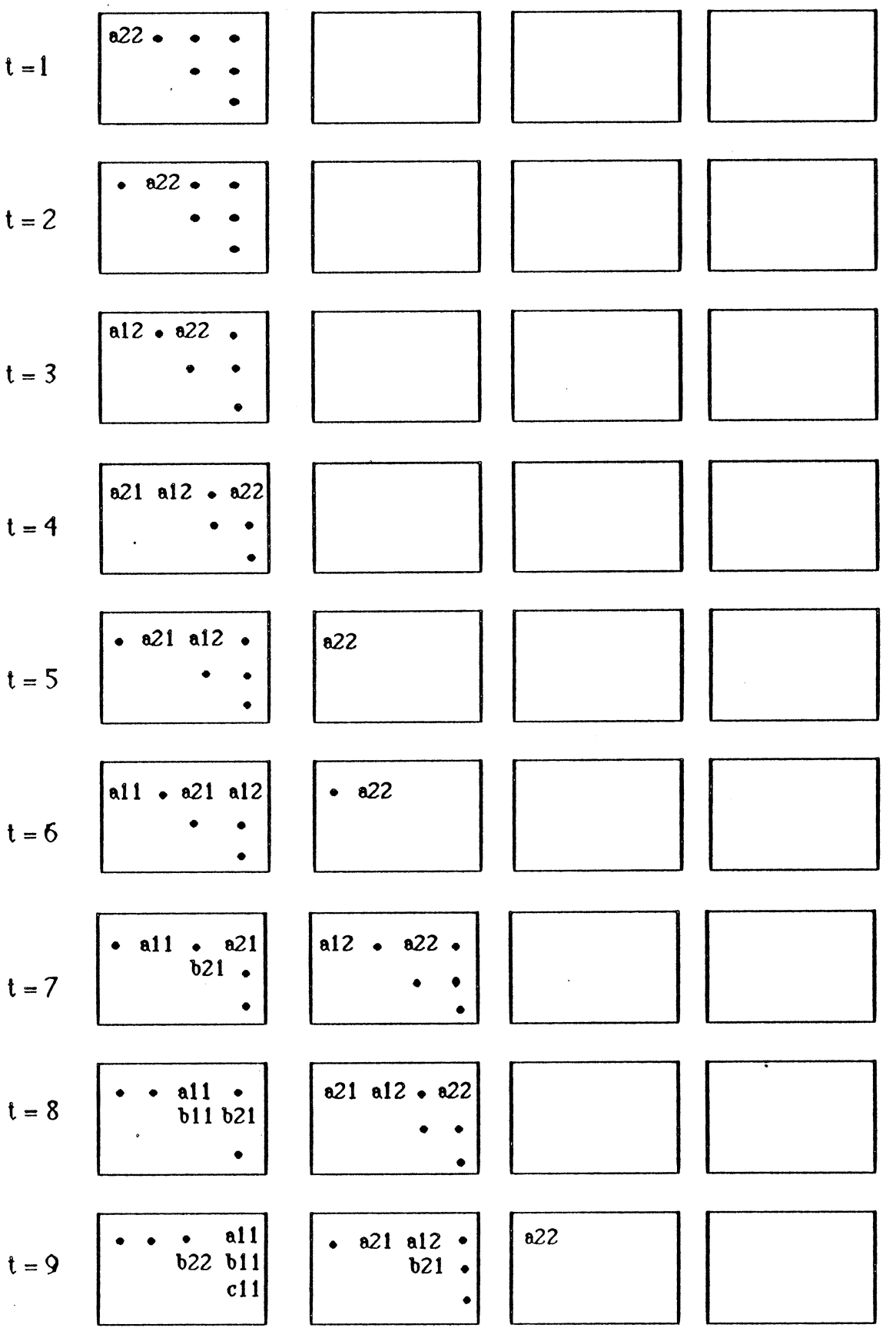
$$c_{21} := c_{21} + a_{21}.b_{11}$$

a lieu dans la cellule 2 à l'instant t = 12.
Plus généralement , on vérifie aisément dans cet exemple que toute variable $c_{i,j}$ effectue l'accumulation

$$c_{i,j} := c_{i,j} + a_{i,k}.b_{k,j}$$

dans la cellule i+j+k-2 à l'instant t = 2i+4j+3k+1.

La figure 2.3 donne une simulation complète de cet algorithme.


t = 1
a22
t = 2
a22
t = 3
a12 a22
t = 4
a21 a12 a22
t = 5
a21 a12
a22
t = 6
a11 a21 a12
a22
t = 7
a11 a21
b21
a12 a22
t = 8
a11
b11 b21
a21 a12 a22
t = 9
a11
b22 b11
c11
a21 a12
b21
a22

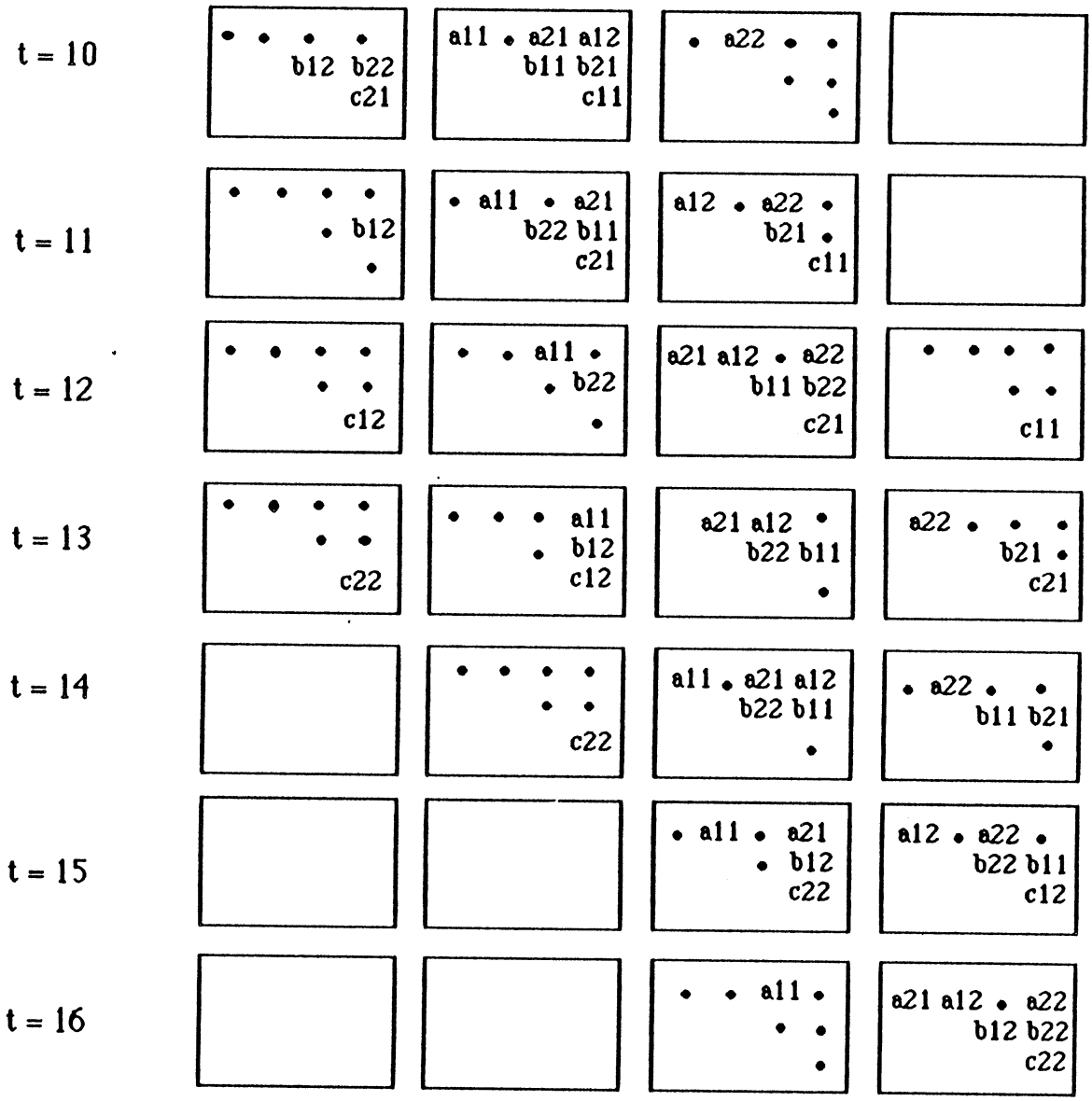


Figure 2.3: simulation de l'algorithme
n = 2 et x = 4.

2.3. Formulation géométrique

Nous allons maintenant proposer une formulation géométrique pour le problème du calcul du produit C = A.B de deux matrices carrées denses de taille nxn, sur un réseau unidirectionnel. Cette formulation étend aux réseaux linéaires, l'approche combinatoire développée par L.Melkemi et M.Tchuente [MeT 85, MeT 88], pour le produit matriciel sur les réseaux bidimensionnels à connexions orthogonales ou héxagonales.

Le diagramme de la figure 2.4 ci-dessous permet de suivre, pour l'exemple 2.1 traité au paragraphe précédent, l'évolution des données en fonction de leur date d'entrée dans le réseau. Ce diagramme comporte:

- Un axe vertical orienté vers le haut qui représente le temps.
- Un axe horizontal représentant les numéros des cellules du réseau.

Suivons dans ce diagramme la trajectoire d'une variable $a_{i,k}$ qui doit traverser 4 registres pour passer d'une cellule à une autre.

- Si elle se trouve à l'entrée du réseau à l'instant to alors on peut la représenter en affectant $a_{i,k}$ au point de coordonnées (0,to).
- Il est alors clair qu'elle sera à l'entrée du multiplieur de la cellule 1 à l'instant to+4, ce qui correspond au point (1,to+4).
- Plus généralement, pour tout j, $1 \leq j \leq 4$, $a_{i,k}$ sera à l'entrée du multiplieur de la cellule j à l'instant to+4j, ce qui correspond au point (j,to+4j).

La trajectoire de $a_{i,k}$ pendant le déroulement de l'algorithme est donc constituée par le segment de pente 4 joignant (0,to) à (4,to+16).

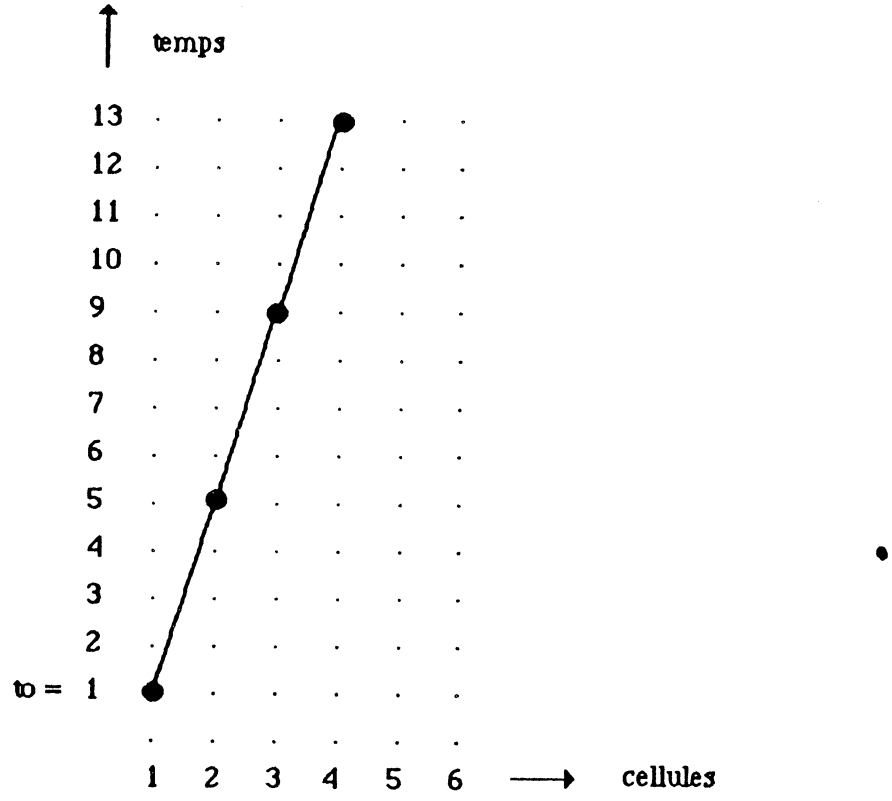


Figure 2.4 : to =1 et x = 4.

Considérons maintenant un algorithme qui efféctue le produit C = A.B de deux matrices denses A, B, carrées de taille n sur un réseau de S cellules, et dans lequel les buffers Buf_a , Buf_b et Buf_c sont de longueurs respectives x, 2 et 1.

- La trajectoire de toute variable $a_{i,k}$ sera représentée par le segment de pente x joignant les points $(0,T_a[i,k])$ et $(S, T_a[i,k]+S.x)$. Dans la suite ce segment sera noté $D_a^{i,k}$.

- La trajectoire de toute variable $b_{k,j}$ sera représentée par le segment de pente 2 joignant les points $(0,T_b[k,j])$ et $(S, T_b[k,j]+2S)$. Dans la suite ce segment sera noté $D_b^{k,j}$.

- La trajectoire de toute variable $c_{i,j}$ sera représentée par le segment de pente 1 joignant les points $(0,T_c[i,j])$ et $(S, T_c[i,j]+S)$. Dans la suite ce segment sera noté $D_c^{i,j}$.

Définition 2.1: Un point P du cadran positif est appelé point de calcul s'il existe (i,j,h,k,p,q) tel que $P = D_a^{i,j} \cap D_b^{h,k} \cap D_c^{p,q}$.

Si $P = D_a^{i,j} \cap D_b^{h,k} \cap D_c^{p,q}$ est un point de calcul de coordonnées (s,t), alors les variables $a_{i,j}$, $b_{h,k}$ et $c_{p,q}$ sont à l'entrée des éléments fonctionnels de la cellule d'indice s à l'instant t. Cette cellule peut donc effectuer l'accumulation

$c_{p,q} := c_{p,q} + a_{i,j}.b_{h,k}$,

à l'instant t.

Dans le modèle que nous étudions dans ce paragraphe, il n'y a pas de signaux de contrôle. En conséquence toute rencontre entre trois variables $c_{p,q}$, $a_{i,j}$ et $b_{h,k}$ donne effectivement lieu à une accumulation

$c_{p,q} := c_{p,q} + a_{i,j}.b_{h,k}$.

Nous appelons diagramme de calcul d'un algorithme, l'ensemble de ses points de calcul. Un algorithme sans signaux de contrôle, défini par les matrices de temps T_a , T_b et T_c, effectue correctement le produit , C=A.B , de deux matrices *denses* de taille nxn, si et seulement si son diagramme de calcul vérifie la condition ci-dessous:

(*) P est un point de calcul ssi il existe (i,j,k) tels que $P = D_a^{i,k} \cap D_b^{k,j} \cap D_c^{i,j}$.

La condition nécessaire assure que tout calcul effectué est de la forme

$c_{i,j} := c_{i,j} + a_{i,k}.b_{k,j}$.

La condition suffisante garantit que toutes les accumulations

$c_{i,j} := c_{i,j} + a_{i,k}.b_{k,j}$, $1 \le i,j,k \le n$

sont effectuées au cours du déroulement de l'algorithme.

Il convient de bien noter que cette caractérisation n'est valable que pour les algorithmes sans signaux de contrôle, et donc où toute rencontre de trois variables donne lieu à une accumulation. Dans le cas plus général où on a des signaux de contrôle, le diagramme de calcul peut contenir des points correspondant à des rencontres indésirables, et les signaux de contrôle ont alors pour rôle d'invalider de tels points. D'autre part, nous verrons au chapitre suivant, des exemples qui correspondent au cas où certaines données $a_{i,k}$ ou $b_{k,j}$ sont lues plusieurs fois.

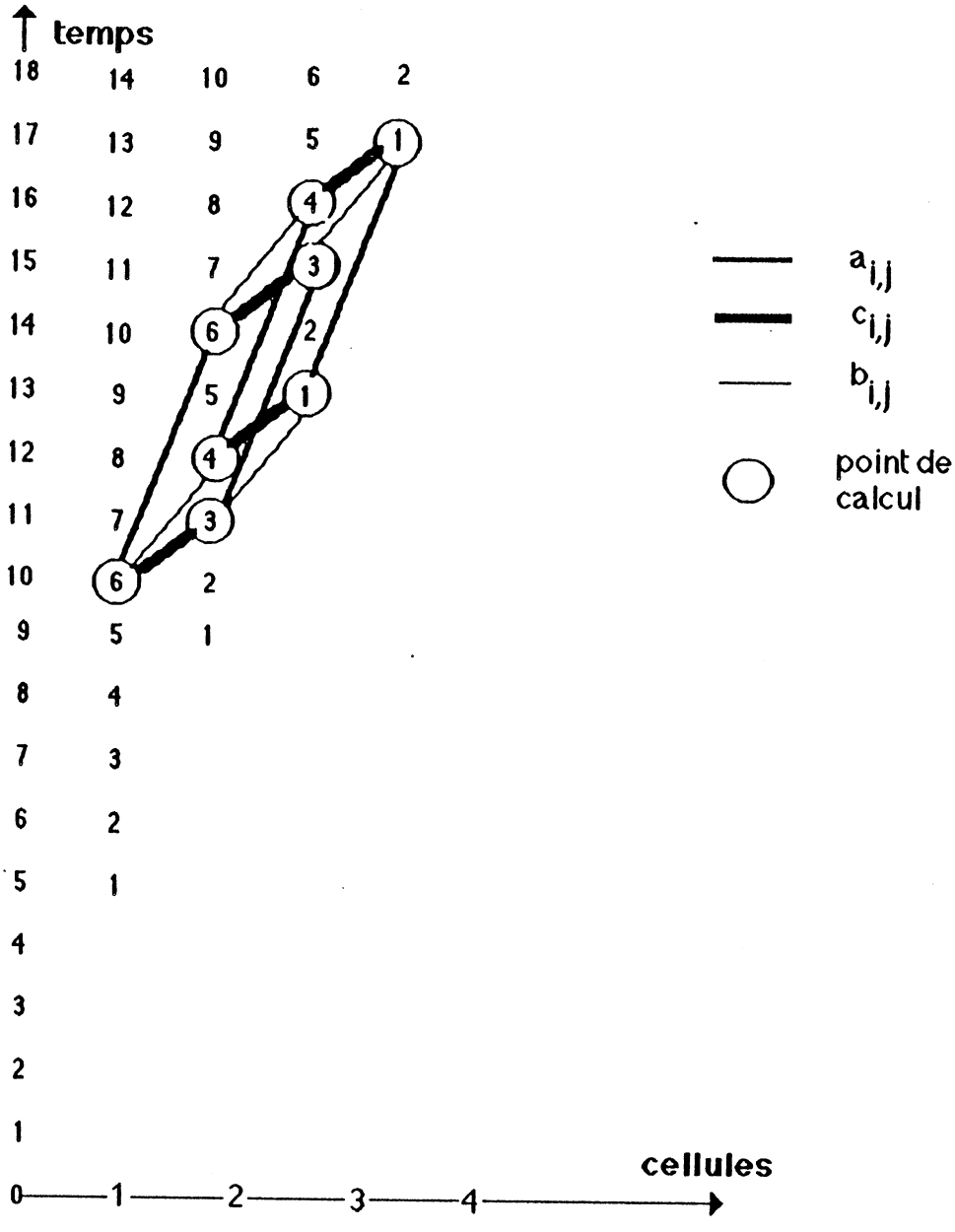


Figure 2.5 : diagramme de calcul: n=2 et x = 4.

Ainsi, le diagramme de calcul permet de visualiser dans le plan les calculs d'un réseau unidirectionnel.

A chaque algorithme, pour le produit de deux matrices et dans lequel Buf_a , Buf_b et Buf_c sont de longueurs respectives x, 2 et 1, est associé un diagramme de calcul. Ce diagramme est constitué de segments de pente x et d'origine $(0,T_a[i,j])$, de segments de pente 2 et d'origine $(0,T_b[i,j])$ et de segments de pente 1 et d'origine $(0,T_c[i,j])$. La figure 2.4 illustre le diagramme de calcul de l'exemple 2.1.

2.3.1. La méthode de projection

Certains algorithmes pour le produit matriciel sur un réseau linéaire unidirectionnel peuvent s'obtenir par une généralisation de la méthode des projections des dépendances développée notamment par Moldovan [Mol 82, Mol 83], Miranker et Winkler [MiW 84], Quinton [Qui 83, Qui 84], et Mongenet [Mon 85]. On procède alors en quatre étapes comme suit:

Etape 1: On associe les calculs aux points d'un domaine de Z^3.

Partant des équations usuelles

$$c_{i,j} = a_{i,1}.b_{1,j} + ... + a_{i,n}.b_{n,j}$$
$$1 \leq i,j,k \leq n$$

On déduit une formulation sous forme d'équations récurrentes uniformes:

$$c(i,j,k) = c(i,j,k-1) + a(i,j,k).b(i,j,k)$$
$$a(i,j,k) = a(i,j-1,k)$$
$$b(i,j,k) = b(i-1,j,k)$$
$$1 \leq i,j,k \leq n.$$

Avec les conditions aux limites:

$$c(i,j,0) = 0 \; ; \quad c(i,j,n) = c_{i,j}$$
$$a(i,0,k) = a_{i,k} \; ;$$
$$b(0,j,k) = b_{k,j}$$

Etape 2: Construction du graphe des dépendances.

Les équations récurrentes ci-dessus font apparaitre un domaine de calcul

$$D_n = \{ (i,j,k) \mid 1 \leq i , j , k \leq n \}.$$

Par ailleurs, l'ensemble des vecteurs de dépendances est

$$U = \{ u_a = (0,1,0), u_b = (1,0,0), u_c = (0,0,1) \}.$$

En effet,

- la valeur de c au point (i,j,k) dépend de la valeur de c au point (i,j,k-1)
- la valeur de a au point (i,j,k) dépend de la valeur de a au point (i,j-1,k)
- la valeur de b au point (i,j,k) dépend de la valeur de b au point (i-1,j,k)

On obtient ainsi le graphe $G_n = (D_n,U)$ illustré dans la figure 2.5 pour n = 2.

Etape 3:

Pour obtenir un réseau bidimensionnel il suffit d'appliquer à G une transformation spatio-temporelle qui respecte les relations de dépendance entre les calculs. Dans le cas de réseaux linéaires unidirectionnels, il faut d'abord construire une représentation de G dans Z^2. La représentation de l'exemple 2.1 présenté précédement pour n=2 est obtenue en appliquant à G une projection π telle que:

$$\pi(0,0,1) = (1,1), \pi(0,1,0) = (1,2) \text{ et } \pi(1,0,0) = (1,4).$$

Contrairement à ce qui se passe en dimension 2, cette projection est soumise à des contraintes assez spéciales liées à la caractérisation des points de calculs étudiée précédemment.

Etape 4:

Une fois l'immersion de G dans Z^2 effectuée à l'étape 3, il suffit d'adopter la stratégie d'allocation correspondant à la projection verticale, pour obtenir un algorithme s'exécutant sur un réseau linéaire unidirectionnel.

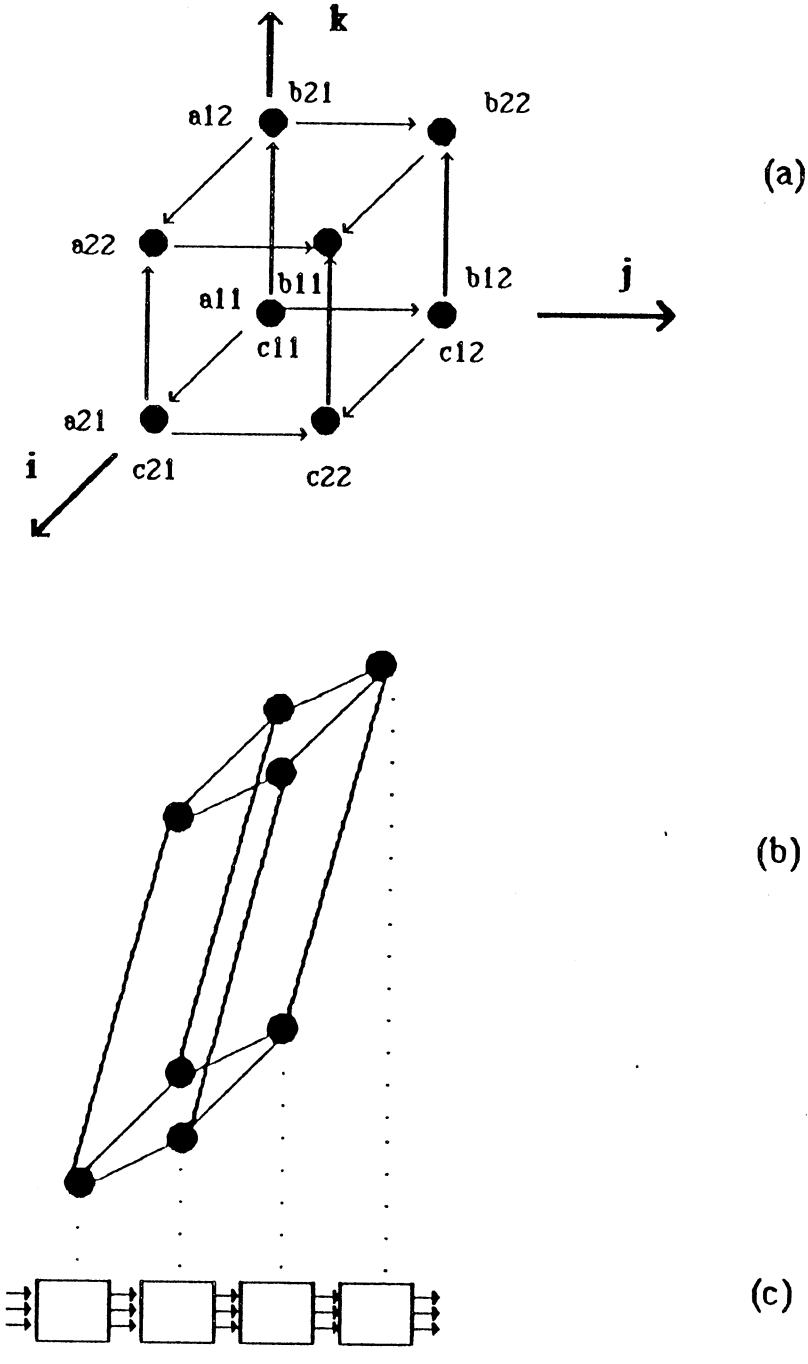


Figure 2.6 (a): graphe de dépendance
(b): immersion du graphe de dépendance dans Z^2
(c): réseau unidirectionnel obtenu.

2.3.2. Approche combinatoire

Nous allons montrer à travers trois exemples comment une approche purement combinatoire permet de construire les diagrammes de calcul correspondant à des algorithmes de produit matriciel sur un réseau linéaire, unidirectionnel et sans signaux de contrôle.

Pour simplifier l'exposé et surtout pour une meilleure clarté des figures, nous allons considérer les réseaux synchrones dans lesquels chaque élément $c_{i,j}$ se propage en cascade à travers toutes les cellules entre deux tops d'horloges. On sait, d'après la méthode de transformation des circuits synchrones, introduite par Leiserson et Saxe [LeS 78], que pour tout algorithme défini sur un tel réseau synchrone, il existe un algorithme défini sur un réseau systolique obtenu en ajoutant un registre supplémentaire à Can_a, Can_b et Can_c, entre deux cellules quelconques. Plus précisement, si un réseau synchrone de taile S comporte

- x-1 registres sur Can_a
- 1 registres sur Can_b
- 0 registre sur Can_c

et s'il résoud le problème en temps T , alors il existe un algorithme équivalent correspondant au même flot de données et qui s'exécute en temps T' = T+S sur un réseau systolique comportant

- x registres sur Can_a
- 2 registres sur Can_b
- 1 registre sur Can_c

Dans le diagramme de calcul, du réseau synchrone, on a les propriétés suivantes:

- toute variable $a_{i,k}$ se déplace le long d'un segment de pente x-1
- toute variable $b_{k,j}$ se déplace le long d'un segment de pente 1
- toute variable $c_{i,j}$ se déplace horizontalement.

On note P(i,j,k) le point de calcul correspondant à l'accumulation

$c_{i,j} := c_{i,j} + a_{i,k}b_{k,j}$

Comme nous considérons des réseaux sans signaux de contrôle tout point de calcul donne lieu à une accumulation. Les points de calculs doivent donc respecter les contraintes géométrique suivantes:

- Si P(i,j,k) et P(i',j',k') appartiennent à une droite horizontale, alors i=i' et j=j'. En effet, ils sont situés sur la trajectoire d'un même élément $c_{i,j}$, ils correspondent donc à des accumulations associées à cet élément.

- Si P(i,j,k) et P(i',j',k') appartiennent à une droite de pente 1, alors k=k' et j=j'. En effet, il sont situés sur la trajectoire d'un même élément $b_{k,j}$, et ils correspondent donc à des accumulations associées à cet élément.

- Si P(i,j,k) et P(i',j',k') appartiennent à une droite de pente x-1 alors i=i' et k = k'. En effet, il sont situés sur la trajectoire d'un même élément $a_{i,k}$, ils correspondent donc à des accumulations associées à cet élément.

Nous appelons pavé (j) l'ensemble des points de calcul correspondant aux accumulations requises par la j^{eme} colonne de C. Il est donc constitué des points P(i,j,k), $1 \leq i, k \leq n$.
Nous construisons successivement les pavés numéros 1, 2 ... n. De plus chaque pavé est construit segment par segment
- soit en construisant successivement les segments de pentes +1
- soit en construisant successivement les segments horizontaux

Exemple 2.2: n=3 et x=5.

Les figures 2.7a, 2.7b, 2.7c, 2.7d et 2.7e montrent les étapes successives de la construction du diagramme de calcul dans ce cas.

Etape 1: consruction du pavé (1)

Nous allons construire ce pavé en considérant successivement les trois segments horizontaux qui le composent.

(a): la construction du premier segment se fait sans contrainte. Comme nous voulons minimiser le nombre de cellules, il faut évidement l'affecter à trois points consecutifs (cf. figure 2.7a)

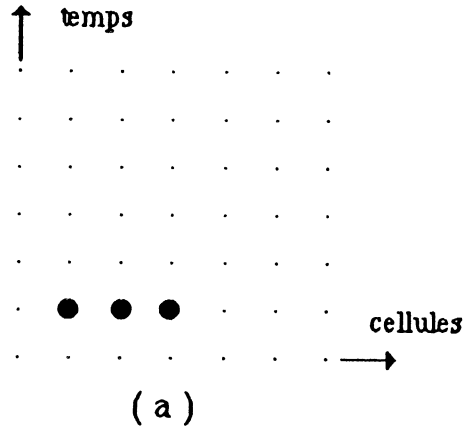


Figure 2.7a: construction du premier segment

(b): pour le deuxième segment, il faut tenir compte du fait qu'il correspond à une même colonne que le premier segment. Il s'obtient donc à partir du premier segment par une translation de vecteur parallèle à (1,1). Comme nous voulons minimiser le temps de calcul, nous prenons (1,1) comme vecteur de translation (cf. figure 2.7b) .

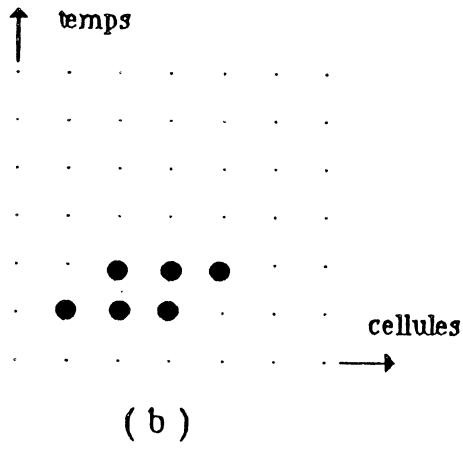


Figure 2.7b: construction du deuxième segment

(c): la construction du troisième segment s'effectue de manière similaire à l'étape (b) (cf. figure 2.7c).

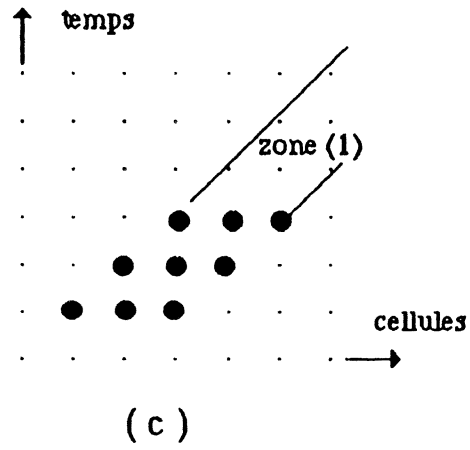


Figure 2.7c: construction du troisième segment

La construction de ce pavé fait apparaitre une bande notée zone (1) dans la figure ci-dessus et qui réservée aux éléments de la première colonne de B. Comme les autres pavés n'utilisent aucun élément de cette première colonne, cette zone leur est interdite.

Etape 2: consruction du pavé (2)

Le pavé (2) correspond aux calculs de la deuxième colonne de C. Il convient de remarquer que la construction du pavé (1) détermine les dates d'entrées de tous les coefficients de la matrice A. Or nous avons supposé au départ que toute donnée est lue une seule fois par le réseau. Ce sont donc les mêmes coefficients $a_{i,k}$ qui traversent les pavé (1) et pavé (2). Comme les $a_{i,k}$ se déplacent le long de segments de pentes x, le pavé (2) s'obtient à partir du pavé (1) par translation de vecteur h(1,x-1), h≥1. Nous constatons que la translation (1,x-1) convient (cf. figure 2.7d)

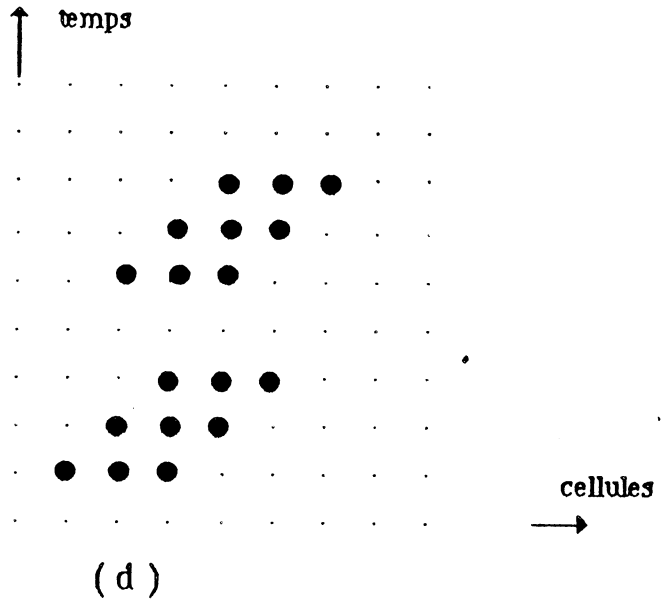


Figure 2.7d: construction du pavé (2)

Etape 3: consruction du pavé (3)

Le raisonnement fait à l'étape 2, montre que le pavé (3) s'obtient par translation de vecteur (1,x-1) à partir du pavé (2) (cf. figure 2.7e).

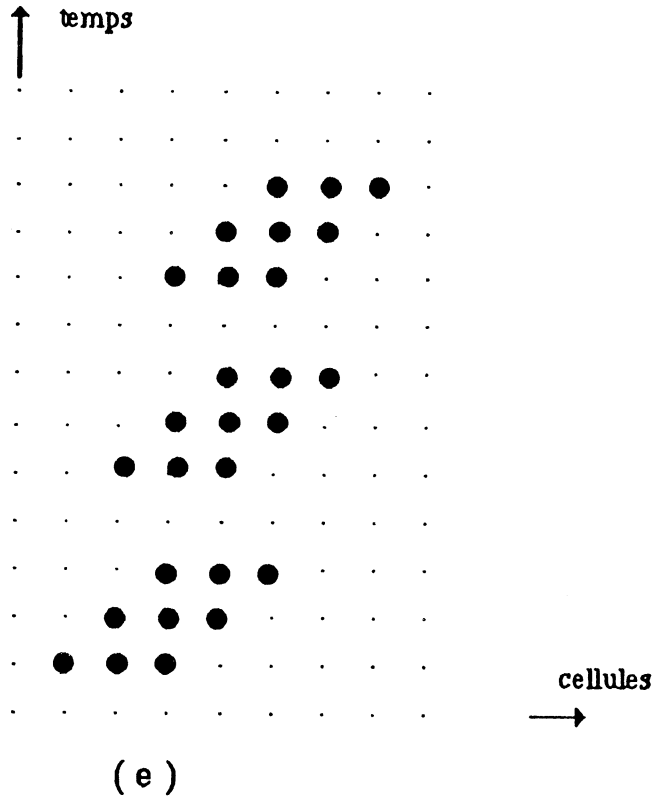
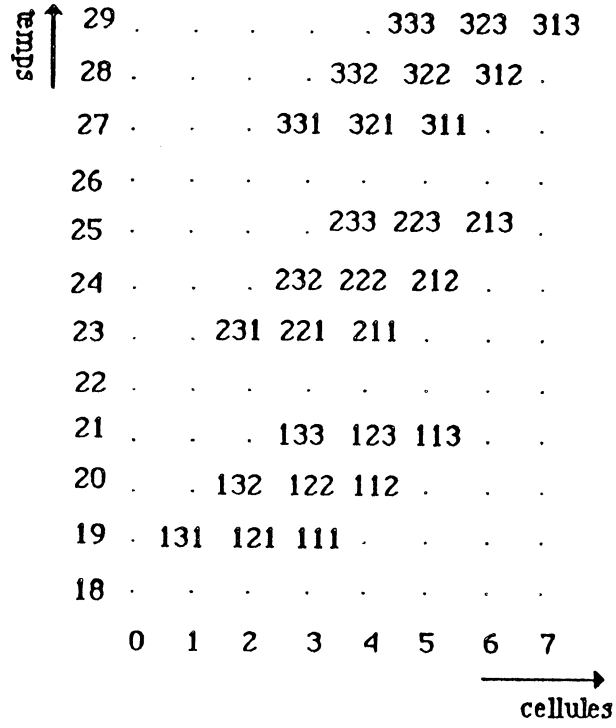


Figure 2.7e: construction du pavé (3)

Le diagramme de calcul ainsi obtenu est décrit dans la figure 2.8



ikj représente l'accumulation $c_{i,j} := c_{i,j} + a_{i,k} \cdot b_{k,j}$.

Figure 2.8: diagramme de calcul pour n=3, x-1=4.

Exemple 2.3: n=3 et x=4.

Comme pour l'exemple 2.2, nous allons construire le diagramme de calcul du réseau synchrone ayant (x-1) registres sur Can_a , 1 registre sur Can_b et aucun registre sur Can_c.

Etape 1: Construction du pavé (1):

Cette étape s'effectue comme dans l'exemple précédent. A la fin de cette étape on a, comme dans l'exemple 1, une zone interdite pour les points du pavé (2).

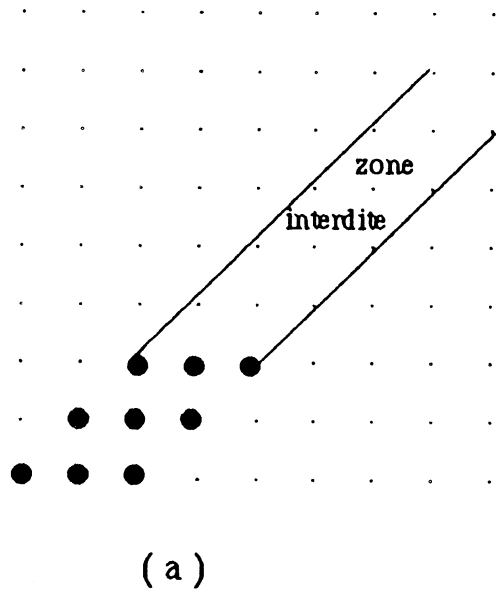
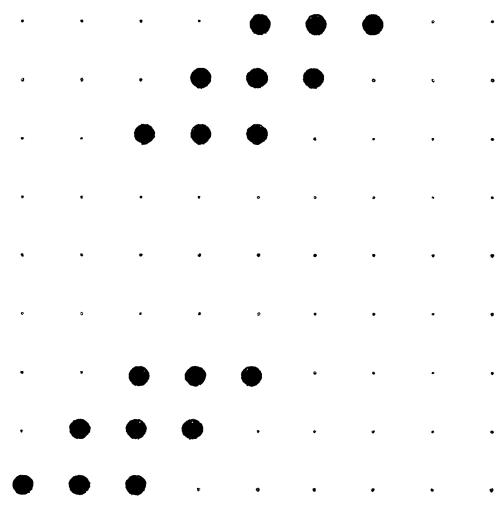


Figure 2.9a: construction du pavé (1)

Etape 2: Construction du pavé (2):

Cette fois, on remarque que la translation (1,x-1) n'est pas autorisée. En effet, le pavé obtenu par translation de vecteur (1,x-1) à partir du pavé (1) contient un point de la zone interdite. En revanche, la translation de vecteur 2(1,x-1) convient. Nous construisons donc le pavé (2) par une translation de vecteur 2(1,x-1) à partir du pavé (1).



(b)

Figure 2.9b: construction du pavé (2)

Plus généralement, tout pavé de numéro j , j ≥ 2 s'obtient par translation de vecteur 2(1,x-1) à partir du pavé (j-1).

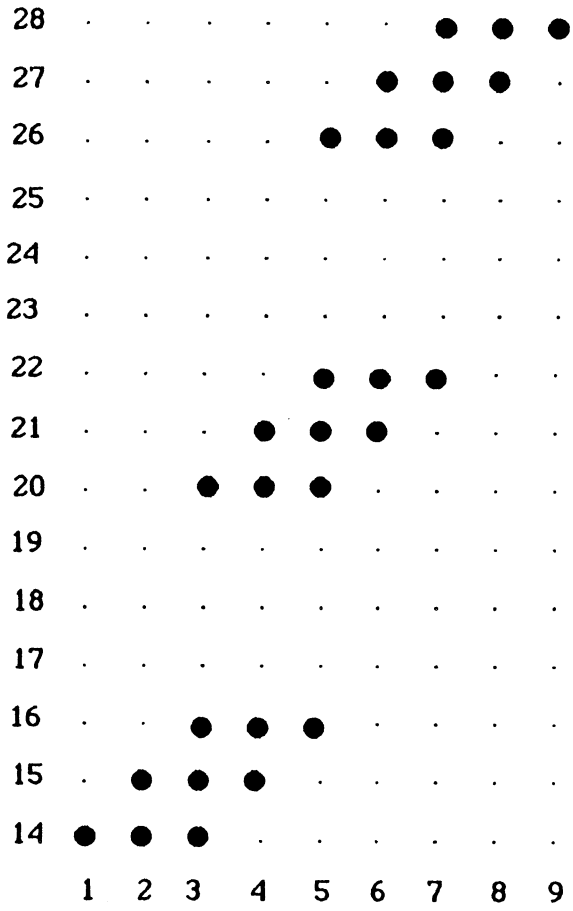


Figure 2.9c : diagramme de calcul, cas n=3 et x-1 = 3.

Exemple 2.4: $n = x = 4$.

Dans cet exemple nous allons exposer deux solutions. Dans la première solution tout pavé se construit à partir de ses segments horizontaux, tandis que dans la deuxième solution, tout pavé se construit à partir de ses segments de pente +1.

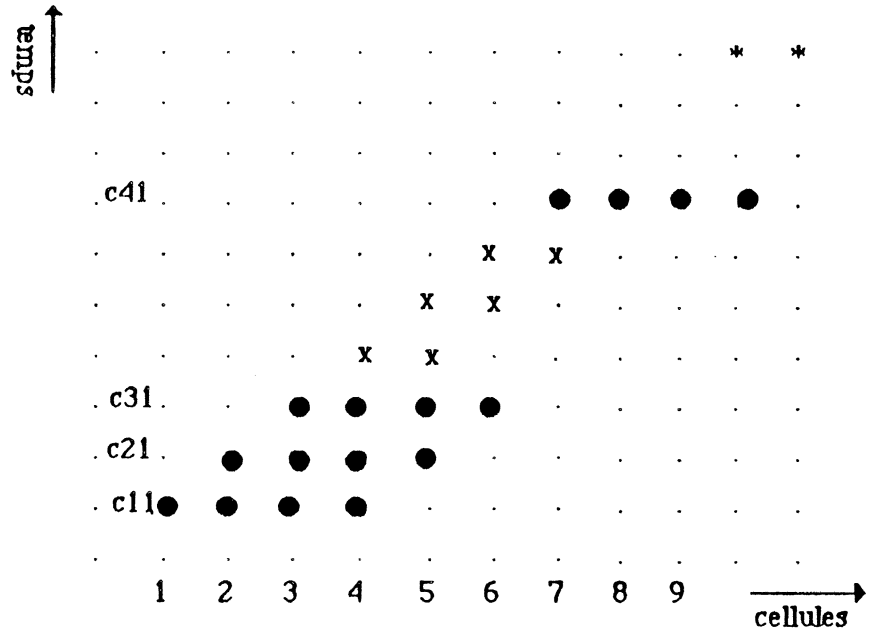
Solution 1:

Etape 1: Construction du pavé (1):

Les trois premiers segments du pavé (1) se placent sans aucune contrainte.Une fois placés ces trois segments, on voit apparaître six points interdits, dans la bande où nous voulons placer les prochains segments du pavé (1). Ces points sont représentés par des croix dans la figure 2.10a, et correspondent à des accumulations

$c := c + a_{i,k}\, b_{h,l}$, où $k \neq h$.

Le quatrième segment est donc placé avec un écart de trois lignes par rapport au précédent.



x : ponit interdit pour le 4^{eme} segment,
$*$: point interdit pour le pavé (2).

Figure 2.10a: construction du pavé (1).

Le 4^{eme} segment fait apparaitre deux points interdits pour le pavé(2). Ces deux points sont représentés par des étoiles sur la figure 2.10a.

Etape 2: Construction du pavé (2):

Le pavé (2) doit vérifié les contraintes suivantes:

(1) Il s'obtient à partir du pavé (1) par une translation proportionnelle à (1,x-1). Cette condition vient du fait que se sont les mêmes $a_{i,k}$ qui traversent le pavé (1) et le pavé (2).

(2) Un point du pavé (2) ne doit pas être sur la même horizontale qu'un point du pavé (1) car sinon le $c_{i,2}$ correspondant ferait une accumulation de la forme

$c_{i,2} := c_{i,2} + a_{i,k}\, b_{k,1}$

(3) Un point du pavé (2) ne doit pas être sur la même horizontale qu'un point interdit par le pavé (1) car sinon le $c_{i,2}$ correspondant ferait une accumulation de la forme

$c_{i,2} := c_{i,2} + a_{i,k}\, b_{h,1}$, avec $h \neq k$.

- La translation (1,x-1) ne convient pas à cause de la condition (3).

- La translation 2(1,x-1) ne convient pas à cause de la condition (2).

- La translation 3(1,x-1) ne convient pas à cause de la condition (3).

En revanche la translation 4(1,x-1) convient.

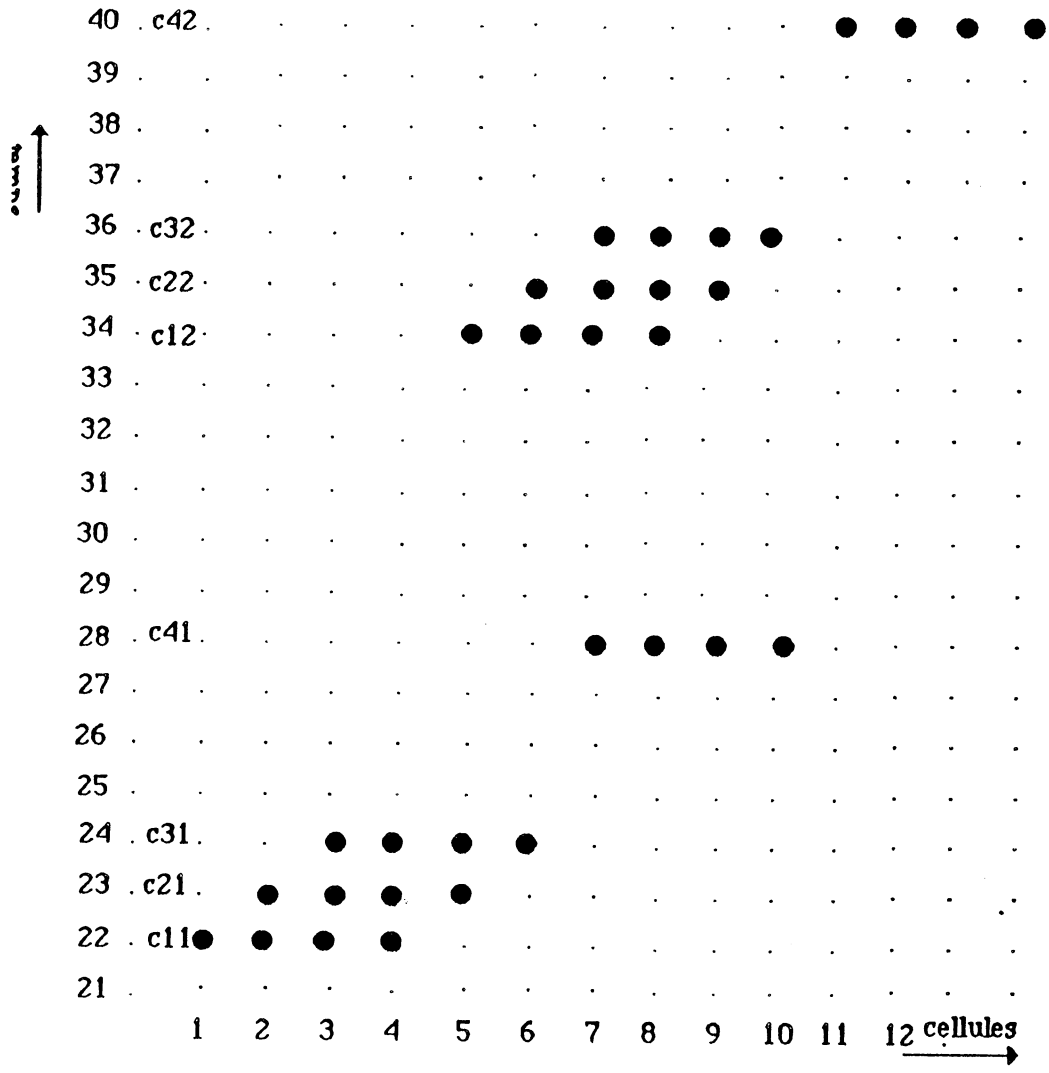


Figure 2.10b: construction du pavé (2)

Plus généralement, tout pavé de numéro j , $j \geq 2$ s'obtient par translation de vecteur 4(1,x-1) à partir du pavé (j-1).

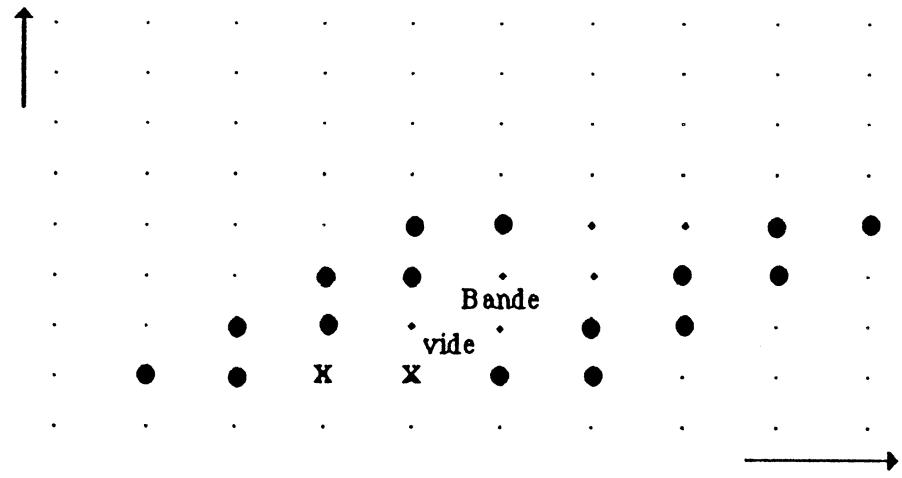
L'algorithme ainsi obtenu s'exécute en 3x22 = 66 cycles sur un réseau synchrone composé de 22 cellules où on a 3 registres sur Can_a, 1 registre sur Can_a et aucun registre sur Can_c. Ceci correspond à un algorithme systolique qui s'exécute en temps T = 4x22 = 88 cycles sur un réseau de taille 22 où on a 4 registres sur Can_a, 2 registres sur Can_a et 1 registre sur Can_c.
Examinons maintenant la deuxième solution.

Solution 2:

Dans cette solution nous construisons chaque pavé par juxtaposition de ses segments de pente 1.

Etape 1: Construction du pavé (1)

La construction des deux premiers segments se fait sans contrainte. Une fois ces deux segments placés, on voit apparaître deux points interdits dans la zone où nous voulons placer les prochains segments de ce pavé. Ces points sont représentés par des étoiles dans la figure 2.11a. Nous sommes donc ammenés à sauter deux lignes avant de placer les deux prochains segments.



x : point interdit par les deux segments de pente +1.

Figure 2.11a: construction du pavé (1)

Etape 2: Construction du pavé (2)

La translation (1,x-1) ne convient pas car le pavé ainsi obtenu aurait un segment horizontal sur la même droite qu'un segment du pavé (1). La translation 2(1,x-1) ne convient pas car le pavé ainsi obtenu aurait des points dans la zone interdite. En revanche la translation 3(1,x-1) convient (cf. figure 2.11b)

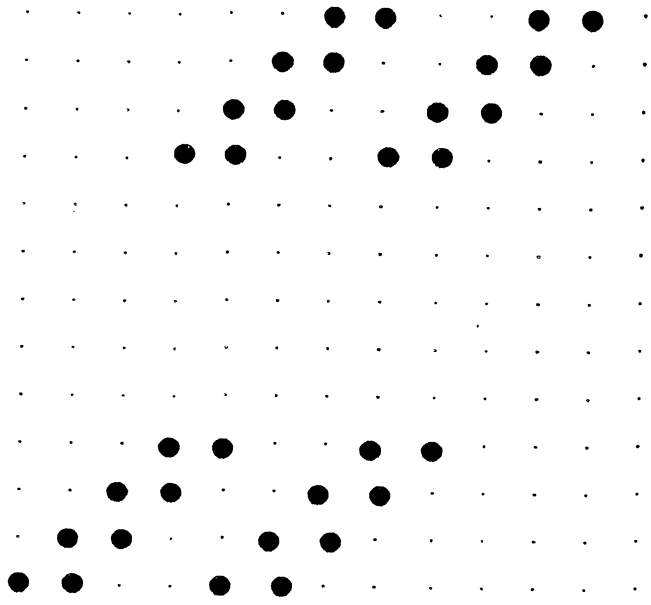


Figure 2.11b: construction du pavé (2)

Plus généralement, tout pavé de numéro j , $j \geq 2$, s'obtient par translation de vecteur 3(1,x-1) à partir du pavé (j-1).

Cet algorithme s'exécute en 3x18 = 54 cycles sur un réseau synchrone composé de 18 cellules où on a 3 registres sur Can_a, 1 registre sur Can_a et aucun registre sur Can_c. Ceci correspond à un algorithme systolique qui s'exécute en temps T = 4x18 = 72 cycles sur un réseau de taille 18 où on a 4 registres sur Can_a, 2 registres sur Can_a et 1 registre sur Can_c. Remarquer que cette solution est meilleure que la première.

2.4. Produit de matrices denses: formules analytiques

Dans ce paragraphe, nous allons présenter de manière analytique une famille d'algorithmes pour le produit de matrices denses sur un réseau unidirectionnel sans signaux de contrôle. Ces algorithmes peuvent être obtenus de manière combinatoire, mais l'approche analytique a l'avantage de donner les formules explicites des dates d'entrée des données dans le réseau.

Dans toute la suite nous supposons que $n = \alpha (x-2)$. Cette hypothèse est faite dans le seul but de facilité l'exposé. Tout les résultats présentés ici peuvent s'étendre au cas où n n'est pas multiple de de x-2.

L'algorithme que nous allons présenter maintenant, et que nous avons baptisé ALG1, est défini par les matrices suivantes:

- $T_a[p(x-2)+r,k] = T_o - (k-1)(x-1) - (x-2)(p(n+ \alpha -1) + r)$
- $T_b[k,j] = T_o - (k-1) + (j-1)\, \alpha^2 (x-2)$
- $T_c[p(x-2)+r,j] = T_o + (j-1)\, \alpha^2 (x-1) + p(n+ \alpha -1) + r$

où $0 \leq p \leq \alpha -1$, $1 \leq r \leq x-2$, $1 \leq k \leq n$, $1 \leq j \leq n$
et $T_o = n^2(1+1/\alpha^2) + n(\alpha -1)$.

La valeur de T_o a été déterminée de sorte que

$T_a[p(x-2)+r,k] \geq 0$
$T_b[k,j] \geq 0$
$T_c[p(x-2)+r,j] \geq 0$

et que la valeur 0 soit atteinte pour la première variable lue par le réseau.

Nous allons établir trois lemmes qui permettent de prouver la validité de l'algorithme.

Lemme 2.1. Toutes les données correspondant à une même matrice entrent dans le réseau à des instants distincts.

Démonstration:
(i) Supposons que l'on ait

$T_a[p(x-2)+r,k] = T_a[p'(x-2)+r',k']$.

Ceci entraîne que

$$(k-k')(x-1) + (p-p')(x-2)(n+\alpha-1) + (r-r')(x-2) = 0.$$

Après division par $x-2$, on obtient

$$k-k' + (k-k')/(x-2) + r-r' + (p-p')(n\alpha-1) = 0 \qquad (*)$$

ce qui entraîne que $x-2$ divise $k-k'$. Comme

$$n = \alpha(x-2) \text{ et } |k-k'| < n$$

on déduit que

$$|k-k'| \leq (\alpha-1)(x-2).$$

D'autre part

$$|r-r'| < x-2,$$

ce qui combiné avec (*) entraîne que

$$\begin{aligned} |p-p'|(n+\alpha-1) &\leq |k-k'| + |k-k'|/(x-2) + |r-r'| \\ &< (\alpha-1)(x-2) + (\alpha-1) + x-2 \\ &< \alpha(x-2) + (\alpha-1) \\ &< n+\alpha-1. \end{aligned}$$

Cette dernière inégalité n'est possible que si $p = p'$.
On déduit alors de l'équation (*) que

$$\begin{aligned} |k-k'|(1+1/(x-2)) &= |r-r'| \\ &< x-2. \end{aligned}$$

Ce qui entraîne que

$$|k-k'| < x-2.$$

Comme $|k-k'|$ est multiple de $x-2$, on a

$$k = k'$$

et par conséquent

$$r = r'.$$

(ii) Examinons maintenant les dates d'entrée des éléments de la matrice B.

$$T_b[k,j] = T_b[k',j'] \text{ entraîne que}$$

$$(k-k') - (j-j')\alpha^2(x-2) = 0.$$

Comme $n = \alpha(x-2)$ et $\alpha \geq 1$, on déduit que

$$\begin{aligned} |k-k'| &= n.\alpha.|j-j'| \\ &> n \end{aligned}$$

avec $1 \leq k,k' \leq n$.

D'où

$$k = k' \text{ et } j = j'.$$

(iii) Examinons enfin les dates d'entrée de la matrice C.

$$T_c[p(x-2)+r,j] = T_c[p'(x-2)+r',j']$$

entraîne que

$$(j-j')\alpha^2(x-1) + (p-p')(n+\alpha-1) + r-r' = \qquad (*)$$

Comme $|p-p'| \leq \alpha-1$ et $|r-r'| < x-2$, on a

$$|(p-p')(n+\alpha-1) + r-r'| < (\alpha-1)(n\alpha-1) + x-2. \qquad (**)$$

Par ailleurs,

$$n = \alpha(x-2),\ |j-j'|.\alpha^2(x-1) = |j-j'|(\alpha n + \alpha^2) \text{ et}$$
$$(\alpha-1)(n+\alpha-1) + x - 2 = (\alpha - 1 + 1/\alpha).n + \alpha^2 - 2\alpha + 1.$$

On déduit alors de (**) que

$$|j-j'|(\alpha n + \alpha^2) < (\alpha - 1 + 1/\alpha)n + \alpha^2 - 2\alpha + 1$$

Comme $a \geq 1$, on déduit que

$$j = j'.$$

La formule (*) entraîne alors

$$|p-p'|(n+\alpha-1) = |r-r'| < x-2.$$

D'où $p = p'$ et $r = r'$ ♦

Lemme 2.2. $a_{p(x-2)+r,k}$, $b_{k,j}$ et $c_{p(x-2)+r,j}$ se rencontrent dans la cellule de numéro $s = p(n\alpha - 1) + (j-1)\alpha^2 + r + k - 1$.

Démonstration:

Un simple calcul montre que

$$T_a[p(x-2)+r,k] + x.s = T_b[k,j] + 2s = T_c[p(x-2)+r,j] + s \quad \blacklozenge$$

Lemme 2.3. Si $a_{p'(x-2)+r',k'}$, $b_{k'',j''}$ et $c_{p(x-2)+r,j}$ se rencontrent dans une cellule s, alors $p' = p$, $r' = r$, $k' = k''$ et $j'' = j$.

Démonstration:

Cette rencontre se traduit par les équations suivantes:

$$T_a[p'(x-2)+r',k'] + x.s = T_b[k'',j''] + 2s \quad (1)$$
$$T_b[k'',j''] + 2s = T_c[p(x-2)+r,j] + s \quad (2)$$

où

$$T_a[p'(x-2)+r',k'] = T_o - (k'-1)(x-1) - (x-2)(p'(n+\alpha-1) + r')$$
$$T_b[k'',j''] = T_o - (k''-1) + (j''-1)\alpha^2(x-2)$$
$$T_c[p(x-2)+r,j] = T_o + (j-1\ \alpha^2(x-1) + p(n+\alpha-1) + r.$$

(1) entraîne que

$$s = k' - 1 + p'(n+\alpha-1) + r' + (j''-1)\alpha^2 + (k'-k'')/(x-2) \quad (3)$$

(2) entraîne que

$$s = (j-1)\alpha^2(x-1) + p(n+\alpha-1) + r + k'' - 1 - (x-2)(j''-1)\alpha^2 \quad (4)$$

De (3) et (4) on déduit que

$$(j-j'')\,\alpha^2(x-1) + (p-p')\,(n+\alpha-1) + r-r' + k''-k' + (k''-k')/(x-2) = 0 \qquad (5)$$

Or

$$|\,k''-k'\,| < n$$
$$< \alpha(x-2)$$

et $|\,k2-k1\,|$ divise $x-2$,
donc

$$|\,k''-k'\,| \leq (\alpha-1)(x-2) \text{ et } |\,r-r'\,| < x-2.$$

On déduit alors de (5) que

$$|\,j-j''\,|(\alpha n+\alpha^2) < (\alpha-1)(n+\alpha-1) + x-2 + (\alpha-1)(x-2) + \alpha - 1$$
$$< n\alpha + \alpha(\alpha-1).$$

Ce qui entraîne que $j = j''$.

L'équation (5) devient alors

$$(p-p')(n+\alpha-1) + r-r' + k''-k' + (k''-k')/(x-2) = 0 \qquad (6)$$

On en déduit que

$$|\,p-p'\,|(n+\alpha-1) \leq |\,r-r'\,| + |\,k''-k1\,| + |\,k''-k'\,|/(x-2)$$
$$< x-2 + (\alpha-1)(x-2) + \alpha - 1$$
$$< n + \alpha - 1.$$

D'où $p = p'$.
L'équation (6) devient alors

$$r - r' + k'' - k' + (k''-k')/(x-2) = 0,$$

ce qui entraîne que $r = r'$ et $k'' = k'$ ♦

Le lemme 2.2 montre que toutes les accumulations sont effectuées, et le lemme 2.3 montre que seuls les calculs correspondant au produit $C = A.B$ sont effecutés. Ceci prouve bien la validité de l'algorithme.

Etudions maintenant les performances en temps et en surface de cet algorithme. D'après le lemme 2.2, les éléments $a_{p(x-2)+r,k}$, $b_{k,j}$ et $c_{p(x-2)+r,j}$ se rencontrent dans la cellule de numéro

$$s = p(n+\alpha-1) + (j-1)\,\alpha^2 + r + k - 1.$$

s atteint son minimum pour

$$p = 0\,,\ j = r = k = 1$$

et vaut alors 1.
La valeur maximale de s est atteinte pour

$$p = \alpha - 1\,,\ r = x-2\,,\ k = j = n,$$

ce qui correspond à

$$S_1 = (\alpha^2 + \alpha + 1/\alpha)n - 2\alpha.$$

Le dernier coefficient de C calculé par cet algorithme est $c_{n,n}$, et la première donnée a être lue est $a_{1,1}$. Comme le temps d'exécution de l'algorithme est égal au délai entre la première entrée et la dernière sortie, on déduit qu'il est égal à

$$T_c[n,n] + S - T_a[1,1] = (\alpha+1+1/\alpha^2)n^2 + 2(\alpha^2+\alpha+2/\alpha-2)n - 4\alpha.$$

ALG1 s'exécute donc sur un réseau unidirectionnel de taille

$$S_1 = (\alpha^2+\alpha+1/\alpha)n - 2\alpha$$
$$= (\alpha^2+\alpha+1/\alpha)n + O(1)$$

et en temps

$$T_1 = (\alpha+1+1/\alpha^2)n^2 + O(n)$$

Rappelons que Fussell, Ramakrishnan et Varman [RaV 84], [FuV 85], n'ont obtenu aucun algorithme modulaire, unidirectionnel et totalement pipeliné.

2.5. Résultat de complexité pour x ≥ n+2.

Dnas le cas où x = n+2, l'algorithme que nous avons présenté s'exécute sur un réseau de taille 3n-2 et en temps $T = 3n^2+4n-4$. Nous allons montrer qu'il est optimal. Plus précisément on a le résultat suivant:

Proposition 2.1. Soit un algorithme pour le produit C = A.B, qui s'exécute sur un réseau linéaire unidirectionnel vérifiant les conditions (1) et (2) ci-dessous:

(1) Chaque cellule du réseau a un nombre constant d'entrées / sorties et elle est dotée de buffers de longueurs x ≥ n+2, 2 et 1, sur les canaux respectifs Can_a , Can_b et Can_c.

(2) Une donnée est lue une seule fois.

Cet algorithme a un temps d'exécution supérieur ou égal à (3n-2)x, et la taille du réseau sur lequel il s'exécute est supérieur ou égal à 3n-2.

Démonstation:
Considérons trois éléments de la matrice C, correspondant aux indices (i,j), (i',j) et (i,j'). Notons

- s(k) la cellule où $c_{i,j}$ rencontre $a_{i,k}$ et $b_{k,j}$
- s'(k) la cellule où $c_{i',j}$ rencontre $a_{i',k}$ et $b_{k,j}$
- s"(k) la cellule où $c_{i,j'}$ rencontre $a_{i,k}$ et $b_{k,j'}$.

On a

$$T_c[i,j] + s(k) = T_b[k,j] + 2s(k)$$

et

$$T_c[i',j] + s'(k) = T_b[k,j] + 2s'(k).$$

D'où

$$T_c[i',j] - T_c[i,j] = s'(k) - s(k) \qquad (1)$$

Dans la suite cette égalité sera appelée propriété 1. Elle signifie que deux éléments d'une même colonne de C et qui sont distants de d, rencontreront tout élément $b_{k,j}$

de B dans des cellules dont les numéros differts de d. C'est une propriété évidente car $c_{i,j}$ et $b_{k,j}$ traversent respectivement un et deux registres pour passer d'une cellule à l'autre comme dans le réseau de convolution unidirectionnel.
D'une manière analogue on a,

$$T_c[i,j] + s(k) = T_a[i,k] + x.s(k)$$

et

$$T_c[i,j'] + s''(k) = T_a[i,k] + x.s''(k).$$

D'où

$$T_c[i,j'] - T_c[i,j] = (x-1)(s''(k)-s(k)) \qquad (2)$$

Dans la suite cette égalité sera appelée propriété 2. Elle est plus subtile à interprêter. Elle signifie que deux éléments d'une même ligne de C qui rencontrent donc un même élément $a_{i,k}$ de A dans des cellules distantes de d, sont lues avec un écart dans le temps de (x-1)d. Ceci s'appliquera en remarquant que si deux éléments $c_{i,j}$ et $c_{i,j'}$ sont séparés par q éléments de la ligne i, alors $c_{i,j}$ et $c_{i,j'}$ rencontreront tout élément $a_{i,k}$ dans des cellules distantes d'au moins q+1, et on déduira alors que

$$| T_c[i,j] - T_c[i,j'] | \geq (x-1)(q+1).$$

Notons $c_{i1,j1}$ le premier élément de la matrice C qui entre dans le réseau. Dans la suite, on suppose pour simplifier les notations que i1 = 1 et j1 = 1.

Comme $c_{1,1}$ entre dans le réseau une seule fois, il effectue ses n accumulations dans n cellules différentes. Notons s le numéro de la cellule où $c_{1,1}$ effectue sa dernière accumulation qu'on suppose être

$$c_{1,1} := c_{1,1} + a_{1,n}.b_{n,1}.$$

Il est évident que $s \geq n$.
Considérons la première colonne de la matrice C et supposons que $c_{r,1}$ soit le dernier élément de cette colonne qui entre dans le réseau (cf. figure 2.13).

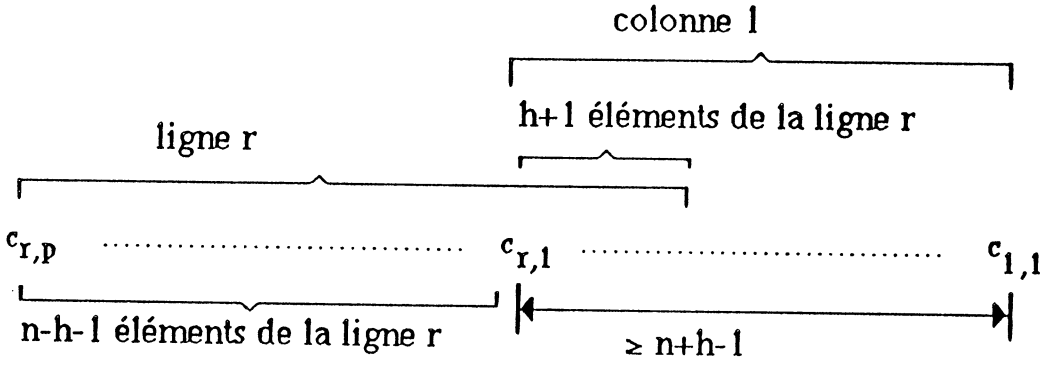


Figure 2.13

Soit h le nombre d'éléments de la ligne r qui entrent dans le réseau avant $c_{r,1}$. On voit qu'entre $c_{1,1}$ et $c_{r,1}$ on a au moins

- n-2 éléments de la première colonne
- h éléments de la ligne r.

Comme la ligne r et la colonne 1 ont au plus un élément en commun , on déduit que

$$T_c[r,1] - T_c[1,1] \geq n + h - 1.$$

D'où, d'après la propriété 1,

$$s' - s \geq n + h - 1$$

où s' est la cellule dans laquelle $c_{r,1}$ rencontre $a_{r,n}$ et $b_{n,1}$.

Considérons maintenant la ligne r de C. Notons $c_{r,p}$ le dernier élément de cette ligne qui entre dans le réseau. Comme h éléments de la ligne r sont lus avant $c_{r,1}$, on déduit qu'il y a n-h-2 éléments de la ligne r entre $c_{r,1}$ et $c_{r,p}$. Ceci entraîne d'après la propriété 2 que

$$T_c[r,p] - T_c[r,1] \geq (n-h-1)(x-1).$$

Notons s" la cellule où $c_{r,p}$ rencontre $a_{r,n}$ et $b_{n,p}$; d'après la propriété 1 on a:

$$s'' - s' \geq n-h-1.$$

La taille S du réseau sur lequel s'exécute l'algorithme doit être forcément supérieure à s". Or

$$\begin{aligned} s'' &\geq (s''-s') + (s'-s) + s \\ &\geq (n-h-1) + (n+h-1) + n \\ &\geq 3n-2. \end{aligned}$$

D'où $S \geq 3n-2$.

D'autre part, il faut un temps au moins égal à x.s" à l'élément $a_{r,n}$ pour qu'il arrive à la cellule s", où a lieu l'accumulation

$$c_{r,p} := c_{r,p} + a_{r,n}.b_{n,p}.$$

Donc le temps d'exécution de l'algorithme T est supérieur à x.s". Soit

$$\begin{aligned} T &\geq x.s'' \\ &\geq x.(3n-2) \text{ où } x \geq n+2 \\ &\geq 3n^2+4n-4 \ \blacklozenge \end{aligned}$$

Le lecteur intéressé peut retrouver l'interpretation géométrique de ces résultats. Il convient aussi de noter que ce résultat concerne la complexité du problème et est donc plus fort que les résultats de complexité asymptotique établis par Varman et Ramakrishnan [VaR 84]. En effet, ils démontrent que le temps optimal est en $O(c.n^2)$, ce qui est évident si on ne donne pas la valeure exacte de la constante c.

2.6. Produit de matrices triangulaires

Dans un premier temps, nous supposons que

$n = \alpha(x-2)$ avec $\alpha \geq 2$.

Nous présentons alors un algorithme pour le produit de deux matrices triangulaires supérieures de taille nxn. Cet algorithme est défini par les dates d'entrée dans le réseau ci-dessous:

$$T_a[\, i\,,\, p(x-2)+r\,] = i(x-2) + p\alpha(x-2)(x-1) + r(x-1) - 2x + 4$$

$$T_b[\, p(x-2)+r\,,\, j\,] = j\alpha^2(x-2) + p\alpha(x-2) + r - 2x + 4$$

$$T_c[\, i\,,\, j\,] = j\alpha^2(x-1) - i - 2x + 4\,.$$

où $i \leq p(x-2)+r \leq j$; $1 \leq r \leq x-2$ *et* $0 \leq p \leq \alpha-1$; $1 \leq i,j \leq n$.

Comme pour les matrices denses, la preuve de validité sera faite en établissant trois lemmes.

Lemme 2.4: Les données de chacunes des matrices A, B et C entrent dans le réseau à des instants distincts.

Démonstration :

(i) Montrons que les cefficients de A sont lus à des instants distincts.

$T_a[\, i\,,\, p(x-2)+r\,] = T_a[\, i'\,,\, p'(x-2)+r'\,]$ entraîne que

$$(i-i')(x-2) + (p-p')\alpha(x-2)(x-1) + (r-r')(x-1) = 0 \qquad (1)$$

D'où

$$(x-1)(r-r') = 0 \pmod{(x-2)}.$$

Comme x-1 et x-2 sont premiers entre eux, on déduit que

$$r-r' = 0 \pmod{(x-2)}.$$

Par ailleurs, $1 \leq r,r' \leq x-2$, ce qui entraîne que r=r.

En divisant (1) par x-2 on a

$(i-i') + (p-p')\alpha(x-1) = 0$, ce qui entraîne que

$$|i-i'| = |p-p'|\alpha(x-1)$$

$$> |p-p'|n \quad \text{car } n = \alpha.(x-2)$$

Comme $1 \leq i,i' \leq n$ on déduit que $i = i'$ et $p = p'$.

(ii) Montrons que les éléments de la matrice B sont lus à des instants distincts.

$T_b[\, p(x-2)+r\,,\, j\,] = T_b[\, p'(x-2)+r'\,,\, j'\,]$ entraîne que

$$(j-j')\alpha^2(x-2) + (p-p')\alpha(x-2) + r - r' = 0 \qquad (2)$$

ce qui entraîne que

$$r-r' = 0 \pmod{(x-2)}.$$

Comme $1 \leq r,r' \leq x-2$, on déduit que

$r = r'$.

Après simplification de (2) par $(x-2)\alpha$, on a

$(j-j')\alpha + (p-p') = 0$

où $0 \leq p, p' < \alpha$

Donc $p = p'$ et $j = j'$.

(iii) Montrons enfin que les éléments de la matrice C sont lus à des instants différents.

$T_c[i,j] = T_c[i',j']$

entraîne que

$(j-j')\alpha^2 (x-1) - (i-i') = 0$.

Donc

$$|i-i'| = |j-j'| \alpha^2 (x-1)$$
$$\geq |j-j'| \alpha n.$$

Or

$|i-i'| < n$ et $\alpha(x-1) > n$.

D'où $j = j'$ et $i = i'$ ♦

Lemme 2.5: Les éléments $a_{i,p(x-2)+r}$, $b_{p(x-2)+r,j}$ et $c_{i,j}$ se rencontrent dans la cellule d'indice $s = j\alpha^2 - p\alpha(x-2) - r - i$.

Démonstration :

Un simple calcul montre que

$$T_a[i,p(x-2)+r] + x.s = T_b[p(x-2)+r,j] + 2.s$$
$$= T_c[i,j] + s.$$

Il nous faut maintenant montrer que s est toujours *positif* pour $\alpha \geq 2$. On a

$s = j\alpha^2 - p\alpha(x-2) - r - i$

$= \alpha^2(j-(p(x-2)+r)) + \alpha(\alpha-1)p(x-2) + (\alpha^2-1)r - i$,

avec $i \leq p(x-2) + r \leq j$.

Donc

$s \geq \alpha(\alpha-1)\, p\,(x-2) + (\alpha^2-1)\, r - i$

Comme $\alpha \geq 2$, on a

$\alpha(\alpha-1)\, p(x-2) > p(x-2)$ et $(\alpha^2-1)\, r > r$,

donc $s > (j-p(x-2)+r) + (p(x-2)+r-i)$

> 0 ♦

Lemme 2.6: Si $a_{i',p'(x-2)+r'}$, $b_{p''(x-2)+r'',j''}$ et $c_{i,j}$ se rencontrent dans une cellule du réseau alors $p' = p''$, $r' = r''$, $i' = i$ et $j = j''$.

Démonstration :

Notons s le numéro de cellule où a lieu cette rencontre. Cette rencontre se traduit par les équations (1), (2) ci-dessous:

$$T_a[\, i',p'(x-2)+r'] + x.s = T_b[\, p''(x-2)+r'',j''] + 2s \qquad (1)$$

$$T_b[\, p''(x-2)+r'',j''] + 2s = T_c[i,j] + s. \qquad (2)$$

où

$$T_a[\, i',p'(x-2)+r'] = i'(x-2) + p'\,\alpha\,(x-2)(x-1) + r'(x-1) - 2.x + 4$$

$$T_b[\, p''(x-2)+r'',j''] = j''\,\alpha^2(x-2) + p''\,\alpha\,(x-2) + r'' - 2x + 4$$

et

$$T_c[i,j] = j\,\alpha^2(x-1) - i - 2.x + 4$$

(1) entraîne que

$$s = j''\,\alpha^2 + p''\,\alpha - i' - r' - p'\,\alpha\,(x-1) + (r''-r')/(x-2).$$

Or

$$|\, r'' - r' \,| < x-2,$$

donc $r' = r''$.

D'où

$$s = j''\,\alpha^2 + p''\,\alpha - i' - r' - p'\,\alpha\,(x-1) \; . \qquad (3)$$

(2) entraîne que

$$s = j\,\alpha^2(x-1) - i - j''\,\alpha^2(x-2) - p''\,\alpha\,(x-2) - r''. \qquad (4)$$

comme $r1 = r2$, (3) et (4) entraînent que

$$(j-j'')\,\alpha^2(x-1) + (p'-p'')\,\alpha\,(x-1) + i' - i = 0 \; . \qquad (5)$$

Comme

$$|\, i'-i \,| < n \;,\; \alpha(x-1) > n \text{ et } i'-i \text{ est multiple de } \alpha(x-1)$$

on déduit que $i'=i$.

La formule (5) devient alors

$$(j-j'')\,\alpha + (p'-p'') = 0 \; . \qquad (6)$$

où $|p'-p''| < \alpha$,

d'où

$$p' = p'' \text{ et } j = j'' \; \blacklozenge$$

Le lemme 2.5 montre que toutes les accumulations correspondant au produit C=A.B sont effectuées pendant le déroulement de l'algorithme. Le lemme 2.6 montre qu'aucune accumulation indésirable n'a lieu.

Analysons maintenant les performances de cet algorithme en temps et en nombre de cellules. D'après le lemme 2.5, $a_{i,p(x-2)+r}$, $b_{p(x-2)+r,j}$ et $c_{i,j}$ se rencontrent dans une cellule d'indice

$$s = j\,\alpha^2 - p\,\alpha(x-2) - r - i.$$

L'entier s est toujours positif. Il atteint son maximun pour

$$j=n \;,\; p=0 \text{ et } r=i=1$$

et vaut alors

$n\alpha^2-2$.

Comme la valeur α^2-2 est atteinte , l'algorithme s'exécute sur un réseau de taille

$S = n\alpha^2 + O(1)$

Lorsqu'on compare ce résultat avec le nombre

$S_1 = (\alpha^2+\alpha+1/\alpha)\, n + O(1)$

obtenu pour le produit de matrices denses, on constate un gain de $(\alpha+1/\alpha)\, n$ cellules dans le cas de matrices triangulaires.

Analysons maintenant le temps d'exécution de l'algorithme.

- Le premier élément qui entre dans le réseau est $a_{1,1}$.
- Le dernier élément de C qui est calculé est $c_{1,n}$.

L'algorithme s'exécute donc en temps

$T = T_c[1,n] + S - T_a[1,1]$

$= (n\alpha^2(x-1)-1 - 2x + 4) + S - (x-2 + x-1 -2x+4)$

$= \alpha n^2 + O(n)$.

Lorsqu'on compare ce résultat au temps $T_1 = (\alpha + 1 + 1/\alpha^2)\, n^2 + O(n)$ obtenu pour les matrices denses, on constate un gain en temps de $(1+1/\alpha^2)\, n^2$.

Examinons maintenant le cas $x= n+2$. Nous avons déja vu que pour cette valeur de x le produit de deux matrices denses s'effectue de manière efficace. Nous donnons ci-dessous un algorithme, pour le produit de deux matrices triangulaires inférieures, s'exécutant sur un réseau de taille $S=n$ et en temps $T= n^2+3n$.

- Cet algorithme est *optimal en surface*. En effet, le calcul de l'élément $c_{1,n}$ nécessite n accumulations qui sont effectuées dans des cellules *différentes*
- Il est *asymptotiquement optimal en temps*. En effet, pour qu'un élément de la matrice A arrive à la cellule n, il lui faut un temps supérieur à $n(n+2)=n^2+2n$.

Cet algorithme est défini par les matrices des temps d'entrée dans le réseau des données, T_a , T_b , T_c ci-dessous:

$T_a[\,i,k\,] = n + n.i - k.(n+1) + 1$

$T_b[\,k,j\,] = n^2+2n - k - n.j + 1$

$T_c[\,i,j\,] = n^2+3n - i - j.(n+1) + 2$

Pour $n \geq i \geq k \geq j \geq 1$.

On vérifie que

$T_a[i,j] \geq 0$, $T_b[i,j] \geq 0$, $T_c[i,j] \geq 0$

et que c[i,j] rencontre (a[i,k] , b[k,j]) dans la cellule de numéro
$s = 1+n-j+k-i$.
L'entier s ainsi obtenu est toujours positif car
$n \geq i \geq k \geq j \geq 1$.
Le détail de la preuve de validité de cet algorihme peut s'effectuer en suivant la même démarche que pour le cas $\alpha \geq 2$.

Exemple 2.5: n=3 et x=5.

Pour $x = 5$ et $n = 3$, les matrices des temps d'entrée des données dans le réseau sont :

$$T_a = \begin{pmatrix} 3 & x & x \\ 6 & 2 & x \\ 9 & 5 & 1 \end{pmatrix} \quad T_b = \begin{pmatrix} 12 & x & x \\ 11 & 8 & x \\ 10 & 7 & 4 \end{pmatrix} \quad T_c = \begin{pmatrix} 15 & x & x \\ 14 & 10 & x \\ 13 & 9 & 5 \end{pmatrix}$$

Les points de calculs sont représentés dans la figure 2.14. L'algorithme correspondant effectue correctement le produit de deux matrices triangulaires d'ordre 3, sur un réseau composé de 3 cellules et en temps $T = 18$ cycles.

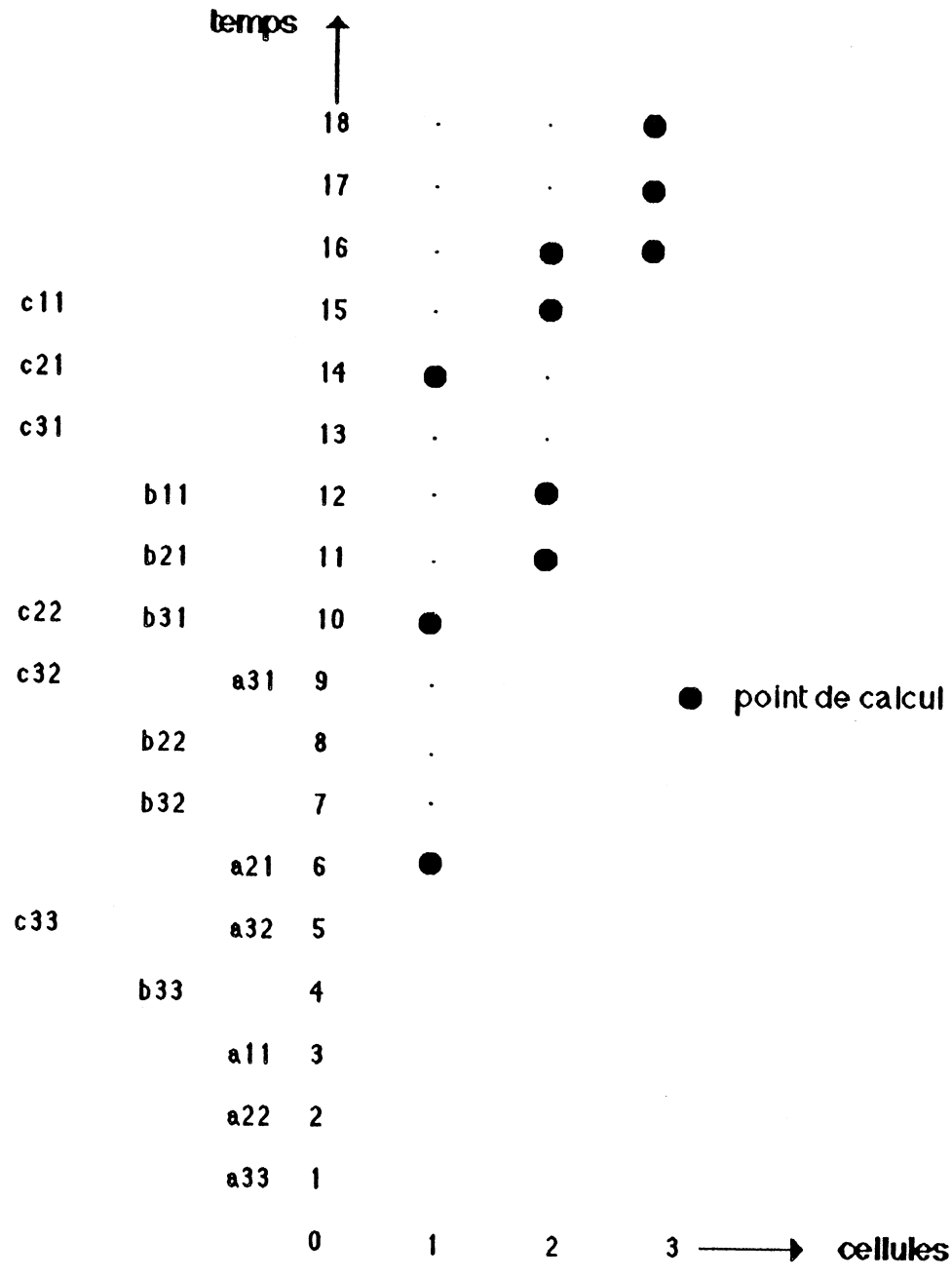


Figure 2.14: diagramme de calcul de l'exemple 2.5

2.7. Influence du nombre de canaux

Sur un réseau systolique donné, il se peut qu'on ait plus de trois canaux disponibles pour les transferts de données. Nous allons montrer comment les canaux supplémentaires peuvent être exploités pour accélérer le calcul du produit matriciel.

Une approche évidente consisterait à répartir ces canaux en triplets, chaque triplet étant dédié au calcul d'une famille de coefficients de C. Cette approche évidente impliquerait que les coefficients $c_{i,j}$ appartenant à des blocs différents puissent être mis à jour simultanément dans une cellule, ce qui nécessite des cellules ayant plusieurs additionneurs et multiplieurs.

L'approche que nous adoptons ici est différente. Nous supposons en effet, qu'on a plus de trois canaux mais qu'une cellule ne contient qu'un multiplieur et qu'un additionneur. A chaque instant, chaque cellule doit donc mettre à jour un seul des $c_{i,j}$ qui se présentent à l'entrée de ses unités fonctionnelles.

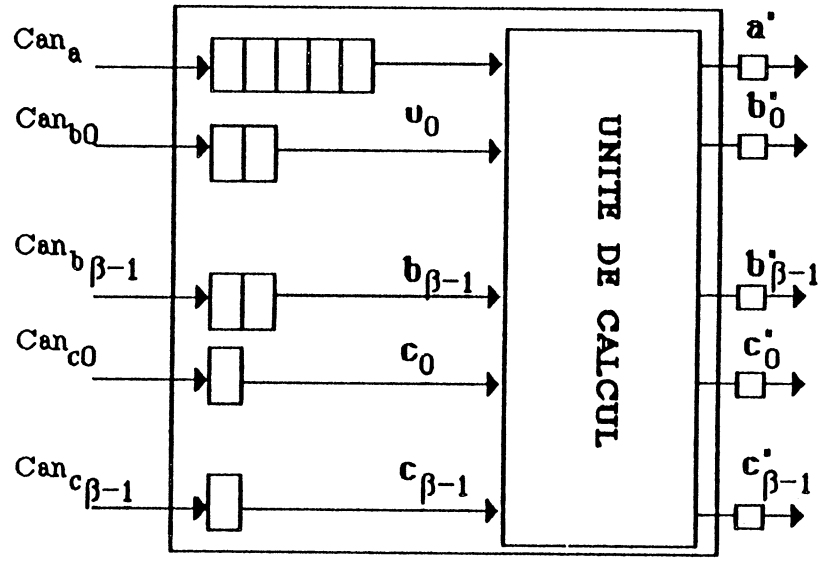
En fait, notre étude permet d'illustrer comment le nombre d'entrées/sorties simultanées influe sur les performances des algorithmes, pour le produit de deux matrices denses de tailles nxn, sur un réseau unidirectionnel. En effet, les cellules extrêmes sont les seules à pouvoir effectuer des opérations d'entrées/sorties.

Ce problème du compromis entre le temps de calcul et le nombre maximun d'entrées/sorties simultanés a été abordé par L. Melkemi et M.Tchuente [MeT 86], pour le produit de deux matrices carrées d'ordre nxn sur des réseaux bidimensionnels de taille nxn. Ces auteurs présentent en effet, des algorithmes qui s'exécutent en temps 3n-2, 2.5n-2 ou 2n-2, selon que le nombre maximun d'entrées/sorties simultanées est 2n , 3n ou 4n.

Considérons un réseau unidirectionnel où chaque cellule possède $(2\beta+1)$ canaux d'entrées/sorties (cf. figure 2.15), où β est un entier compris entre 1 et α. Nous nous limitons à cet intervalle car les algoritmes que nous proposons nous permettent d'exploiter au plus $2\alpha+1$ canaux. En fait ces algorithmes ont les propriétés suivantes:

- Un seul canal est réservé à la matrice A
- β canaux , numérotés de 0 à β-1, sont réservés aux éléments de B, mais à chaque instant au plus l'un d'entre eux , de numéro h, est activé pour la lecture d'un élément $b_{k,j}$ tel que h = (j-1) modulo β. De plus tout élément $b_{k,j}$ lu sur un canal reste sur ce même canal pendant toute la traversée du réseau.
- β canaux , numérotés de 0 à β-1, sont réservés aux éléments de C. Un $c_{i,j}$

est lu avec une valeur initialisée à 0, sur le canal numéro h tel que h = (j-1) modulo β. De plus tout élément $c_{i,j}$ lu sur un canal reste sur ce même canal pendant toute la traversée du réseau. Il convient de noter ici, qu'une cellule peut avoir à un instant donné plusieurs éléments $c_{i,j}$ placés à l'entrée de ses éléments fonctionnels et placés dans des canaux différents. Cependant, elle ne peut effectuer que la seule opération correspondante aux données qui se trouvent dans Can_a, Can_{bh} et Can_{ch}, pour un unique (s'il existe) indice h , $0 \leq h \leq \beta-1$.



a' := a ;

pour j : = 0 à β-1 faire $b'_j := b_j$

pour j := 0 à β-1 faire

si ($c_j \neq$ nil et $b_j \neq$ nil) alors

$c'_j := c_j + a.b_j$

sinon $c'_j := c_j$.

Figure 2.15. cellule de base .

Comme leur nom l'indique, les canaux Can_a, Can_{bk} et Can_{ck}; $0 \leq k \leq \beta-1$, sont empruntés respectivement par les éléments $a_{i,j}$, $b_{i,j}$ et $c_{i,j}$.

Noter qu'une cellule a une seule unité de calcul et qu'elle ne peut donc effectuer qu'une seule opération d'accumulation à un instant donné. Cependant, la multitude des canaux d'entrées/sorties permet d'éliminer les calculs indésirables en faisant passer les $b_{k,j}$ et $c_{i,j}$ par des canaux différents. L'algorithme est défini de telle sorte qu'une cellule ait une seule opération à effectuer.

L'algorithme est défini par le temps d'entrée dans le réseau de chaque donnée. Bien sûr nous devons spécifier le numéro de canal, lorsqu' il s'agit d'un coefficient de la matrice B ou de la matrice C.

On pose

$T_o=n^2(1/\alpha^2+1)+n(\alpha-1)$

$1 \le i,j \le n$, $0 \le p \le \alpha-1$ et $1 \le r \le x-2$:

On a alors:

- $T_a[p(x-2)+r,k] = T_o-r(x-2)-(k-1)(x-1)-p(x-2)(n+\alpha-1)$,
  et $a_{p(x-2)+r,k}$ est lue sur Can_a .
  Pour les autres instants on lit 0 sur Can_a.
- $T_b[k,j] = T_o+\alpha\tau(j-1)(x-2)-(k-1)$,
  où $\tau = \alpha/\beta$ (on suppose donc que β divise α)
  Cette lecture se fait sur le canal numéro $(j-1) \bmod \beta$
  Dans les autres cas on lit la valeur nil.
- $T_c[\, p(x-2)+r\, ,\, j\,] = T_o+ \alpha\tau(j-1)(x-1) +p\,(n+\alpha-1) +r$,
  Cette lecture se fait sur le canal numéro $(j-1) \bmod \beta$
  Dans les autres cas on lit la valeur nil.

T_o est calculé de sorte que les dates d'entrée dans le réseau soient positives. Rappelons qu'un calcul est effectué dans une cellule s lorsqu'il existe k, $0 \le k \le \beta-1$, tel que le canal d'entrée Can_{bk} contient une donnée différente de nil.

Exemple 2.6: *x-1= 2, n = 4 et* $\beta = 2$.

Pour ce cas, l'algorithme est défini par les matrices des temps d'entrée dans le réseau ci-dessous:

$$T_a = \begin{pmatrix} 22 & 19 & 16 & 13 \\ 20 & 17 & 14 & 11 \\ 12 & 9 & 6 & 3 \\ 10 & 7 & 4 & 1 \end{pmatrix} \quad T_b = \begin{pmatrix} 24 & 28 & 32 & 36 \\ 23 & 27 & 27 & 35 \\ 22 & 26 & 30 & 34 \\ 21 & 25 & 29 & 33 \end{pmatrix} \quad T_c = \begin{pmatrix} 25 & 31 & 37 & 43 \\ 26 & 32 & 38 & 44 \\ 30 & 36 & 42 & 48 \\ 31 & 37 & 43 & 49 \end{pmatrix}$$

La première et la troisième ligne de C sont lues dans le canal numéro 0, tandis que la deuxième et la quatrième ligne de C sont lues dans le canal numéro 1.
La même remarque est valable pour la matrice B.

On voit que si $T_c[i,j] = T_c[i',j']$ alors $c_{i,j}$ et $c_{i',j'}$ entrent dans le réseau par des canaux différents. Par exemple $T_c[1,3] = T_c[4,2]$ mais il n'y a pas de conflit d'entrée car ces deux éléments entrent dans le réseau par des canaux différents.
Les matrices T_a, T_b et T_c définissent un algorithme qui s'exécute en temps T =65, sur un réseau de 16 cellules .

Sur la figure 2.6, sont représentés les deux premiers pavés du diagramme de calcul correspondant à cet exemple.

- Le pavé (1) qui correspond aux accumulations effectuées par les éléments $c_{i,1}$, $1 \leq i \leq 4$. Ces calculs correspondent aux éléments portés par les canaux Can_a, Can_{b0} et Can_{c0}.

- Le pavé (2) obtenu par translation de 4(1,x-1) du pavé(1). Il correspond aux accumulations effectuées par les éléments $c_{i,2}$, $1 \leq i \leq 4$. Ces calculs correspondent aux éléments portés par les canaux Can_a, Can_{b1} et Can_{c1}.

De la même manière, pavé (3) s'obtient à partir de pavé (2), par translation de vecteur 4(1,x-1).

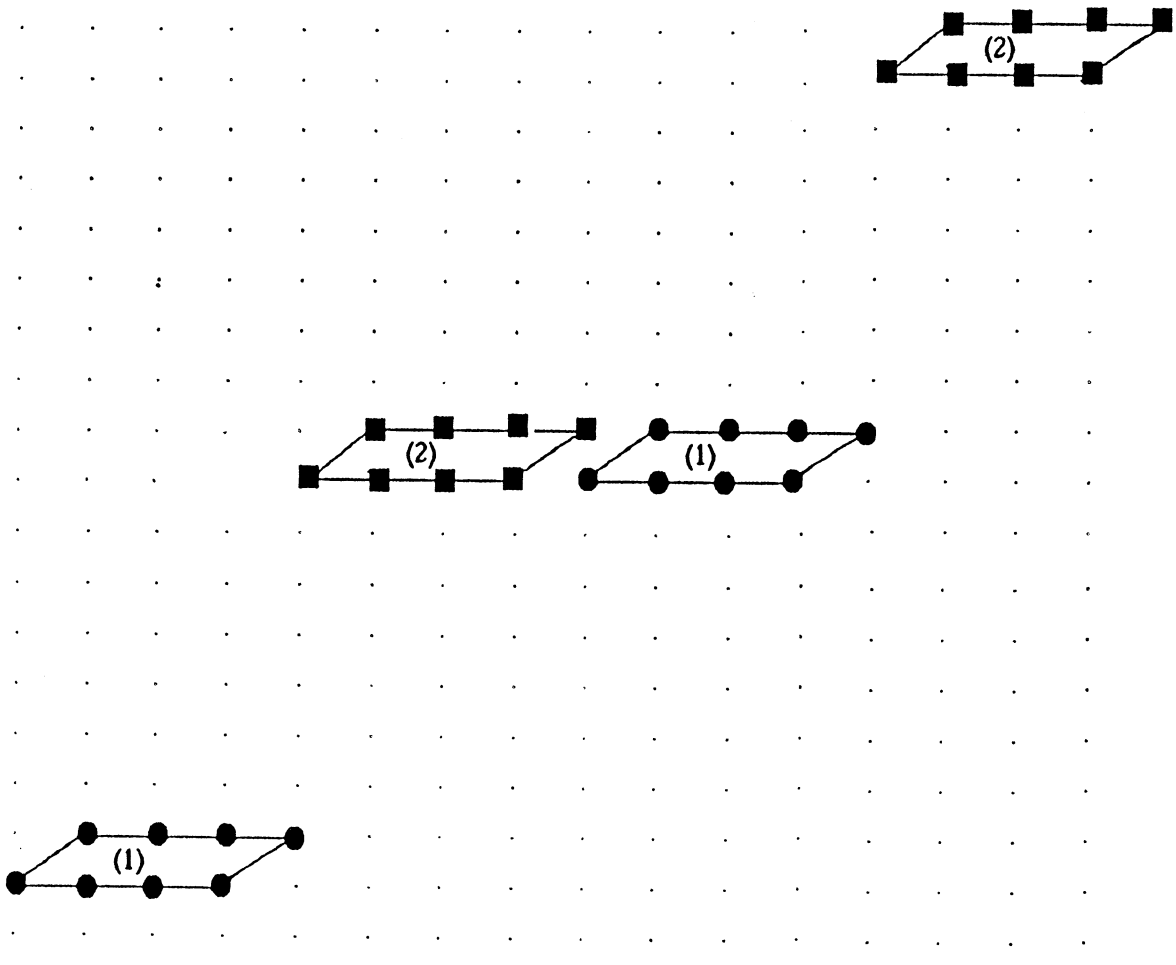


Figure 2.16: diagramme de calcul de l'exemple 2.6

Nous allons établir trois lemmes qui prouvent la correction de l'algorithme.

Lemme 2.7 Deux éléments de la même matrice ayant la même date d'entrée dans le réseau empruntent deux canaux d'indices différents.

Démonstration :
(i) Les dates d'entrées des $a_{i,j}$ sont identiques à celles de ALG1. Par conséquent le lemme 2.1, montre que les $a_{i,j}$ dans le réseau à des dates différentes.

(ii) Examinons maintenant la matrice B.

$$T_b[k,j] = T_b[k',j']$$

entraîne que

$$\alpha\tau(j-j') (x-2) -(k-k') = 0. \qquad (1)$$

où $\tau \geq 1$, $\alpha(x-2) = n$ et $| k-k' | < n$,

ce qui entraîne que

$$j = j' \text{ et } k = k'.$$

(iii) Montrons que le résultat est vrai pour la matrice C.
Si $c_{p(x-2)+r,j}$ et $c_{p'(x-2)+r',j'}$ entrent dans le réseau par un même canal et au même instant, alors

$$T_c[p(x-2)+r,j] = T_c[p'(x-2)+r',j']$$

et

$$(j-j') \bmod \beta = 0.$$

D'où

$$\alpha\tau(j-j') (x-1) + (p-p') (n+\alpha-1) + r-r' = 0 . \qquad (3)$$

où $| p-p' | \leq \alpha-1$, $| r-r' | < x-2$, $n = \alpha(x-2)$ et $\alpha > 1$

On déduit que

$$\begin{aligned} \tau(n+\alpha)| j-j' | &= | (p-p')(n+\alpha-1)+r-r'| \\ &\leq | (p-p') (n+\alpha-1) | + | r-r'| \\ &< (\alpha-1)(n+\alpha-1)+x-2 \\ &< (\alpha-1)(n+\alpha-1)+n/\alpha \qquad (4) \end{aligned}$$

Comme

$$j-j'=0 \bmod \beta$$

on peut poser

$$| j-j' | = y.\beta.$$

Comme $\tau\beta = \alpha$, on déduit de l'inéquation (4) que

$$\begin{aligned} \alpha(n+\alpha).y &< (\alpha-1)(n+\alpha-1)+n/\alpha \\ &< \alpha(n+\alpha) - (\alpha + n + \alpha -1-n/\alpha) \\ &< \alpha(n+\alpha). \end{aligned}$$

Ce qui entraîne que

$y=0$,

ou encore que

$j=j'$.

La formule (3) devient alors

$$| p-p' | (n+\alpha-1) = | r-r' |.$$

Comme

$$\alpha-1 \geq 0 \text{ et } | r - r' | < n ,$$

on déduit que

$$p = p' \text{ et } r = r'. \quad \blacklozenge$$

Le lemme 2.7 montre en fait que tous les éléments de la matrice B entrent dans le réseau à des dates différentes. Ceci assure qu'à chaque instant une cellule effectue au plus une opération d'accumulation.

Lemme 2.8: L'élément $c_{p(x-2)+r,j}$ rencontre les éléments $a_{p(x-2)+r,k}$ et $b_{k,j}$ dans la cellule de numéro $s = \alpha\tau(j-1)+p(n+\alpha-1)+k+r-1$.

Démonstration :

Un simple calcul montre que

$$T_a[p(x-2)+r,k]+x.s = T_b[k,j]+2s$$
$$= T_c[p(x-2)+r,j]+s$$

où $s = \alpha\tau(j-1)+p(n+\alpha-1)+k+r-1$.

Montrons maintenant que le réseau n'effectue pas de calcul indésirable.

Lemme 2.9 Si $c_{p(x-2)+r,j}$ rencontre $a_{p'(x-2)+r',k'}$ et $b_{k'',j''}$ et si $(j-j'')=0 \mod \beta$ alors $p=p'$, $r=r'$, $k'=k''$ et $j=j''$.

Démonstration :

Supposons que $c_{p(x-2)+r,j}$ rencontre $a_{p'(x-2)+r',k'}$ et $b_{k'',j''}$ dans une cellule de numéro s. On a alors

$$T_a[p'(x-2)+r',k] + x.s = T_b[k'',j''] + 2s \qquad (1)$$
$$T_b[k'',j''] + 2s = T_c[p(x-2)+r,j] + s \qquad (2)$$

où

$$T_a[p'(x-2)+r',k] = T_o-(k'-1)(x-1)-(x-2)r'- p'(n+\alpha-1)(x-2)$$
$$T_b[k'',j''] = T_o+ \alpha\tau(j''-1)(x-2) - (k''-1)$$

et

$$T_c[p(x-2)+r,j] = T_o+ \alpha\tau(j-1)(x-1)+p(n+\alpha-1)+r$$

(1) entraîne que

$$s = \alpha\tau(j''-1)+p'(n+\alpha-1)+r'+k'-1+(k'-k'')/(x-2) \qquad (3)$$

(2) entraîne que

$$s = \alpha\tau(j-1)(x-1) - \alpha\tau(j''-1)(x-2)+k''-1+p(n+\alpha-1)+r \qquad (4)$$

D'où

$$\alpha\tau(j-j'')(x-1)+(p-p')(n+\alpha-1)+k''-k'+r-r'+(k''-k')/(x-2) = 0 \qquad (5)$$

Comme

$$| j-j'' | = 0 \bmod \beta,$$

on peut noter

$$|j-j''| = y.\beta$$

où β est un entier

L'equation (5) entraîne alors que

$$y.\beta.\ \alpha\tau(x-1) \leq | p-p' |(n+\alpha-1)+|k''-k'| + |k''-k'|/(x-2)+|r-r'| \qquad (6)$$

Donc $x-2$ divise $k'' - k'$.

Comme

$$1 \leq k', k'' \leq n \text{ avec } n = \alpha(x-2)$$

on deduit que

$$|k''-k'| \leq (\alpha-1)(x-2).$$

Or

$$\beta\tau = \alpha,\ \alpha(x-1) = n+\alpha,\ |p-p'| \leq \alpha-1 \text{ et } |r-r'| < x-2.$$

L'inéquation (6) entraîne que

$$\begin{aligned} y.(n\alpha+\alpha^2) &< (\alpha-1)(n+\alpha-1) + (\alpha-1)(x-2) + (\alpha-1) + x-2 \\ &< n.\alpha+\alpha(\alpha-1) \\ &< n\alpha+\alpha^2. \end{aligned}$$

Ceci n'est possible que si $y=0$, soit $j=j''$.

L'inéquation (5) devient

$$(p-p')(n+\alpha-1)+k''-k'+r-r'+(k''-k')/(x-2) = 0 \qquad (7)$$

D'où

$$\begin{aligned} | p-p' | (n+\alpha-1) &\leq | k''-k'| + | r-r'| + | (k''-k')/(x-2)| \\ &< (\alpha-1)(x-2) + x-2 + \alpha-1 \\ &< n+\alpha-1. \end{aligned}$$

D'où

$$p=p'.$$

La formule (7) devient

$$k''-k'+r-r'+(k''-k')/(x-2) = 0$$

où $| r-r' | < x-2$ et $x-2$ divise $k''-k'$.

Ceci entraîne que $k'=k''$ et $r=r'$. ♦

Cet algorithme généralise celui que nous avons exposé dans la section précédente car pour $\beta=1$ on retrouve l'algorithme que nous avons déja présenté.

Analysons maintenant les performances de cet algorithme.

- D'après le lemme 2.8, les éléments $a_{p(x-2)+r,k}$, $b_{k,j}$ et $c_{p(x-2)+r,j}$ se rencontrent dans la cellule de numéro

$$s = \alpha\tau(j-1)+p(n+\alpha-1)+k+r-1.$$

Cet indice s est toujours positif. Il est minimum pour

$$j = 1, p=0, k = r = 1$$

et vaut alors 1. Par ailleurs, il atteint son maximun pour

$$j = k = n\ ,\ p = \alpha-1 \text{ et } r = x-2,$$

il vaut alors

$$\alpha\tau(n-1)+(\alpha-1)(n+\alpha-1)+n+x-2-1 = n(\alpha\tau+\alpha+1/\alpha)- \alpha\tau+\alpha^2-2\alpha.$$

où $\tau = \alpha/\beta$, $\alpha \geq 2$.

Lorsqu'on compare ceci à la taille

$$S_1 = n(\alpha^2+\alpha+1/\alpha) -2\alpha.$$

du réseau obtenu précédemment dans le cas particulier $\beta = 1$, on s'aperçoit que le gain en nombre de processeurs correspond au facteur multiplicatif

$$(\alpha^2+\alpha+1/\alpha) / (\alpha\tau+\alpha +1/\alpha)$$

qui lorsqu'on néglige $1/\alpha$, est égal à

$$(\alpha+1) / (\tau+1)$$

Cette quantité est elle même proche de β. On peut donc dire que lorsque le réseau comporte $(2\beta+1)$ canaux le nombre de cellules est réduit d'un facteur approximativement égal à β, ce qui correspond à la réduction maximale qu'on pouvait espérer.

Examinons maintenant le temps d'exécution de l'algorithme.

- Le premier élément a être lu par le réseau est $a_{n,n}$
- Le dernier élément calculé est $c_{n,n}$

Le temps d'exécution de l'algorithme est donc

$$T_c[n,n] + S - T_a[n,n]$$
$$= \alpha\tau(n-1)(x-1)+(\alpha-1)(n+\alpha-1)+x-2+S+(x-2)^2+(n-1)(x-1)$$
$$+(\alpha-1)(x-2)(n+\alpha-1)$$
$$= n^2 (\tau+1/\alpha^2+1/\alpha)$$

Lorsqu'on compare ceci au temps

$$T_1 = n^2(\alpha+1+1/\alpha^2) +O(n)$$

de l'algorithme obtenu précédemment dans le cas particulier $\beta = 1$, on s'aperçoit que le gain en temps correspond au facteur multiplicatif

$$(\alpha+1+1/\alpha^2) / (\tau+1/\alpha +1/\alpha^2)$$

qui est proche de

$$\alpha/ \tau = \beta.$$

On peut donc dire que lorsque le réseau comporte $(2\beta+1)$ canaux le temps est divisé par un facteur approximativement égal à β, ce qui correspond à l'accélération maximale qu'on pouvait espérer.

La figure 2.16 donne une interprétation géométrique de ce résultat, dans le cas où

$$x-1 = 2 \ , \beta = 2 \text{ et } n = 4.$$

En effet, la présence de β canaux pour les $c_{i,j}$ permet de calculer plusieurs $c_{i,j}$ dans des canaux différents. β segments appartenant à des pavés différents peuvent donc appartenir à la même ligne horizontale. Ceci permet de diviser par un facteur proche de β la longueur du vecteur de translation qui permet d'obtenir le pavé (j) à partir du pavé (j-1).

Conclusion

Nous avons proposé une formulation combinatoire au problème du produit matriciel sur les réseaux systoliques, unidirectionnels, composés de cellules combinatoires. Cette formulation varie suivant les contraintes imposées par le problème. Dans ce chapitre, nous nous sommes limités au cas des réseaux unidirectionnels composés de cellules combinatoires. Dans le dernier paragraphe, nous avons fait varier le nombre des canaux d'entrées / sorties du réseau, et nous avons adapté la formulation combinatoire à ce cas. A l'aide de cette formulation nous avons conçu:

- un algorithme pour effectuer le produit C= A.B, qui s'exécute sur un réseau unidirectionnel, composé de cellules combinatoires, ayant chacune un buffer local de longueur x = $O(n/\alpha)$. Les performances atteintes par cet algorithme sont en $O((\alpha^2+\alpha+1/\alpha)n)$ pour la surface du réseau et en $O((\alpha+1+1/\alpha^2)n^2)$ pour le temps de calcul.

- un algorithme pour le produit C = A.B qui s'exécute sur le même réseau, où A et B sont des matrices triangulaires. La surface requise pour le réseau, dans ce cas, est de l'ordre de $O(\alpha^2 n)$ et le temps d'exécution est en $O(\alpha n^2)$. Lorsque x= n+2, l'algorithme obtenu est optimal en temps et en surface, si on s'astreint à lire chaque donnée une seule fois.

- un algorithme pour le produit C = A.B qui s'exécute sur un réseau possédant β canaux d'entrées / sorties; où β est un paramètre. Ainsi, nous avons montré, à travers les algorithmes que nous avons proposés, que la taille du réseau et le temps d'exécution croissent de manière quasi-linéaire avec $1/\beta$.

Chapitre 3

RESEAUX LINEAIRES AVEC SIGNAUX DE CONTROLE

3.1. Introduction

Dans le chapitre précédent, nous avons considéré le cas le plus simple où l'on dispose d'un réseau linéaire unidirectionnel sans signaux de contrôle. Dans de tels réseaux, toute rencontre entre trois coefficients $a_{i,k}$, $b_{k,j}$ et $c_{i,j}$ au cours d'un algorithme de produit de deux matrices, donne lieu à une accumulation. Nous avons montré que dans ce cas les algorithmes de produit de deux matrices admettent une interprétation géométrique qui consiste à immerger le graphe de dépendance dans Z^2 avec trois contraintes qui lorsque Buf_a , Buf_b et Buf_c sont de longueurs respectives x, 2 et 1 sont les suivantes:

- Si P(i,j,k) et P(i',j',k') sont affectés à une droite de pente x , alors i=i' et k=k'.

- Si P(i,j,k) et P(i',j',k') sont affectés à une droite de pente 2 , alors j=j' et k=k'.

- Si P(i,j,k) et P(i',j',k') sont affectés à une droite de pente 1, alors i=i' et j=j'.

- Pour tout couple (i,j), $1 \leq i,j \leq n$, il existe une droite de pente +1, contenant les n points de calcul P(i,j,k), $1 \leq k \leq n$ et ne contenant pas d'autres points de calcul.

Dans le cas où les réseaux comportent des canaux supplémentaires pouvant transmettre des signaux de contrôle, la dernière contrainte doit être remplacée par une condition plus faible, à savoir que

- Pour tout couple (i,j), $1 \leq i,j \leq n$, il existe une droite de pente +1 contenant les n points de calcul P(i,j,k), $1 \leq k \leq n$.

En effet, si une telle droite destinée à contenir la trajectoire de $c_{i,j}$ passe par d'autres points de calcul, ces derniers peuvent être invalidés grâce aux signaux de contrôle. La conception d'un algorithme sur des réseaux ayant des canaux supplémentaires pour les signaux de contrôle, comporte donc moins de contraintes dans l'immersion du graphe de dépendance dans Z^2.

Cette liberté supplémentaire va être exploitée notamment pour lire plusieurs fois certaines données. Ceci est important car tous les algorithmes optimaux de produit matriciel pour réseaux systoliques bidimensionnels, et qui sont dus essentiellement à L. Melkemi et M. Tchuente [MeT 85] comportent plusieurs lectures de certains coefficients $a_{i,k}$ ou $b_{k,j}$. On peut donc espérer améliorer comme en dimension deux, les performances des algorithmes de produit matriciel sur réseaux linéaires, en lisant certaines données plusieurs fois. Rappelons que la formulation combinatoire [MeT 85] consiste tout simplement à affecter les $c_{i,j}$ à un domaine de Z^2 construit en fonction de la taille du réseau et de celle des matrices, de sorte que :

- Si $c_{i,j}$ et $c_{h,q}$ appartiennent à une même droite de pente +1 alors $i = h$
- Si $c_{i,j}$ et $c_{h,q}$ appartiennent à une même droite de pente -1 alors $j = q$.

Il convient ici de noter, que seule l'approche combinatoire permet de gérer ces multiples lectures. C'est pourquoi Varman et Ramakrishnan, dont tout les articles sont basés sur une approche analytique font toujours l'hypothèse d'une lecture unique des données. Au prochain paragraphe, nous allons présenter les algorithmes systoliques optimaux présentés par L. Melkemi et M. Tchuente pour le produit matriciel sur réseaux bidimensionnels. Nous traiterons ensuite, les réseaux unidirectionnel avec signaux de contrôle et enfin les réseaux linéaires bidirectionnels.

3.2. Algorithmes optimaux en dimension 2

Nous commençons par le cas où les coefficients $a_{i,k}$ et $b_{k,j}$ circulent horizontalement dans des directions opposées, tandis que $c_{i,j}$ circule de haut en bas. L'algorithme de la figure 3.1 est optimal lorsqu'on s'impose de lire chaque $a_{i,k}$ et $b_{k,j}$ une seule fois. Il s'exécute en temps $T=4n-2$ sur un réseau de taille $n(2n-1)$.

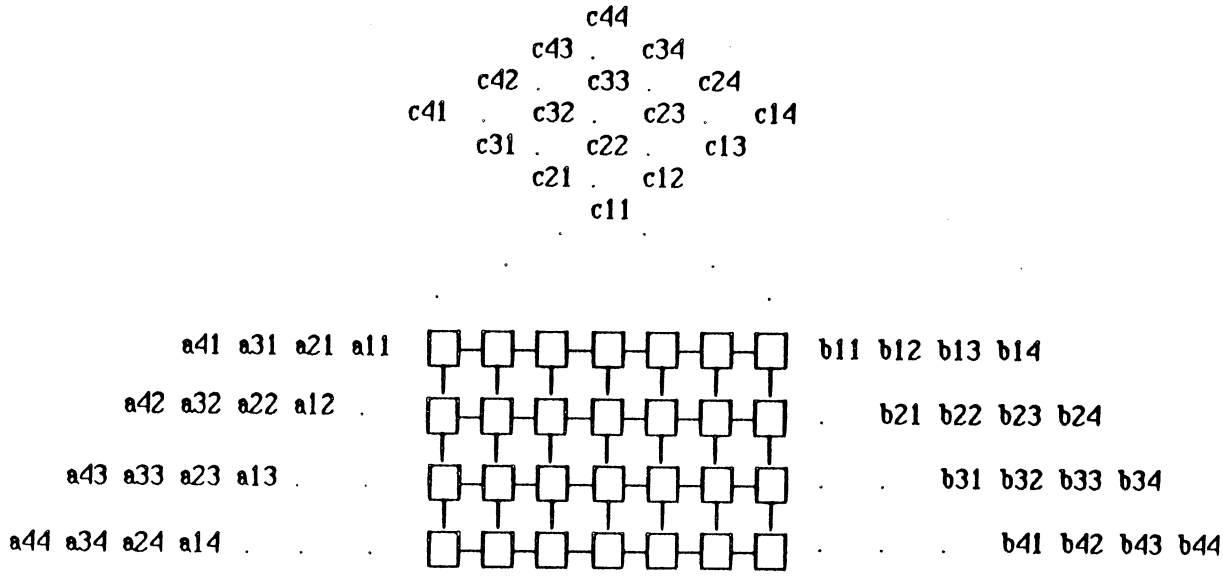


Figure 3.1

Lorsque les $a_{i,k}$ et les $b_{k,j}$ peuvent être lues plusieurs fois, on montre que le temps minimum nécessaire pour multiplier deux matrices nxn sur cette classe de réseaux est T=3n-2; de plus le plus petit réseau pour lequel cette borne est atteinte est de taille nxn ou n(n+1) selon que n est impair ou pair. La figure 3.2 donne l'algorithme optimal pour n=3.

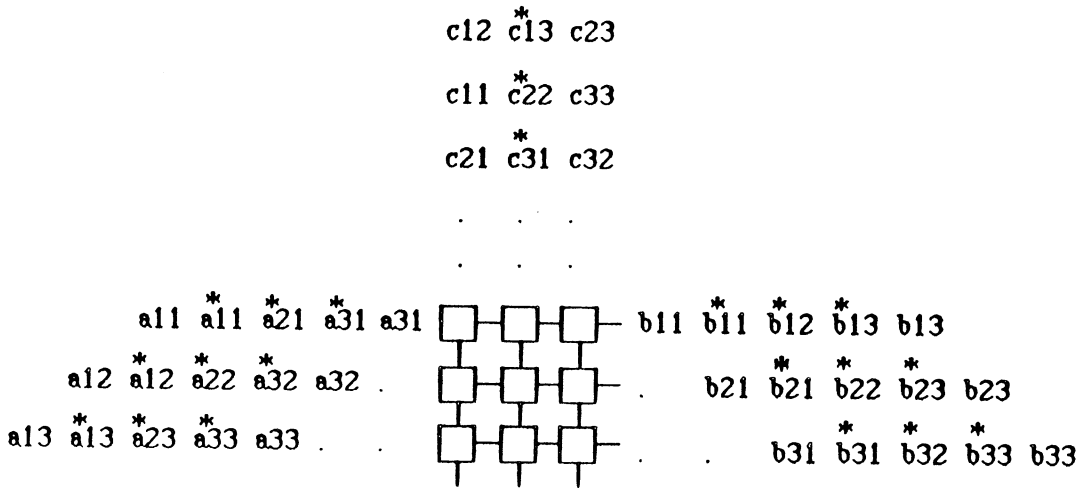


Figure 3.2

Considérons maintenant les réseaux à connexions orthogonales. Lorsqu'on impose de lire une seule fois les $a_{i,k}$, $b_{k,j}$, le meilleur algorithme a été trouvé indépendamment par Guibas, Kung et Thompson [GKT 79], Preparata et Vuillemin [PRV 80], et Katona [Kat 80] (cf. figure 3.3). Il s'exécute en temps T=4n-2 sur un réseau de taille nxn. Ce temps d'exécution comporte 3n-1 étapes de calcul, les n-1 dernières étapes sont consacrées à la sortie des résultats.

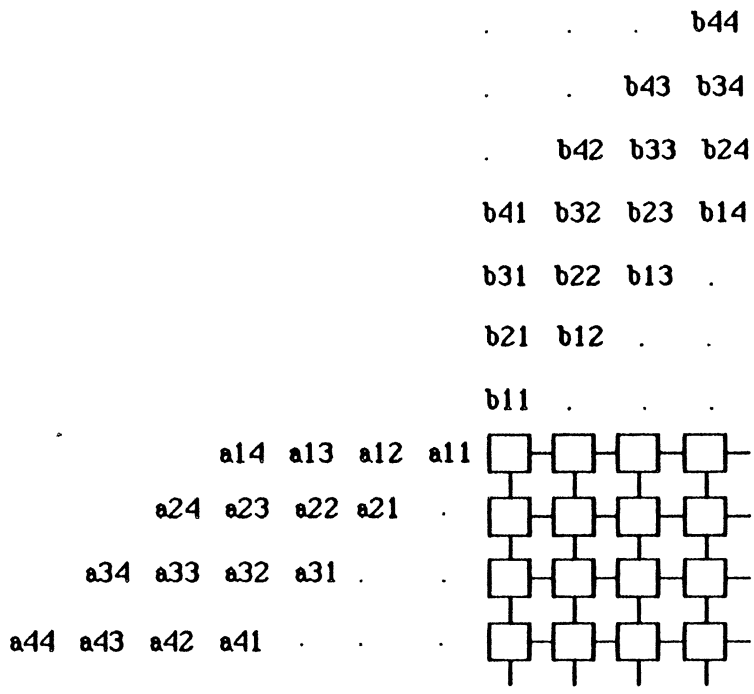


Figure 3.3

Examinons le cas des réseaux hexagonaux. L'algorithme de la figure 3.4 s'exécute en temps T=5n-2 sur un réseau de taille $3n^2 + O(n)$. C'est la meilleure solution que l'on puisse obtenir si l'on s'impose de lire une seule fois les données.

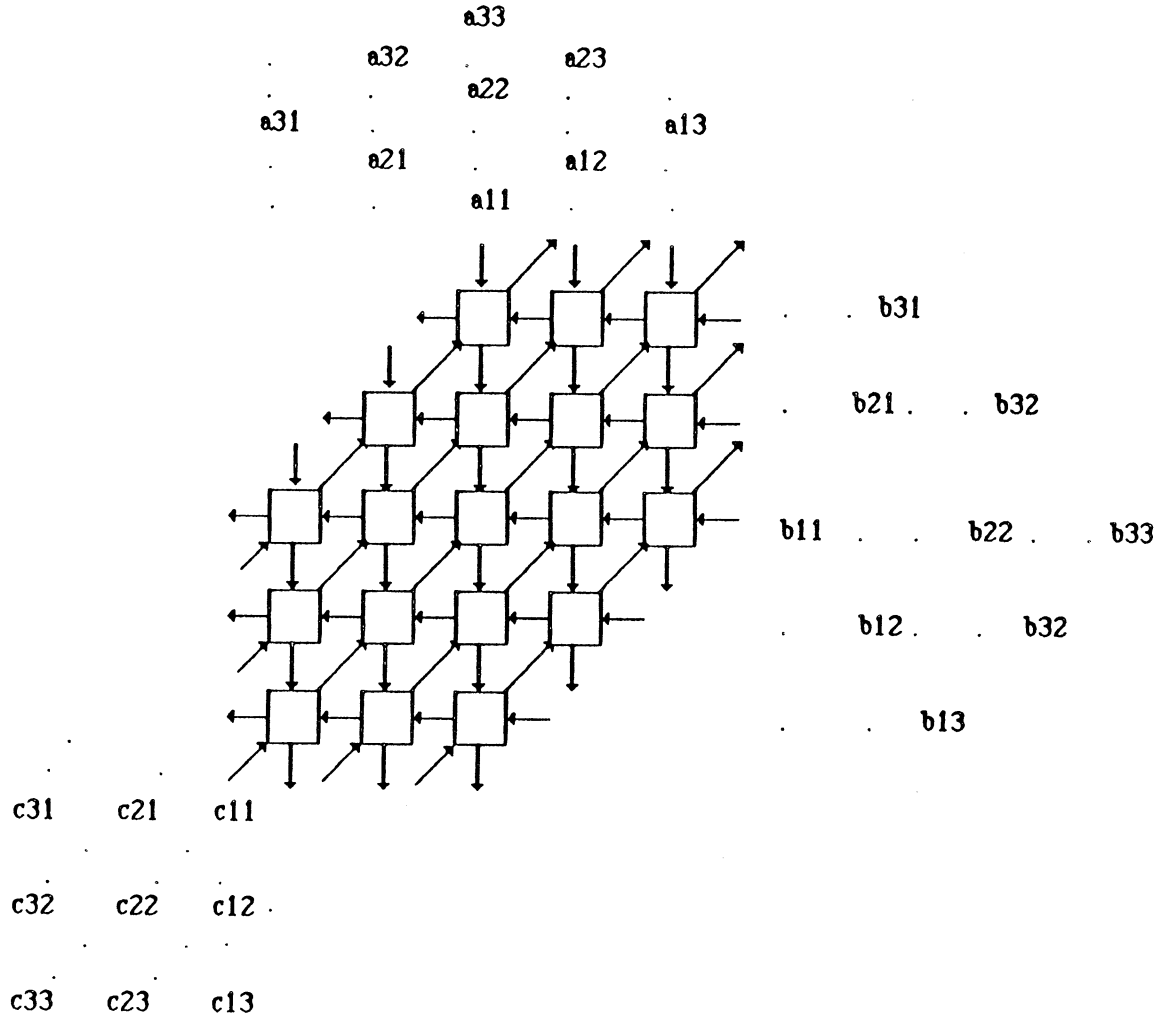
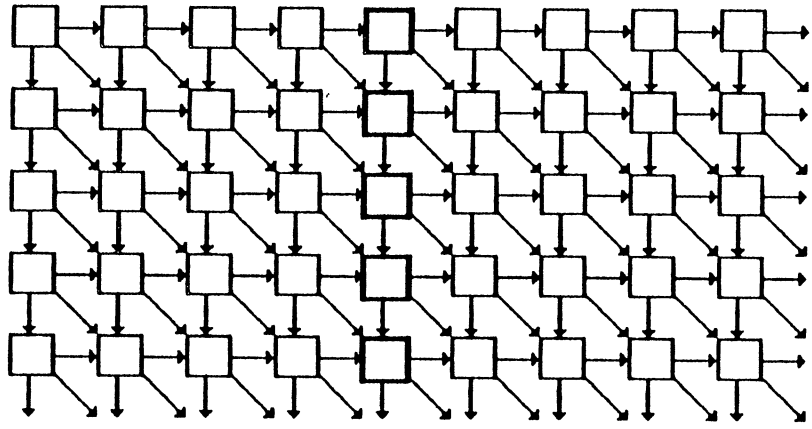


Figure 3.4

En revanche, lorsque des lectures multiples sont autorisées on obtient le réseau de la figure 3.5 qui calcule le produit de deux matrices carrées d'ordre n, en temps T=2n, sur un réseau hexagonal d'ordre $2n^2 + O(n)$.



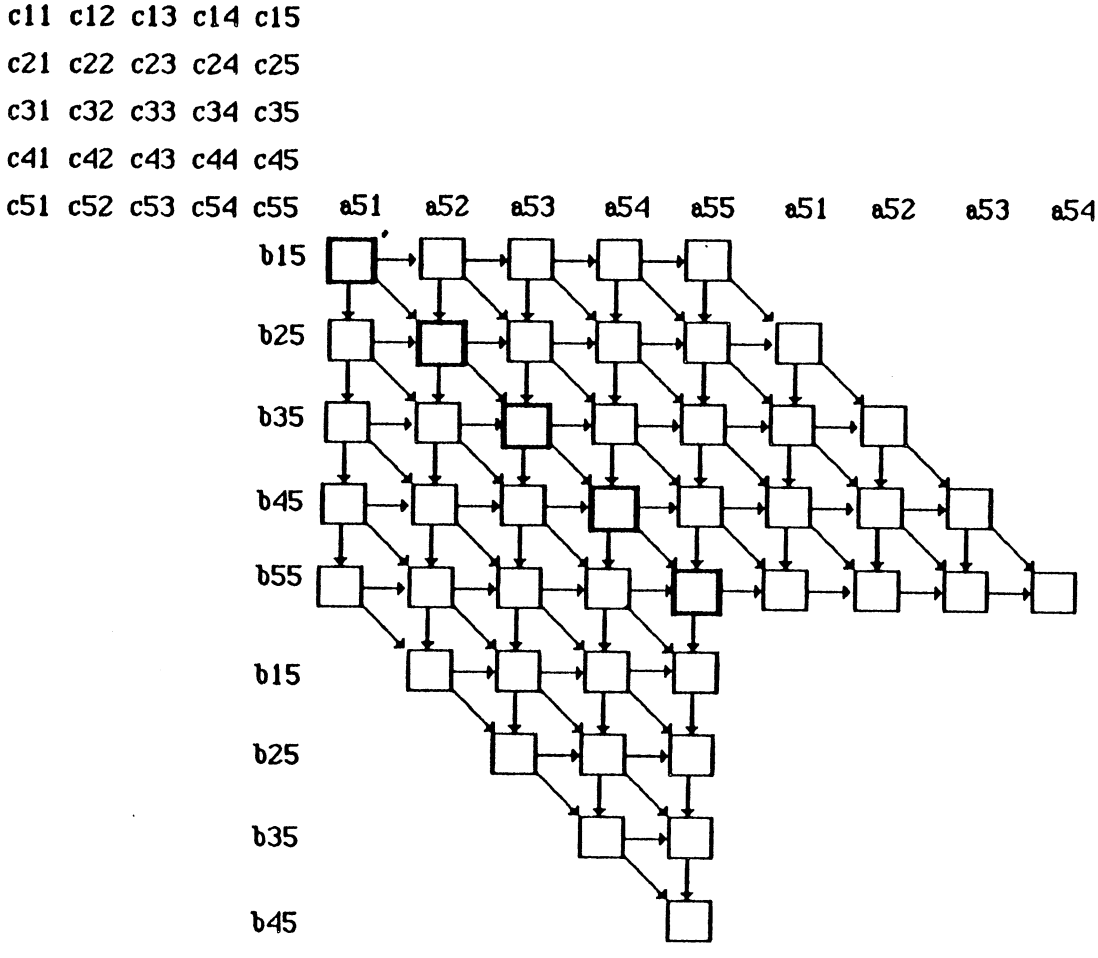


Figure 3.5

Il convient de noter que toutes les solutions optimales avec lecture unique de données peuvent être obtenues avec les méthodes d'analyse des dépendances [JoG 87]. En revanche, seule l'approche combinatoire introduite par Melkemi et Tchuente [MeT 85, 86, 87] permet de dériver les solutions optimales avec lectures multiples.

3.3. Algorithmes unidirectionnels totalement pipelinés

Nous disons qu'un algorithme est totalement pipeliné lorsque tous les coefficients $a_{i,k}$, $b_{k,j}$ et $c_{i,j}$ circulent pendant le déroulement de l'algorithme. Les cellules de base sont donc combinatoires puisqu'elles n'ont rien à mémoriser et que les données circulent de manière statique. La différence avec les algorithmes étudiés au chapitre 2 réside essentiellement dans la présence des signaux de contrôle.

Comme au chapitre 2, nous considérons, pour des raisons de commodités, des réseaux synchrones dans lesquels Buf_a, Buf_b et Buf_c sont de longueurs respectives x-1, 1 et 0.

D'après la formulation combinatoire nous pouvons procéder en deux étapes.

Etape 1: Immersion du graphe de dépendance dans Z^2.

Dans cette étape le graphe de dépendance associé au produit C = A.B, et que nous avons présenté au chapitre 2, est immergé dans Z^2 avec les trois contraintes suivantes:

- Pour tout couple (i,j), les P(i,j,k), $1 \leq k \leq n$, doivent être affectés à une droite horizontale.

- Si P(i,j,k) et P(i',j',k') appartiennent à une droite de pente +1, alors j=j' et k=k'

- Si P(i,j,k) et P(i',j',k') appartiennent à une droite de pente x-1, alors i=i' et k=k'.

Etape 2:

Une fois l'immersion de l'étape 1 effectuée, le diagramme de calcul peut contenir des points de calcul indésirables qu'il faut invalider à l'aide des signaux de contrôle. Rappelons qu'un point de calcul P est par définition de la forme

$$P = D_a^{r,s} \cap D_b^{h,k} \cap D_c^{i,j}.$$

Il est donc le le point d'intersection de:

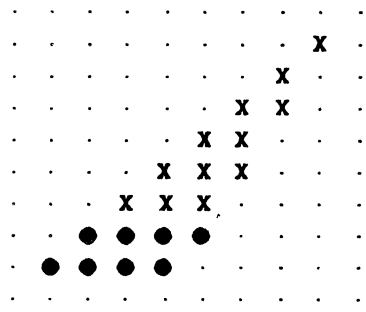
- la droite $D_a^{r,s}$ de pente x-1, contenant la trajectoire de $a_{r,s}$
- la droite $D_b^{h,k}$ de pente 1, contenant la trajectoire de $b_{h,k}$
- la droite $D_c^{i,j}$ de pente 0, contenant la trajectoire de $c_{i,j}$

La présence de ces points indésirables vient du fait que l'immersion s'effectue en respectant des contraintes moins sévères qu'au chapitre précédent.

Nous allons juste traiter un exemple, le cas général pouvant être déduit aisément.

Exemple 3.1: n=4 et x-1 = 2.

Les deux premiers segments du pavé (1) se placent sans contraintes. Il font apparaitre des points interdits représentés par des croix sur la figure 3.6.



x : point interdit pour les premiers segments horizontaux.

Figure 3.6

Les deux derniers segments du pavé (1) sont donc placés en sautant six lignes horizontales (cf. figure 3.7)

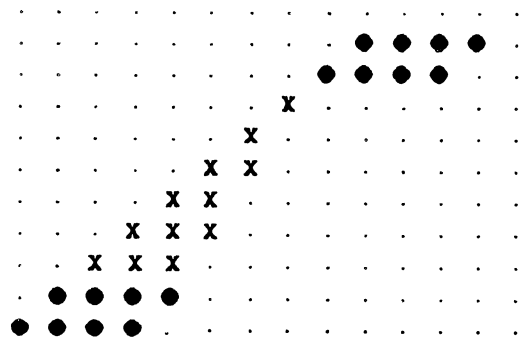


Figure 3.7. pavé (1).

Cas 1: pas de signaux de contrôle

Si on était dans un contexte sans signaux de contrôle alors le pavé (2) s'obtiendrait à partir du pavé (1) par translation de vecteur 8(1,2) (cf. figure 3.8).

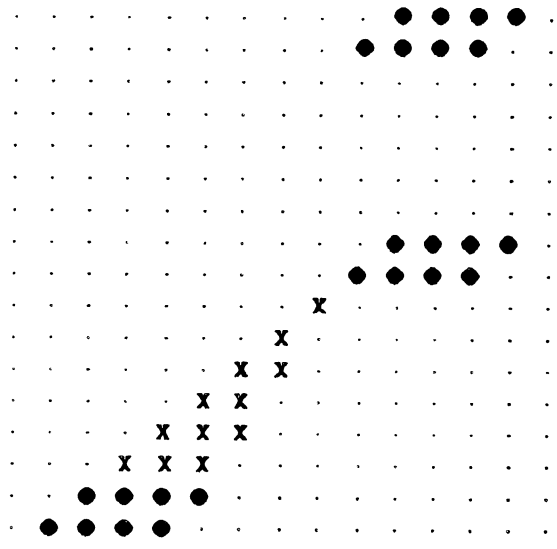


Figure 3.8 : pavé(1) et la moitié du pavé (2), cas 1.

Pour cet exemple:

- n = 4 , x - 1 = 2
- données lues une seule fois
- réseau composé de cellules compbinatoires

l'algorithme s'exécute sur un réseau de taille S = 34 et en temps t = 74.

Cas 2: avec signaux de contôle, données lues une seule fois

Lorsqu'on a la possibilité d'utiliser des signaux de contrôle, le placement du pavé (2) est soumis uniquement à trois contraintes, dont la première est due au fait que les données sont lues une seule fois:

- Il doit être obtenu à partir du pavé (1) par translation de vecteur h.(1,x-1).
- Il ne doit pas chevaucher la bande de pente +1 et de largeur n, contenant le pavé (1).

Cette contrainte vient du fait que le pavé (2) utilise la deuxième colonne de B, alors que cette bande est réservée à la première colonne de B.

- Aucune droite horizontale ne doit contenir des éléments du pavé (1) et du pavé (2).

Il convient de noter que contrairement à ce qu'on avait pour les algorithmes sans signaux de contrôle, il y a une contrainte en moins:

- On peut avoir des lignes honrizontales contenant à la fois des éléments du pavé (2) et des points interdits par le pavé (1).

La figure 3.9 présente une solution dans laquelle il faut définir des signaux de contrôle permettant d'invalider tous les points de calcul indésirables. Comme les $c_{i,j}$ se calculent de gauche à droite, on voit que ces signaux de contrôle doivent permettre aux $c_{i,j}$ d'effectuer des accumulations uniquement au moment de la traversée des segments horizontaux de longueurs n contenues dans les pavés. Il suffit donc d'adjoindre aux $b_{k,j}$ des signaux ayant trois valeurs possibles d, f et nil:

- La valeur ' d ', signale le debut des accumulations

- La valeur ' f ', signale la fin des accumulations

Cette solution est facilement généralisable pour n et x quelconques.

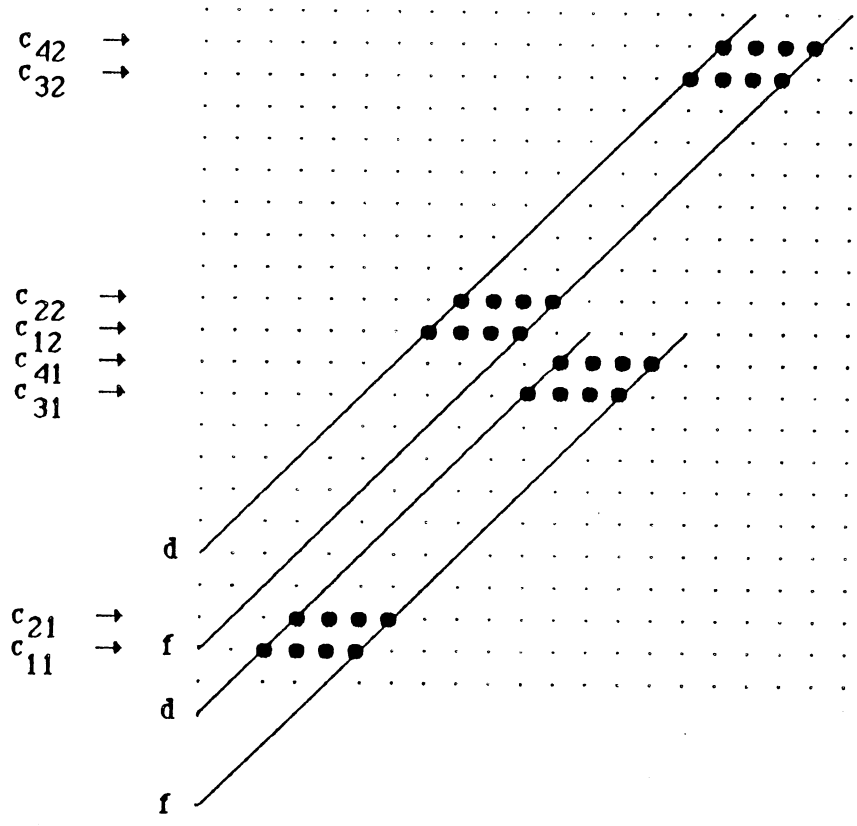


Figure 3.9 : pavé (1) et pavé (2) , cas 2.

Pour cet exemple:

- n = 4 , x - 1 = 2
- données lues une seule fois
- réseau composé de cellules programmables

l'algorithme s'exécute sur un réseau de taille S = 28 et en temps T = 56.

Cas 3: avec signaux de contrôle, données lues plusieurs fois

Lorsqu'on a la possibilité d'utiliser des signaux de contrôle, le placement du pavé (2) est soumis uniquement à deux contraintes:

- Il ne doit pas chevaucher la bande de pente +1 et de largeur n, contenant le pavé (1). Cette contrainte vient du fait que le pavé (2) utilise la deuxième colonne de B, alors que cette bande est réservée à la première colonne de B.

- Aucune droite horizontale ne doit contenir des éléments du pavé (1) et du pavé (2).

Ce qui donne la solution représentée dans la figure 3.10 ci-dessous.

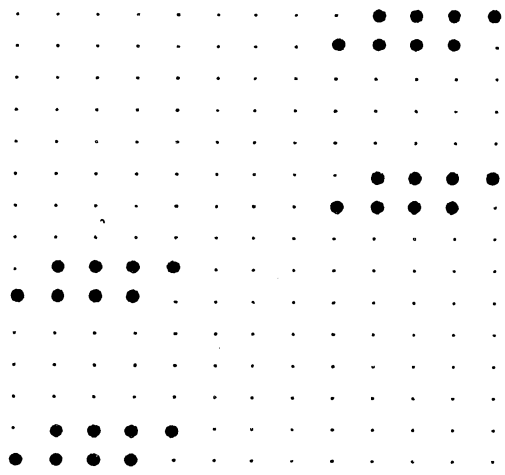


Figure 3.10. pavé (1) et pavé (2), cas 3.

Pour cet exemple:

- n = 4 , x - 1 = 2
- données lues plusieurs fois
- réseau composé de cellules programmables

l'algorithme s'exécute sur un réseau de taille S = 13 et en temps T = 41.

3.4 Algorithmes unidirectionnels partiellement pipelinés

Nous nous intéressons ici au cas où:

- Les $a_{i,k}$ se déplacent de gauche à droite avec Buf_a de taille 1
- Les $b_{k,j}$ se déplacent de gauche à droite avec Buf_b de taille 2
- Les $c_{i,j}$ accumulent les produits $a_{i,k}.b_{k,j}$ de manière statique.

Nous supposons que chaque cellule du réseau est dotée d'une mémoire de taille fixée x. Dans toute la suite nous supposons que x divise n. Le cas général se traite en remplaçant n/x par [n/x], où [z] designe le plus entier superieur ou égal à z. Le produit de deux matrices carrées de taille n requiert au moins n^2/x cellules. Ramakrishnan et Varman [RaV 85] ont exhibé une solution qui requiert un réseau de taille (n-1)n/x + n, et s'exécute en temps

$T = (n-1)(1+n/x+y) + n$,

où $n \leq y \leq n^2$ et $y \geq n.\ pgcd(y,n/x)$. L'entier y peut donc être choisi comme étant le plus petit entier superieur à n et premier avec n/x.

L'approche combinatoire consiste ici à immerger le graphe dans Z^2 avec les contraintes suivantes, où nous avons raisonné sur le réseau synchrone avec propagation en cascade.

- Pour tout couple (i,j), les P(i,j,k), $1 \leq k \leq n$, doivent être affectés à une droite verticale, qui ne doit pas contenir plus de n.x points de calcul.

- Si P(i,j,k) et P(i',j',k') appartiennent à une droite horizontale, alors j=j' et k=k'

- Si P(i,j,k) et P(i',j',k') appartiennent à une droite de pente +1 alors i=i' et k=k'.

Nous présentons dans les figures 3.11 et 3.12 deux algorithmes pour les cas particuliers n=5 , x=2, et n = 3 , x = 3. Ils diffèrent par le sens dans lequel on construit les pavés et requierent des réseaux de taille respectivement 15 et 5.

Ces deux stratégies conduisent dans le cas général à des réseaux de tailles $n^2/x+x-1$, qui résolvent le problème en temps $T= n^2(1/x + 1) + n + x - 1$, et qui sont donc meilleurs en general que celui de [RaV 85]. Cette supériorité est confirmée dans le cas particulier n=5, x=2 traité dans [RaV 85] et qui nécessite 17 cellules.

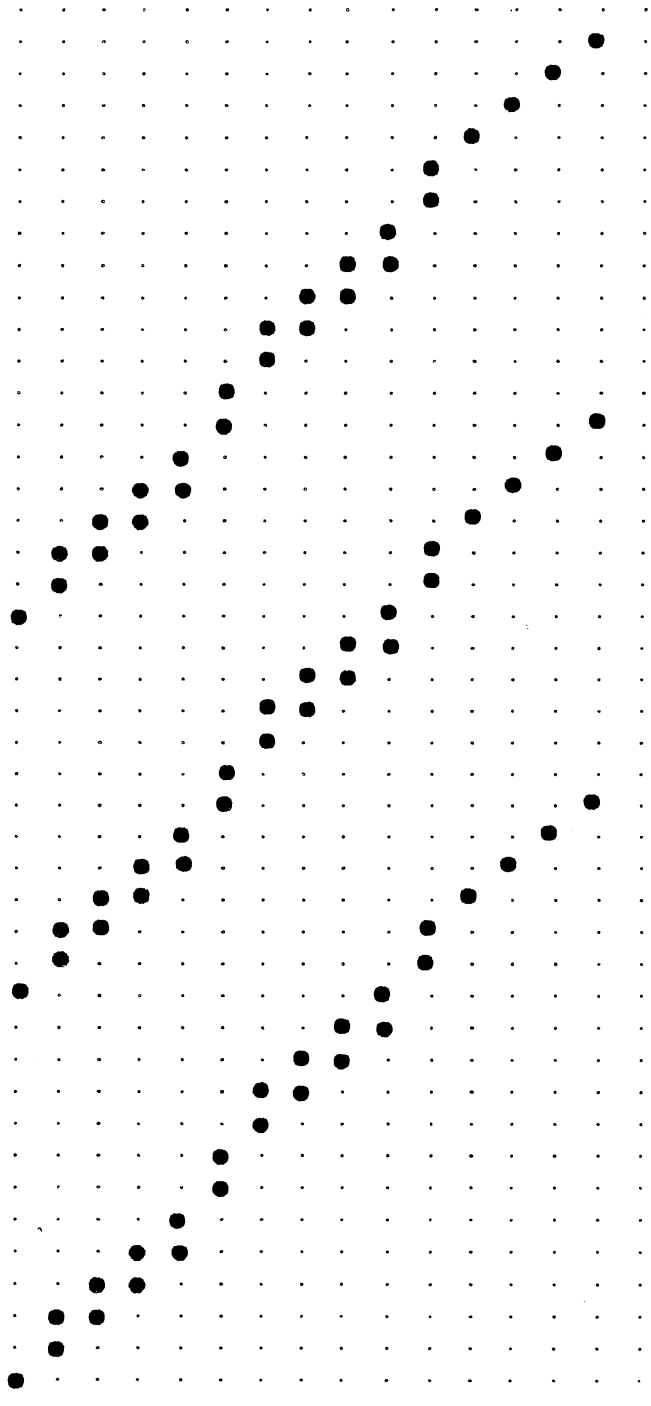


Figure 3.11

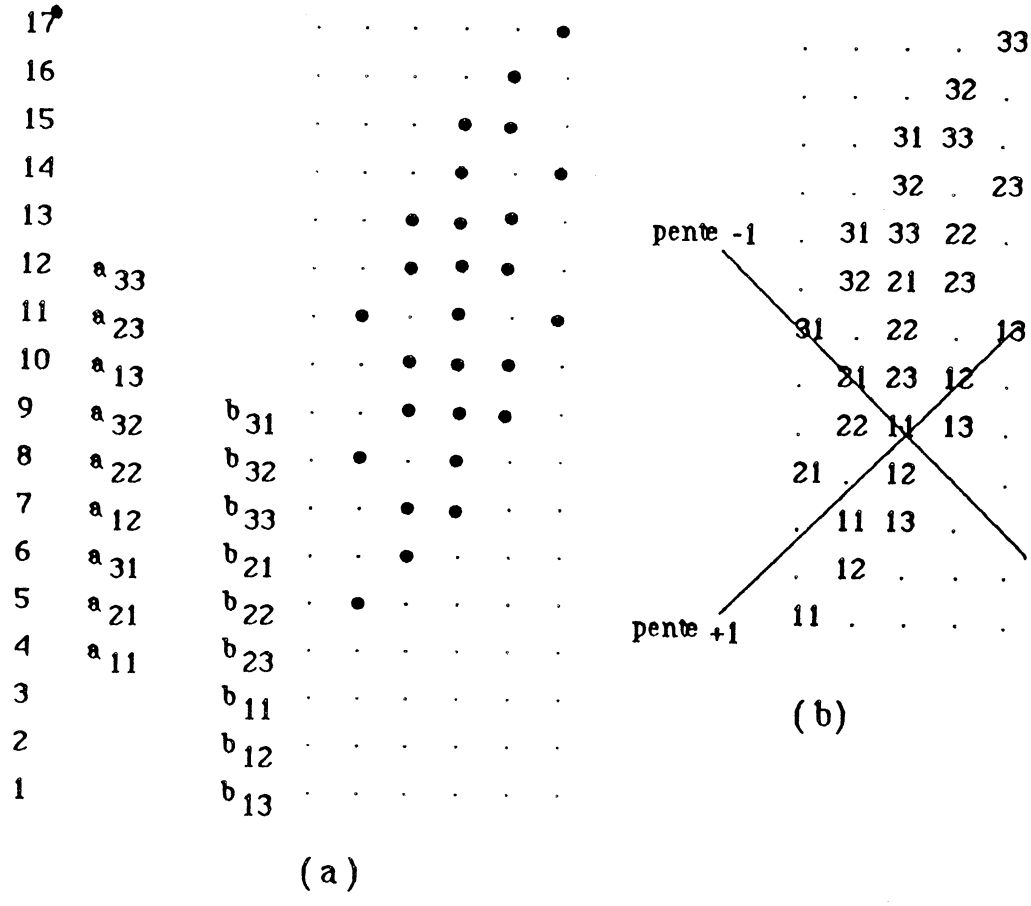


Figure 3.12. (a) diagramme de calcul, n = x = 3
(b) disposition des $b_{i,j}$ dans le diagramme de calcul.

Chaque point de calcul représente par définition une rencontre ($a_{i,k}$, $b_{k,j}$, $c_{i,j}$). Nous avons remplacé chaque point de calcul par le $b_{k,j}$ correspondant dans la figure 12(b). Cette dérnière peut être retrouvée , dans le cas $n = 2p + 1$ et $x = n$, en affectant les $b_{i,j}$ au domaine $D = \{ (x,y) \mid 1 \leq x \leq 2n-1 \leq y \leq n^2+3n-1 \}$ de sorte que:

- Si $b_{i,j}$ et $b_{i',j'}$ appartiennent à une même droite de pente +1 alors $i = i'$
- Si $b_{i,j}$ et $b_{i',j'}$ appartiennent à une même droite de pente -1/p alors $j = j'$
- Si $b_{i,j}$ et $b_{i',j'}$ appartiennent à une même droite de pente +2 alors $i = i'$, $j=j'$.

Noter la similitude entre les deux premières contraintes et la formulation combinatoire, pour le produit matriciel sur les réseaux réctangulaires, concernant les emplacements des $c_{i,j}$ [MeT 85, MeT 88]. Quant à la troisième contrainte, elle traduit simplement le fait que le réseau est linéaire et que Buf_b est de taille 2.

3.5. Algorithmes bididirectionnels

Dans ce paragraphe, nous allons étudier les algorithmes de produit matriciel pour les réseaux linéaires bidirectionnels. Commençons par un cas déja étudié par Varman et Ramackrishnan [RaV 84] et que nous avons nettement amélioré. Ce cas correspons aux hypothéses suivantes:

- Les $a_{i,k}$ circulent de gauche à droite avec Buf_a de taille 2
- Les $b_{k,j}$ circulent de gauche à droite avec Buf_b de taille 1
- Les $c_{i,j}$ circulent de droite à gauche avec Buf_c de taille 1.

Le seul résultat connu pour le produit matriciel sur cette classe de réseaux, est dû à Ramakrishnan et Varman. Ces auteurs ont en effet proposé un algorithme pour le produit de deux matrices denses qui s'exécute en temps $T= 9n^2/2 + O(n)$ sur un réseau de taille $3n^2/2$. Cet algorithme, qui pour n =3 correspond au trois matrices des dates d'entrée ci-dessous, est illustré dans la figure 3.13. La présence des points de calculs indésirables a ammené à introduire des signaux de contrôle dont nous ne donnerons pas les détails. Cette solution correspond en fait à une projection du graphe des dépendances. Elle s'exécute en temps T=51 sur un réseau de taille 13.

$$T_a = \begin{pmatrix} 20 & 24 & 28 \\ 23 & 27 & 31 \\ 26 & 30 & 34 \end{pmatrix} \quad T_b = \begin{pmatrix} 15 & 14 & 13 \\ 21 & 20 & 19 \\ 27 & 26 & 25 \end{pmatrix} \quad T_c = \begin{pmatrix} 16 & 18 & 20 \\ 25 & 27 & 29 \\ 34 & 36 & 38 \end{pmatrix}$$

Nous nous sommes basés sur notre approche pour concevoir un algorithme, pour ce cas, plus performant que celui de Ramackrishnan et Varman. La formulation combinatoire de ce problème consiste à immérger le graphe des dépendances dans Z^2 de sorte que:

- Pour tout couple (i,j), les P(i,j,k), $1 \leq k \leq n$, doivent être affectés à une droite de pente -1.

- Si P(i,j,k) et P(i',j',k') appartiennent à une droite de pente +1, alors j=j' et k=k'

- Si P(i,j,k) et P(i',j',k') appartiennent à une droite de pente 2, alors i=i' et k=k'.

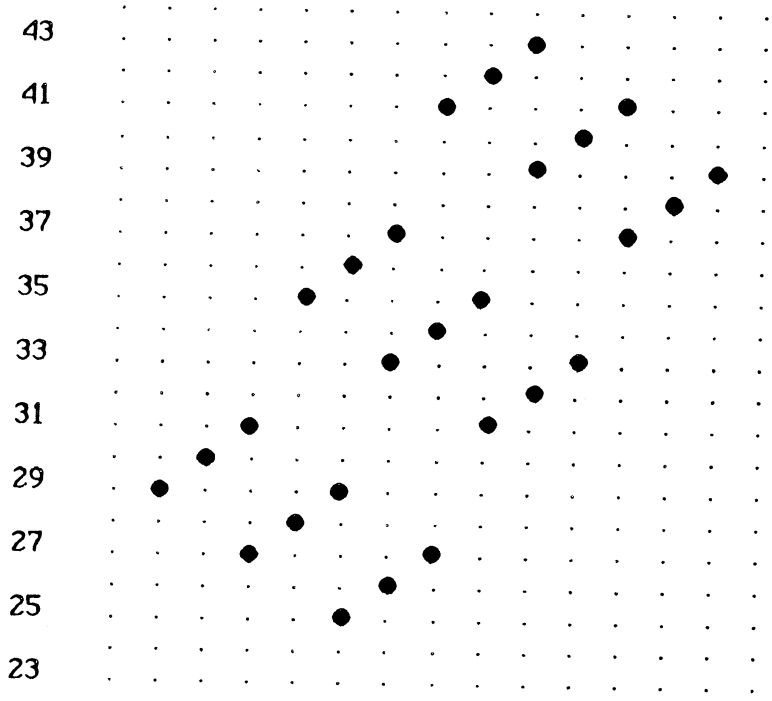


Figure 3.13 diagramme de calcul correspondant à l'algorithme de Varman.

Pour n=3 il suffit d'appliquer la méthode du paragraphe précédent, pour le placement des pavés. La solution obtenue s'exécute en temps T=29 sur un réseau de taille 5, ce qui est beaucoup mieux que la solution de [RaV 84]. Des signaux de contrôle analogues à ceux du paragraphe précédent, permettent d'invalider les points de calcul indésirables (cf. figure 3.14).

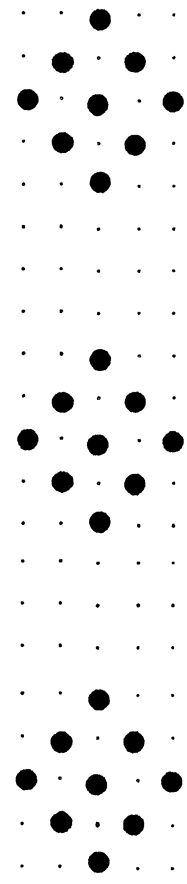


Figure 3.14

Cet algorithme est totalement pipeliné. Sous la condition que les données soient lues une seule fois, cet algorithme se généralise à n quelconque. Ceci peut se faire tout simplement en laissant un écart de 2(n-1) entre deux pavés successifs. On obtient alors un algorithme qui s'exécute sur un réseau de taille $S = 2n - 1$ et en temps $T = 3n^2 + O(n)$.

Dans le cas où on recherche une solution partiellement pipelinée, où les $c_{i,j}$ se calculent de manière statique, les $a_{i,k}$ et $b_{k,j}$ se déplaçant horizontalement dans des sens opposés, avec Buf_a et Buf_b de taille 1, l'approche combinatoire est la même que celle développée par Melkemi et Tchuente en dimension 2. En effet, il suffit d'affecter les couples (i,j), $1 \leq i,j \leq n$ dans Z^2 avec les deux contraintes suivantes:

- Si (i,j) et (h,k) appartiennent à une droite de pente +1 alors i=h
- Si (i,j) et (h,k) appartiennent à une droite de pente -1 alors j=k

On obtient alors le flot de données correspondant à une accumulation pour chacun des $c_{i,j}$, et il suffit de répéter ce processus (n-1) fois. Les figures 3.15 et 3.16 donnent deux solutions déduites de cette formulation et qui correspondent aux flots des données des algorithmes proposés dans [KuL 80] et [MeT 88] en dimension 2. La généralisation à n quelconque de ces deux algorithmes donne des algorithmes qui s'exécutent respectivement sur des réseaux de tailles $S = n$ et $S = 2n - 1$, et en temps $T = 2n^2 + O(n)$.

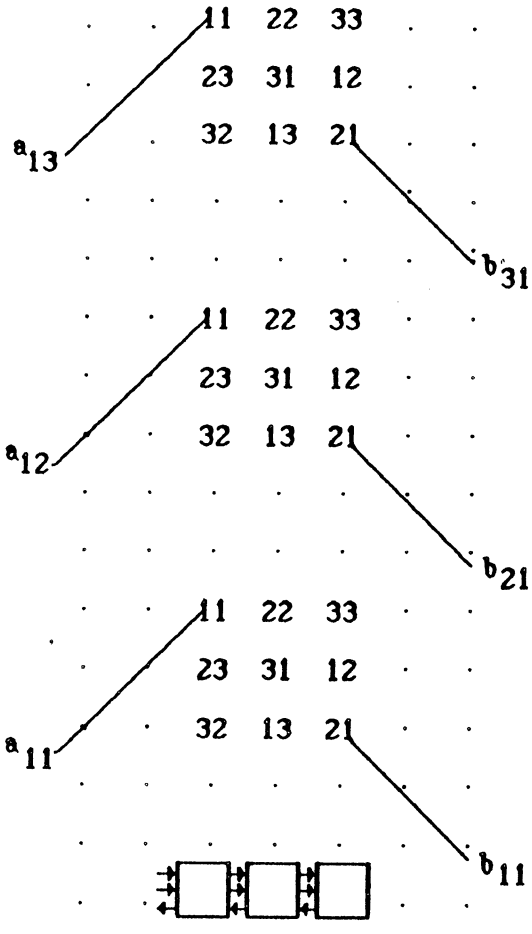


Figure 3.15

flots des données correspondant à l'algorithme de [MeT 88]

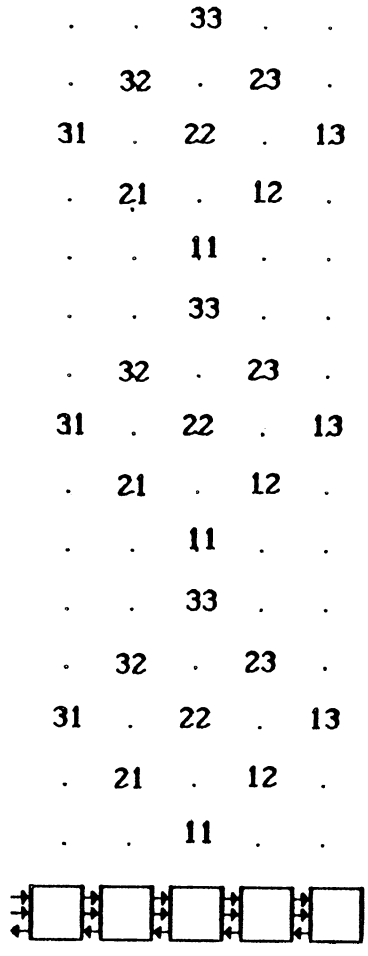


Figure 3.16

flots des données correspondant à l'algorithme de [KuL 80].

Chapitre 4

ELIMINATION DE GAUSS SUR UN RESEAU UNIDIRECTIONNEL

4.1. Introduction

Soit A une matrice carrée dense d'ordre n et b un vecteur à n composantes réelles. La quasi-totalité des méthodes directes pour la résolution de systèmes linéaires

(1) $Ax = b$

procédent en deux étapes.

- D'abord, on transforme (1) en un système triangulaire équivalent

(2) $Tx = b'$.

Cette phase dite de triangularisation est réalisée en combinant successivement les lignes entre elles pour éliminer les éléments sous diagonaux. Les matrices carrées d'ordre deux utilisées pour ces combinaisons ont des formes diverses selon qu'on utilise des pivots, ou des matrices de rotations. L'élimination de Gauss que nous analysons ici, utilise les matrices 2x2 ci-dessous, pour combiner deux lignes.

$$\begin{pmatrix} 0 & \dots & 0 & a_{k,k} & \dots & a_{k,n} \\ 0 & \dots & 0 & 0 & \alpha_{i,k+1} & \dots & \alpha_{i,n} \end{pmatrix} := \begin{pmatrix} 1 & 0 \\ -\frac{a_{i,k}}{a_{k,k}} & 1 \end{pmatrix} \begin{pmatrix} 0 & \dots & 0 & a_{k,k} & \dots & a_{k,n} \\ 0 & \dots & 0 & a_{i,k} & \dots & a_{i,n} \end{pmatrix}$$

- Ensuite, on résoud le système triangulaire (2).

Nous nous intéressons ici à l'implémentation systolique de la phase de triangularisation. Ce problème a été largement traité sur des réseaux de dimension deux. Gentleman et Kung [GeK 81], Bojanszyk, Brent et Kung [BBK 84], et Delosme [Del 82] ont proposé indépendamment des solutions qui s'exécutent en temps T = 3n sur des réseaux de taille $n^2/2$. Cette performance en temps est optimale. En revanche, Cosnard, Daoudi, Muller et Robert [CDM 86], ont réduit la surface à $3n^2/8$. Plus réçemment Bermond, Peyrat, Sakho et Tchuente [BPS 88] ont montré que la borne inférieure en surface est $n^2/4$, et ont proposé un réseau de taille $3n^2/10$ pour l'élimination de Gauss. Louka et Tchuente [LoT 88] ont enfin montré qu'on peut paralléliser en temps optimal le schéma d'élimination de Givens proposé par Sameh et Kuck [Sam 82] sur un réseau de taille $5n^2/18$.

Nous allons proposer ici un réseau unidirectionnel et modulaire pour la triangularisation, à l'aide de la méthode d'élimination de Gauss.

4.2. Structure générale du réseau

Nous considérons un réseau unidirectionnel, composé de cellules identiques, et ayant un nombre d'entrées/sorties constant c.à.d indépendant de la taille de la matrice. Chaque cellule est dotée d'une mémoire adressable et la nature des opérations qu'elle effectue, à chaque instant, est déterminée par des signaux de contrôle qui circulent d'une cellule à l'autre de manière systolique.

Dans la suite nous supposons que la mémoire locale de chaque cellule est de taille x-1. Nous éxhibons un algorithme qui s'exécute sur un réseau de taille

$$S = n^2/x + O(1)$$

et en temps

$$T = n^2(1+1/x) + O(n),$$

Comme le montre la figure 4.1 ci-dessous, le réseau est décomposé en n-1 tranches qui traitent chacune une colonne de la matrice A. La matrice A est lue colonne par colonne, et le premier élément de chaque colonne est repéré par un signal de contrôle.

Les cellules de chaque tranche doivent donc, en plus de l'élimination effectuée sur une colonne, effectuer le décalage des signaux qui repèrent les débuts des colonnes.

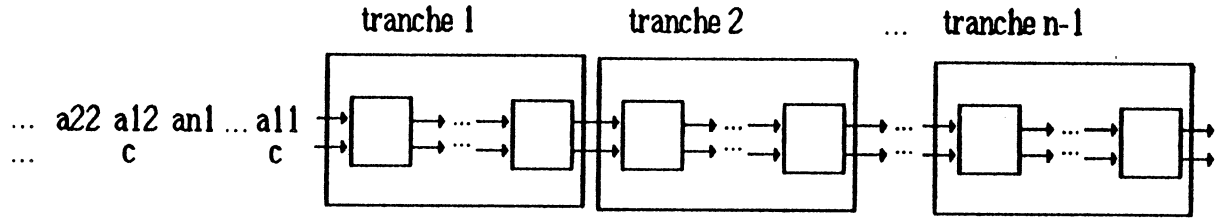


Figure 4.1 Structure générale du réseau

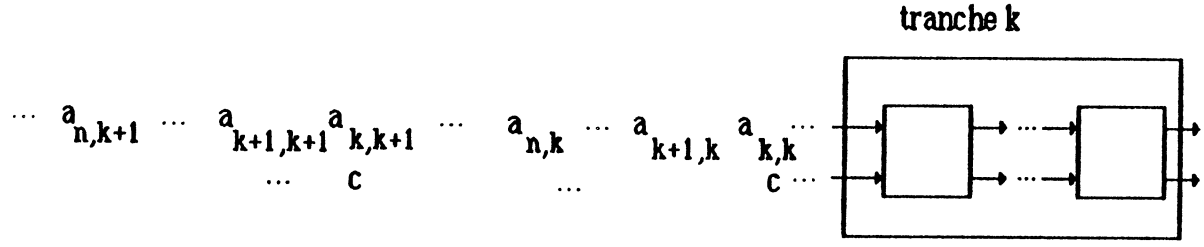


Figure 4.2. Fonctionnement d'une tranche.

Il convient de remarquer qu'à la fin de l'élimination la matrice triangulaire des coefficients utilisés pour l'élimination est stockée dans le réseau. Il faut donc, au moins $n^2/x + O(n)$ cellules.

4.3. Solution sur réseau de taille n-1.

Dans ce cas, chaque tranche est réduite à une cellule où chacune a une mémoire de taille n-1. Le premier signal de début de colonne reçu par une cellule est absorbé par elle, tandis que les autres sont transmis.

- Ce premier signal provoque la génération des coefficients $-a_{k,1}/a_{1,1} = p_{k,1}$, et leur stockage dans la cellule.
- Les signaux de contrôle suivants provoquent d'abord le stockage de l'élément correspondant, $a_{1,k}$, puis le calcul successif des nouvelles valeurs de $a_{2,k}$, $a_{3,k}$, ..., $a_{n,k}$ en fonction de $a_{1,k}$ et de $p_{k,1}$.

Il est clair que cet algorithme exige n^2+n étapes de calcul. En outre, chaque élément de la ligne k a sa valeur définitive calculée dans la cellule numéro k. Comme les dates auxquelles les $a_{i,k}$ sont calculés correspondent au tableau ci-dessous, on vérifie aisement qu'un couloir de largeur 1 permet de faire sortir les résultats en parallèle avec les calculs.

temps \ cellules	1	2	3	4
1	a_{11}			
2		a_{11}		
3			a_{11}	
4				a_{11}
5	a_{12}			
6		a_{12}		
7		a_{22}	a_{12}	
8			a_{22}	a_{12}
9	a_{13}			a_{22}
10		a_{13}		

temps \ cellules	1	2	3	4
11		a_{23}	a_{13}	
12			a_{23}	a_{13}
13	a_{14}		a_{33}	a_{23}
14		a_{14}		a_{33}
15		a_{24}	a_{14}	
16			a_{24}	a_{14}
17			a_{34}	a_{24}
18				a_{34}
19				

Table 4.3. Tableau représentant les dates de production des résultats, n=4.

4.4. Solution modulaire

Dans ce paragraphe nous supposons que la structure de base de la cellule est fixe, indépendante de la taille de la matrice. Plus précisémmant, nous supposons que la taille de la mémoire permet de stocker x coefficients $p_{i,k}$. Le seul problème ici est de montrer qu'une tranche peut être réalisée en mettant bout à bout n/x cellules de ce type. Pour ce faire on procéde comme suit:

Toute cellule d'une tranche se comporte comme dans le pragraphe précédent avec la seule différence que dans chaque phase elle effectue les x premiers traitements, et transmet ensuite systématiquement à sa droite toute les informations nécessaires aux traitements suivants.

Terminons cette présentation en précisant comment s'effectue le décalage des signaux de contrôle. Il faut bien remarquer qu'une tranche de longeur n/x doit effectuer un seul décalage. Il faut donc seule la première cellule d'une tranche soit autorisée à retarder d'un top les signaux de contrôle afin de réaliser les décalages. La première cellule d'une tranche est caractérisée par le fait qu'elle reçoit un signal de contrôle c sans avoir généré aucun $p_{i,k}$.

La figure 4.4 simule la première tranche du réseau dans le cas n=6 et x=2. La tranche 1 est alors composée de trois cellules et l'ordre dans lequel elle reçoit ses données est donné par la figure 4.3.

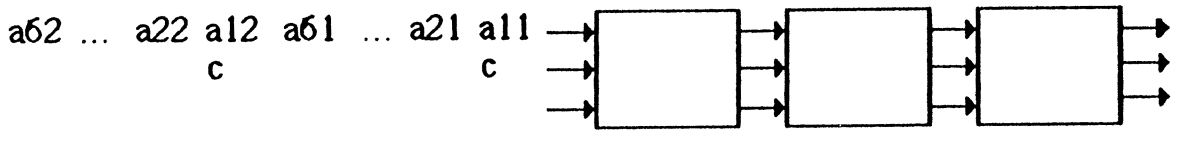
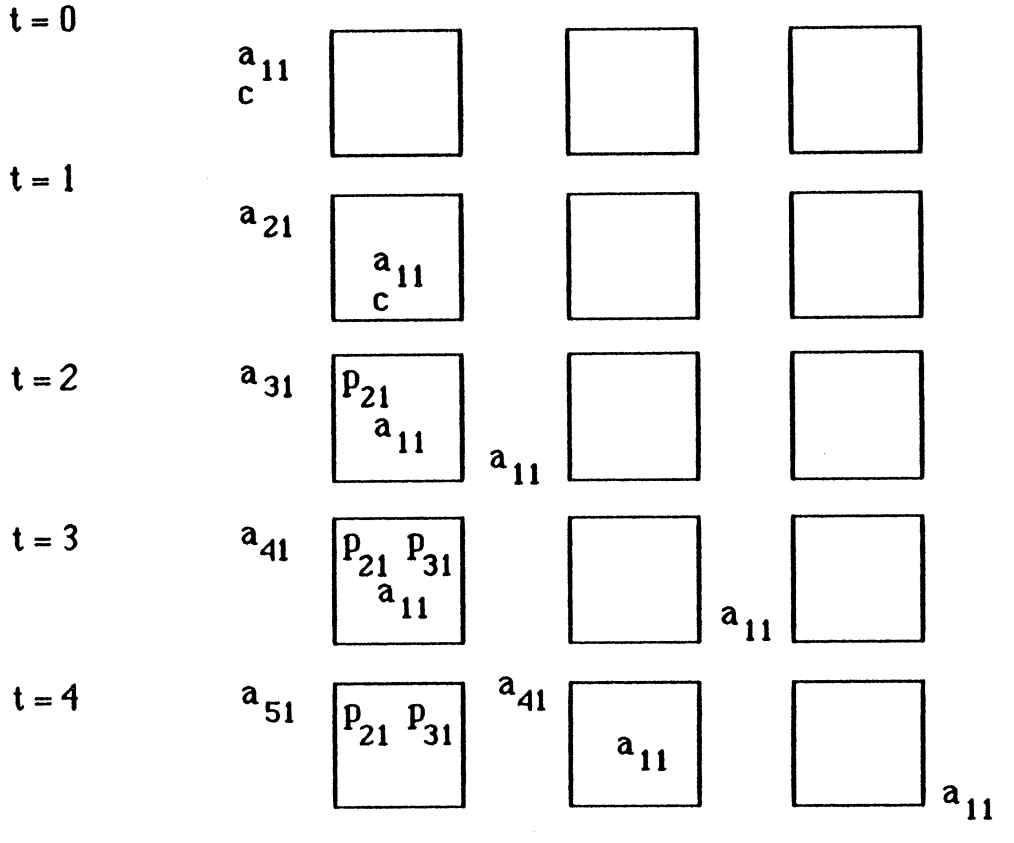


Figure 4.3 : flot de données de la tranche 1.



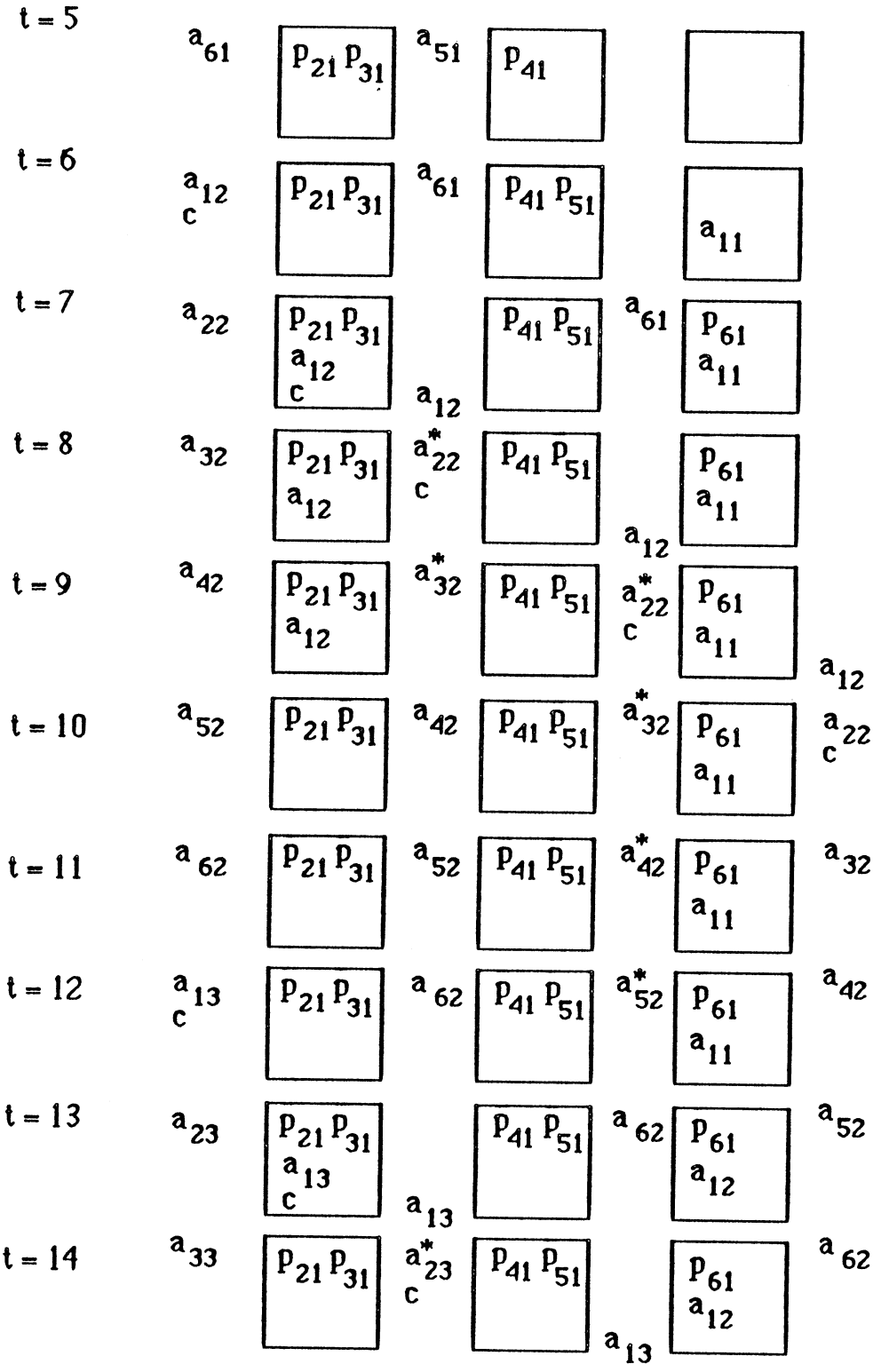


Figure 4.4. simulation de la première tranche du réseau
n = 6 et x = 2

Résumons briévement le fonctionnement de la cellule de base.

- Si elle reçoit un signal de contrôle c alors qu'elle n'a encore généré aucun coefficient $p_{k,j}$ alors c'est la première cellule d'une tranche. Dans ce cas, elle stocke l'élément $a_{k,k}$ correspondant dans un registre interne . A partir de cet instant, elle génére x-1 coefficient $p_{k,j}$, $k+1 \leq j \leq k+x-1$, si durant ces instant elle ne reçoit pas d'autre signal de contrôle c.

L'instant qui suit la génération de $p_{k,k+x}$, elle envoi a à sa cellule voisine et transmet inchangées les données $a_{i,j}$ qu'elle reçoit.

- Lorsqu'elle reçoit le signal de contrôle c alors qu'elle a déja généré des coefficients $p_{k,j}$, elle stocke le signal c dans un registre interne (pour le retarder d'un top) et stocke aussi l'élément $a_{i,j}$ correspondant ($a_{k,j}$ est dans ce cas l'élément pivot). L'instant d'après, elle envoie le contrôle c avec l'élément qu'elle lit et qu'elle modifie (cet élément est $a_{k+1,k+1}$). Durant les x-2 tops suivant, elle applique les $p_{k,j}$ aux éléments qu'elle reçoit. L'instant d'après, elle envoie l'élément $a_{k,j}$ qui se trouve dans son registre interne. Les instants d'après , elle transmet à sa droite les éléments $a_{i,j}$ inchangés.

Partie 2

SIMULATION ET VALIDATION DES ALGORITHMES SYSTOLIQUES

L'étude qui fait l'objet de cette partie est celle de la simulation et de la validation des algorithmes systoliques. Ce problème s'impose pour les réseaux qui ont été conçus de manière heuristique (ceux obtenus à l'aide d'une méthodologie sont valides par construction).

Comme l'a fait remarquer H.T. Kung [Kun 82], dans son célèbre papier "why systolic architectures?", ce problème nécessite des formalismes stricts, qui permettent la description des architectures systoliques, de leur fonctionnement et des algorithmes qu'elles implémentent.

Parmi les formalismes visant ces objectifs, celui de R.Melhem et W. Reinboldt [MeR 84 , Mel 85], permet de décrire de manière fonctionnelle les algorithmes systoliques. L'idée est de représenter les données qui passent dans le même canal, par une suite et d'écrire le programme des cellules à l'aide d'opérateurs algébriques sur les suites. Ainsi, le fonctionnement du réseau systolique se résume en un système d'équations. Pour résoudre de telles équations, R. Melhem a développé un système SCE (pour Sytem of Causal Equations), [Mel 83, Mel 85].

Cependant, la classe des réseaux qu'il couvre est très restreinte. Nous nous sommes inspirés de ce logiciel pour écrire un simulateur (numérique) simple et général, baptisé SISYC.
Nous exposons dans le chapitre 5, ses principes avec quelques exemples simples d'utilisation.

Notre but est d'écrire un système qui puisse vérifier si un réseau systolique implémente correctement un algorithme séquentiel donné. SISYC, étant un simulateur numérique, ne peut effectuer cette tache. Le doter d'outils pour manipuler des variables symboliques n'est pas une solution satisfaisante. En effet, d'une part les résultats qu'il fournirait seraient inexploitables (imaginer ce qui se passe lorsqu'il s'agit d'un réseau pour la décomposition LU par exemple), d'autre part il nécessiterait beaucoup de traitements inutiles d'expressions symboliques.

Au vu de ces remarques, nous avons cherché à définir une méthode qui permette de séquentialiser les algorithmes systoliques. Théoriquement, elle fournit l'un des algorithmes séquentiels implémentés par le réseau considéré. Sa mise en pratique nécessite des calculs symboliques des indices. Pour cette raison, nous nous sommes contentés d'écrire un logiciel SISYC2, basé sur cette méthode, qui retourne la trace d'un réseau systolique.

Nous présentons, dans le chapitre 6, la technique de séquentialisation et le logiciel SISYC2 avec quelques applications simples.

Chapitre 5

SIMULATION DES ALGORITHMES SYSTOLIQUES

Nous commençons ce chapitre par la présentation du formalisme qui permet de décrire le fonctionnement des réseaux systoliques , c'est à dire la méthode permettant de passer d'une écriture schématique d'un réseau à une écriture fonctionnelle équivalente. Ce modèle est dû à R.Melhem et W.Reinboldt [MeR 84, Mel 85]. L'idée de base est de représenter les données qui passent dans un même canal par une suite et d'écrire les programmes des cellules à l'aide d'opérateurs sur ces suites.

On se limite, dans ce chapitre, au cas où les données sont des réels et on note d le delai systolique et R_d l'ensemble des suites $(x(t), t \geq 1)$ où $x(t)$ est un réel ou $x(t)=d$.

5.1. Description des réseaux systoliques

Dans ce paragraphe, nous donnons une représentation fonctionnelle des algorithmes systoliques.

Un réseau systolique est représenté par un multigraphe orienté et coloré $G=(V,E)$; où

- V est l'ensemble des sommets qui représentent les cellules du réseau,

et

- E l'ensemble des arcs qui modélisent les canaux de communication.

Un arc sortant de u et entrant dans v, ne peut être identifié par le couple (u,v). En effet, deux arcs ayant la même source et la même destination seraient alors confondus. La fonction de coloration col permet de les identifier.

La fonction col associe à chaque arc de E, une couleur i appartenant à un ensemble fini { 1, ..., p }. En pratique, p désigne le nombre de signaux qui circulent dans le réseau.

Elle est définie de manière à ce que tous les arcs entrant dans un sommet u , soient de couleurs différentes, et il en est de même pour les arcs sortants d'un sommet u.

Par conséquent, un arc de couleur i, entrant dans un sommet v est caractérisé par le couple (i,v), et la suite de données qui passe dans cet arc est notée x (i,v).

x (i,v,t) = la donnée qui se trouve dans le canal (i,v) à la date t.

Avec ces notations, on peut écrire le programme d'une cellule en exprimant ses entrées et ses sorties en terme de suites.

Il faut donc des opérateurs qui manipulent les suites globalement. Ces opérateurs peuvent être soit une extension des opérateurs arithmétiques soit des conditionnels. En fait, le problème majeur de ce modèle est de trouver une base finie d'opérateurs à l'aide de laquelle on peut écrire le programme de n'importe quelle cellule systolique.

Nous présentons dans le paragraphe suivant, les opérateurs implémentés dans SISYC et qui constituent une base.

5.1.1. Opérateurs de base

Nous proposons comme opérateurs, de base les extensions aux suites des opérateurs arithmetiques , les opérateurs de decalage et les conditionnels.

5.1.1.1. Opérateurs arithmétiques

L'extension des opérateurs réels aux suites se fait comme suit:

Soient

- x, y et z trois suites de R_d,
- t un entier positif ,
- op un élement de l'ensemble { + , - , * , / }.

L'extension de l'opérateur op à R_d, notée aussi op, est définie sur chaque composante des suites.

Si

$$z = x \ op \ y$$

alors

$$z(t) = \begin{cases} x(t) \ op \ y(t) & \text{si } x(t) \neq d \text{ et } y(t) \neq d \\ d & \text{sinon} \end{cases}$$

Exemple:

$$x + y = x(1) + y(1),\ x(2) + y(2), \ldots$$

Soit a un réel, alors

$$y = a \cdot x$$

est tel que

$$y(t) = \begin{cases} a \cdot x(t) & \text{si } x(t) \neq d \\ d & \text{sinon.} \end{cases}$$

5.1.1.2. Opérateurs de décalage

Soient

- k un entier,
- x et y deux suites de R_d

Alors les opérateurs de décalage O , Z et T sont définis comme suit:

Opérateur O:

$$y = O\{k\}\ x$$

est définie par

$$y(t) = \begin{cases} x(t-k) & \text{si } t > k \\ d & \text{sinon} \end{cases}$$

Opérateur Z:

$$y = Z\{k\}x$$

est définie par

$$y(t) = \begin{cases} x(t-k) & \text{si } t > k \\ 0 & \text{sinon} \end{cases}$$

Opérateur T:

$$y = T\{k\}\ x$$

est définie par

$$y(t) = \begin{cases} x\left((t+k)/(k+1)\right) & \text{si } t=1,\ 2+k,\ 3+2k,\ 4+3k,\ \ldots \\ d & \text{sinon} \end{cases}$$

Exemple:

$$x = a_1 , a_2 , a_3 , a_4 , ...$$

$$O\{2\}\ x = d , d , a_1 , a_2 , a_3 , a_4 , ...$$
$$Z\{1\}\ x = 0 , a_1 , a_2 , a_3 , a_4 , ...$$
$$T\{2\}\ x = a_1 , d , d , a_2 , d , d , a_3 , d , d , a_4 , ...$$

5.1.1.3. Les conditionnels

Les conditionnels permettent la description d'une cellule ayant un fonctionnement qui dépend soit des *valeurs* de ses entrées, soit de signaux de contrôle. Considérons par exemple le comparateur ci-dessous, qui sert de bloc de base dans les réseaux de tri.

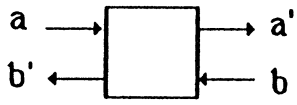


Figure 5.1 : cellule de tri

Son fonctionnement qui dépend des valeurs de ses entrées puisqu'à chaque instant t, elle exécute le programme suivant :

```
SI ( a = d  OU b = d ) {  a' = a ;
                         b' = b ;  }
SI  ( a ≠ d  ET  b ≠ d ) {  a' = max ( a , b ) ;
                            b' = min ( a , b ) ;  }
```

Les conditions sur a et b sont causales en ce sens que a'(t) et b'(t) ne dépendent que des conditions sur a(t') et b(t') avec t' < t. Remarquons que les conditionnels sont de cette forme pour les réseaux systoliques.

Les conditionnels définies en SISYC sont de la forme

```
SI ( condition ) {
                   suite d'instructions ;
                 };
```

où (condition) est une suites booléenne .

Si op est un élément de l'ensemble { <, <= , = , ≠ } et
c = (a op b)
alors

c (t) = 1 si (a(t) ≠ d , b(t) ≠ d , (a(t) op b(t))) = vrai
ou (op = '=' , a(t) = b(t) = d) = vrai.
0 sinon

Par exemple,

si a = 2.0 , d , 3.0 , d , 1.0 , ...
b = 2.0 , 0.0 , d , d , 3.0 , ...
op est l'opérateur '='

alors

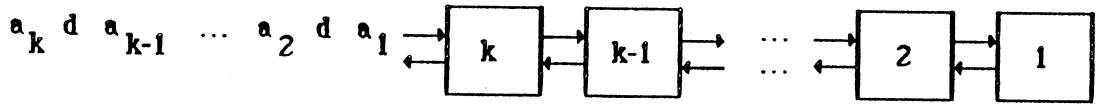
a op b = 1 , 0 , 0 , 1 , 0 , ...

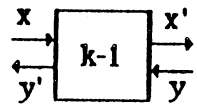
Le conditionnel temporel est autorisé en SISYC. Il peut servir dans le cas de cellules programmables où l'instruction exécutée par une cellule dépend de la date considerée.Naturellement, ce conditionnel n'obéis pas aux lois du systolique, néamoins il peut être utile pour la vérification des réseaux, ou pour remplacer des contrôles externes encombrants. D'autre part, ce test sur le temps est l'unique moyen , en SISYC, pour isoler un élement d'une suite.

Il est défini tout simplement par

```
SI ( condition sur le temps ) {
                                suite d'instructions ;
                               } ;
```

Il est temps de donner un exemple complet du modèle! Il s'agit de trier la suite $(a_t, t=1 \ldots k)$ dans l'ordre croissant. L'algorithme systolique proposé par Kung est le suivant:





$$x' = \begin{cases} \max(x,y) & \text{si } x \neq d \text{ et } y \neq d \\ x & \text{sinon} \end{cases} \qquad y' = \begin{cases} \min(x,y) & \text{si } x \neq d \text{ et } y \neq d \\ y & \text{sinon} \end{cases}$$

Figure 5.2. réseau systolique pour le tri et cellule de base.

Le graphe du réseau est donné par la figure ci-dessous:

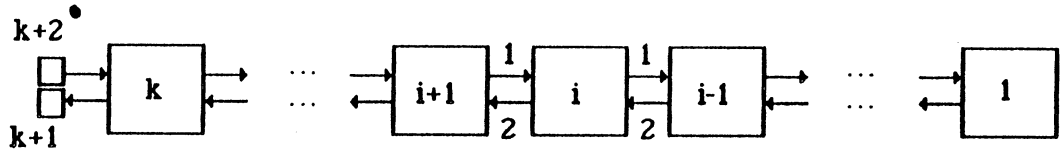


Figure 5.3: graphe du réseau de tri

- L'ensemble des cellules du réseau est { 1, ..., k },
- La suite entée du réseau est
  $x_1\{k\} = T\{1\}\, a$;
  où $a(t) = a_t$,
- La suite sortie du réseau est $y\{k+1\}$.
- Deux couleurs pour les canaux, (p = 2)

Le programme P_i exécuté par le noeud i et exprimé à l'aide des opérateurs sur les suites est le suivant:

P_i, $2 \leq i \leq k$:

SI ($O\,x_1\{i\} = d$ OU $O\,x_2\{i\} = d$) {
$x_1\{i-1\} = O\,x_1\{i\}$;
$x_2\{i+1\} = O\,x_2\{i\}$;
}

SI ($O\,x\{i\} \neq d$ ET $O\,y\{i\} \neq d$) {
$x_1\{i-1\} = \max(O\,x_1\{i\}, O\,x_2\{i\})$;
$x_2\{i+1\} = \min(O\,x_1\{i\}, O\,x_2\{i\})$;
}

Pour le noeud 1 on a :

P_1 : $x_2\{2\} = O\,x_1\{1\}$.

où l'opérateur O désigne $O\{1\}$.

5.2. SISYC : SImulation of SYstolic Circuits

Dans ce paragraphe, nous présentons un logiciel de simulation des algorithmes systoliques. Ce logiciel, baptisé SISYC, est fondé sur le modèle présenté dans le pragraphe 5.1.

En effet, ce modèle permet la description des réseaux systoliques à l'aide d'équations sur les suites, où l'indice courant d'une suite représente le temps. En pratique, ces suites n'ont qu'un nombre fini d'éléments différents du délai noté d. On fixe un entier (noté MAXT dans tout le reste de cette partie) qui sera la taille maximale des suites . Au delà de MAXT les élements de toutes les suites, utilisées par un même algorithme systolique ont la valeur d. Ces suites peuvent donc être assimilées à des vecteurs.

L'environnement logiciel que nous avons mis en oeuvre est schématisé par la figure 5.4. Tout les modules qui composent le logiciel sont écrits en langage C et tournent sous le système d'exploitation UNIX.

SISYC manipule ces vecteurs globalement (on ne peut accéder à un élément isolé d'un vecteur qu'à l'aide d'un conditionnel sur le temps).

Le programme qui simule un réseau n'est rien d'autre que la description de ce dernier en terme de vecteurs. Ainsi, la notion de synchronisme devient implicite lorsqu'on représente les données par des suites.

Trois modules principales composent SISYC:

1) L'analyseur lexicographique:

Il reconnait les mots du programme et retourne leurs codes . En cas d'erreur, il fournit la ligne où se trouve l'erreur et le mot qui précède cette erreur. Il a été construit à l'aide du générateur lex [LeS 74] qui existe sur le système UNIX .

Lex est un générateur d'analyseur lexicographique. IL prend en entrée un ensemble d'expressions régulières, et retourne un programme capable de reconnaitre les mots décrits par ces expressions.
Par exemple si on donne à lex, l'expression régulière suivante:

[0-9]+ return (entier) ;

alors il génère un programme qui peut reconnaitre les entiers , c'est à dire toute suite de chiffres.

2) L'analyseur syntaxique:

Il prend en entrée la description d'un réseau et produit en sortie la table d'analyse selon la grammaire du simulateur. La grammaire de SISYC correspond à celle du système SCE [Mel 83, Mel 85], à laquelle nous avons ajouté les régles grammaticales des conditionnels et des opérateurs introduits dans le paragraphe 5.1. Ce module fait appel à ana_lex pour avoir les codes des mots du programme.

SISYC

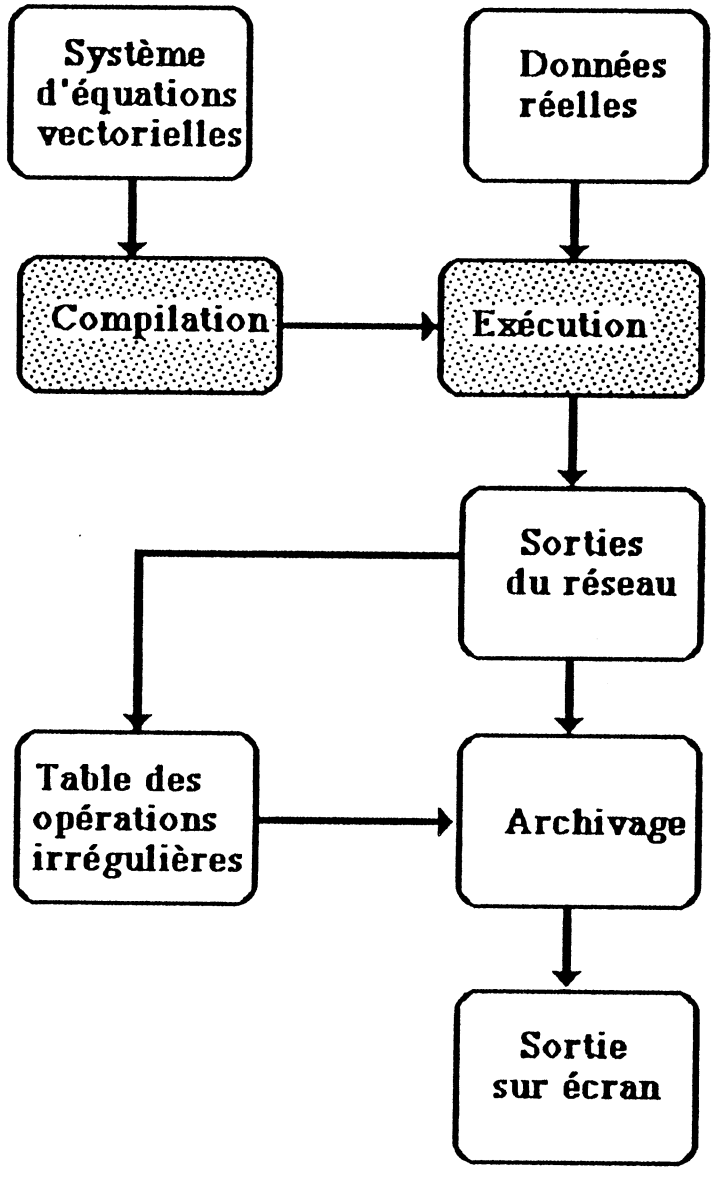
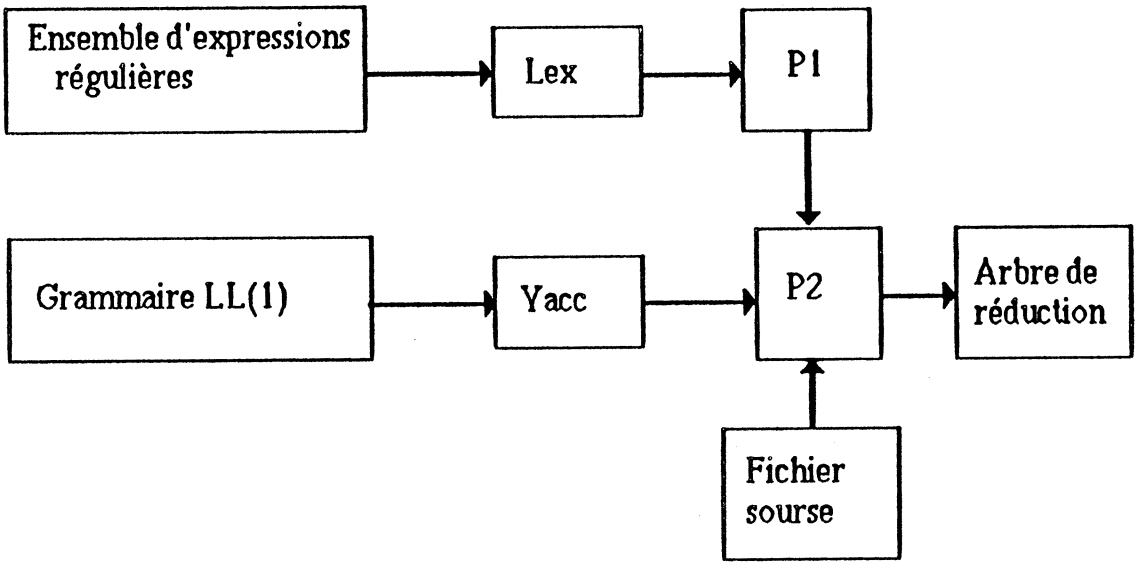


Figure 5.4

Son rôle est de vérifier la syntaxe du programme; en cas d'erreur, il retourne un message d'erreur et s'arrête. Dans le cas d'un programme syntaxiquement correct, l'arbre de réduction du programme, selon la grammaire, est formé dans un fichier nommé nom.sisyc. Ce module a été construit à l'aide du générateur yacc [AhU 77, Joh 74].

Yacc est un générateur d'analyseurs syntaxiques. Il prend en entrée une grammaire LL(1) , et retourne un programme (P). Ce dernier prend comme point de départ, un fichier texte et génére son arbre de réduction (en terme de shifts et de reduces) selon la grammaire. En cas d'erreurs, il renvoit un message et s'arrete. Il fait appel à lex pour la reconnaissance des mots du fichier source.



Il est nécessaire de savoir comment fonctionne le programme fourni par yacc, pour pouvoir le modifier en cas de besoin.

3) Le module d'exécution:

Ce module exécute le fichier nom.sisyc. Il lit le fichier du début à la fin, couple par couple et exécute les routines sémantiques rencontrées. En résumé, il procéde ainsi

- Construit la table des symboles
- Lit les éléments des vecteurs spécifiés dans la partie *ENTRE*
- Calcule itérativement les vecteurs de la partie gauche de chaque équation en fonction de ceux qui apparaissent en partie droite.
- Retourne les vecteurs specifiés dans la partie *SORTIE* .

5.2.1. Structure d'un programme

Un programme SISYC est constitué de quatres parties: Les déclarations, les entrées , le corps du programme et les sorties. Dans ce qui suit, nous essayons d'expliquer chacune de ces parties.

• ***Les déclarations*** : Elles se composent de trois parties principales:

- *La déclaration des paramètres* :
Elle se fait à l'aide du mot clé *PAR* et se termine par un point virgule. Un paramètre en SISYC est l'équivalent d'une constante en Pascal.

Exemple: PAR n1=15, x = 3 ;

Un identificateur est une chaîne de lettres minuscules et de chiffres. Le premier caractère doit être une lettre. Seuls les 6 premiers caractères sont pris en compte.

- *La déclaration des indices* :
Elle se fait à l'aide du mot clé *INDICE* et a la même syntaxe que les paramètres.

Exemple : INDICE i , compt ;

Tout identificateur déclaré comme INDICE est de type entier. Il ne peut être que l'indice courant d'une boucle ou l'un des indices d'un tableau de vecteurs .

- *Déclaration des vecteurs* :
Elle se fait à l'aide du mot *VECT* . On peut déclarer un vecteur ou un tableau de vecteurs . L'identificateur d'un vecteur correspond à la couleur de l'arc dont il est le vecteur de données.

Exemple : VECT x {1:4,-2:7} , y {0:n }, p ;

Pour cet exemple
- x est un tableau [1..4,-2..7] de vecteurs et l'accès aux élements de x se fait par x{i,j} où i et j doivent vérifier $1 \leq i \leq 4$ et $-2 \leq j \leq 7$, autrement un message d'erreur est retourné et l'exécution du programme est interrompue. Le vecteur x {i,j} est de taille [1..MAXT]; où MAXT doit être déclaré dans la partie ENTRE, et on ne peut pas accéder à ses élements.
- y est un tableau a une seule dimension [0. . n] où n doit être déclaré comme paramètre.
- p est un vecteur [1. . MAXT].

Trois vecteurs prédéfinis dans le simulateur, d, z, u, sont respectivement le vecteur nil le vecteur nul et le vecteur unité.
$d(t) = d$, $z(t) = 0$, $u(t) = 1.0$, pour tout t , $1 \leq t \leq$ MAXT.

•***Les entrées :***

Elles se font à l'aide du mot clé *ENTRE* . Dans cette partie on précise la taille des vecteurs (ou le temps de simulation) *MAXT* et les vecteurs qui doivent être lus dans le fichier de données, qui représentent les suites entrées du réseau.

Exemple :

ENTRE (MAXT 18, POUR i=1,4 POUR j=i+1,7 x {i,j }, a {0,1});

Dans ce cas, le fichier de données doit contenir dans l'*ordre* :

x{1,2}, x{1,3}, ..., x{1,7}, x{2,3},..., x{4,7}, a{0,1},

qui doivent être des vecteurs de taille [1...MAXT] = [1...18].
Trois points de suspension ' ... ' dans le fichier de données indiquent que le reste du vecteur est d.

• ***Corps du programme*** :

Il contient les programmes des cellules exprimés en terme d'opérateurs définis dans le paragraphe 1. C'est une suite d'instructions équationnelles et de conditionnels, séparées par des points virgules. Il peut être exprimée à l'aide de POUR FAIRE FIN.

Exemple: POUR i=1,n FAIRE x{i} = z FIN ;

Cette boucle affecte la valeur 0.0 à x{i} (t) , i=1,..., n et t = 1,..., MAXT.
Rappelons que z est la suite (predéfinie) de valeur identiquement nulle.

Une condition peut être:

- Une condition sur les données.

Exemple : SI ((x<y) ET (x≠d)) { x = x+y+z ; };

- Une condition temporelle. Dans ce cas, elle se fait à l'aide de la variable *temp*.

Exemples :

SI (temp= 2) { x = 4.0.u ; } ;
affecte 4.0 à l'élément d'indice 2 du vecteur x.

SI ((3 < temp < i+j) ET (temp ≠ i)) { x = z ; } ;
affecte zéro aux éléments x(k) avec 3 < k < i + j et k ≠ i.

La variable *temp* peut servir lorsqu'une cellule a un programme qui dépend de la date à laquelle il s'exécute. On verra dans le paragraphe 5.4 , un exemple d'utilisation de cette variable.

- Une combinaison de conditions sur les données et sur le temps.

Exemple, SI ((temp = 3) ET ((x = d) OU (y < x))) { y = O x ; } .

• ***Les sorties :***

Elles se font à l'aide du mot *SORTIE*. Dans cette partie, on précise les vecteurs qu'on veut récupérer en sortie du réseau .

Exemple : SORTIE (p , POUR i=1,3 x{i}) ;

signifie que le résultat retourné est p , x{1} , x{2} , x{3} dans cet ordre.

5.2.2. Les opérateurs de SISYC

Comme on l'a signalé, le type de base de SISYC est le vecteur et on ne peut accéder à un élément isolé d'un vecteur qu'à l'aide du conditionnel temporel. Tous les opérateurs définis dans le simulateur ont des effets globaux sur les vecteurs. Les opérateurs prédéfinis sont :

O , Z , T , . , * , / , + , - , MOD , RAC et DIV (cf. paragraphe 5.1).

O , Z , T ont la plus forte priorité ,
suivie de celle de la multiplication par un scalaire '.' et la racine carré RAC ,
suivie du modulo MOD et la division entiere DIV
suivie de celle de *, /

l'addition ,+, et la soustraction ,-, ont la plus faible priorité.

Les crochets ' [ ] ' sont utilisés pour forcer les priorités et O{1}, Z{1}, T{1} peuvent être notés simplement O, Z, T.

5.3. Opérations irrégulières

Chaque vecteur x utilisé dans un programme SISYC définit une sortie d'une cellule ou une entrée du réseau.

- Lorsque x est une entrée du réseau, alors il n'est pas affecté c'est à dire, il n'existe pas d'équation du type

x := expression

- Dans les autres cas, il existe forcément une équation du type x := expression. Cette équation est évaluée une seule fois à chaque instant. Même si x est défini par un conditionnel, il est affecté une seule fois à chaque instant puisque les conditions qui le définissent devraient être disjointes et complémentaires.

Lorsque l'évaluation de l'équation donne

$$x(t) := d$$

on dit que cette opération est irrégulière. Le nombre de telles opérations permet de connaître le taux d'activité du réseau. En effet, si k désigne le nombre de vecteurs utilisés par le réseau (sans compter ses entrées) alors théoriquement, le réseau devrait effectuer k opérations à chaque instant. Si parmi ces k opérations il y a k(t) opérations irrégulières à l'instant t, alors le taux de calcul du réseau à la date t est:

$$p(t) = 1 - k(t) / k$$

Le réseau travaille à son maximun à l'instant t si p(t) = 1, autrement dit s'il n'y a pas d'opérations irrégulières. Le taux moyen de calcul du réseau au cours des MAXT tops est

$$p = (1 / MAXT) \sum_{t=1}^{MAXT} p(t)$$

SISYC calcule les paramétres k(t), p(t) et p, en même temps qu'il simule le réseau.

5.4. Exemples de simulation de réseaux systoliques

Dans ce paragraphe, nous présentons quelques exemples simples de simulation de réseaux systoliques à l'aide de SISYC. La plupart de ces réseaux ont été conçus au laboratoire TIM3.

A travers ces exemples, nous essayons de montrer qu'il n'est pas nécessaire de comprendre comment un réseau fonctionne pour pouvoir le simuler. En effet, il suffit d'exprimer les programmes des cellules à l'aide des opérateurs prédéfinis dans SISYC. Ces opérateurs sont suffisamment généraux pour couvrir la quasi-totalité des réseaux systoliques qui existent dans la littérature (même la pile systolique [GuL 82], les réseaux en structure d'anneaux , ... peuvent être simulés à l'aide de SISYC).

A vrai dire, nous ne connaissons pas de réseau systolique qui ne manipule que des réels et qui ne puisse pas être simulé en SISYC.

5.4.1. Produit de deux matrices sur un réseau rèctangulaire

Il s'agit d'effectuer le produit C = A.B, de deux matrices carrées de tailles n, dont nous avons longuement parlé aux chapitres 2 et 3. L'algorithme systolique conçu pour cette tache est dû à L.Melkemi et M.Tchuente [MeT 85]. Il fonctionne en temps optimal égal à 3n-2.

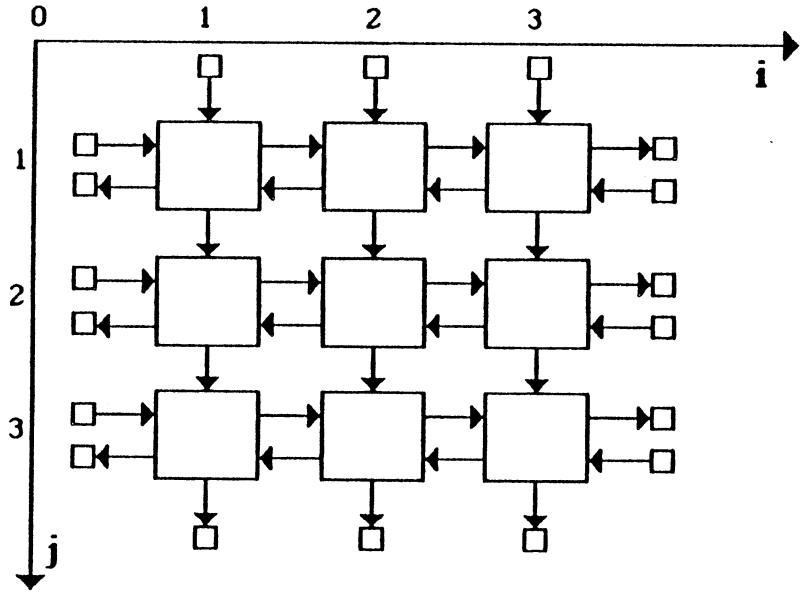
Le graphe du réseau pour n=3 est illustré par la figure 5.5(a) où les petits carrés représentent des cellules fictives d'entrées/sorties. Une cellule est repérée par ses coordonnées dans le repère (i,j).

La figure 5.5(b) illustre le programme d'une cellule exprimé à l'aide des opérateurs vectoriels O, $*$, +. Les entrées du réseau sont les vecteurs

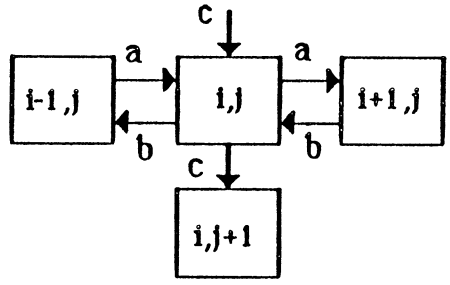
$$a\{1,j\},\ b\{n,j\} \text{ et } c\{j,1\}\ \ 0,\ 1 \le j \le n,$$

avec

$a\{1,1\} = a_{3,1}\ \ a_{2,1}\ \ a_{1,1}\ \ a_{3,1}\ \ a_{2,1}\ \ \ldots$
$a\{1,2\} = d\ \ \ a_{3,2}\ \ a_{2,2}\ \ a_{1,2}\ \ a_{3,2}\ \ a_{2,2}\ \ \ldots$
$a\{1,3\} = d\ \ \ d\ \ \ a_{3,3}\ \ a_{2,3}\ \ a_{1,3}\ \ a_{3,3}\ \ a_{2,3}\ \ \ldots$
$b\{3,1\} = b_{1,1}\ \ b_{1,2}\ \ b_{1,3}\ \ b_{1,1}\ \ b_{1,2}\ \ \ldots$
$b\{3,2\} = d\ \ \ b_{2,1}\ \ b_{2,2}\ \ b_{2,3}\ \ b_{2,1}\ \ b_{1,2}\ \ \ldots$
$b\{3,3\} = d\ \ \ d\ \ \ b_{3,1}\ \ b_{3,2}\ \ b_{3,3}\ \ b_{3,1}\ \ b_{3,2}\ \ \ldots$



(a)




```
a(i+1,j) = O a(i,j) ;
b(i-1,j) = O b(i,j) ;
c(i,j+1) = O[ c(i,j) + a(i,j) * b(i,j) ] ;
```

(b)

Figure 5.5
(a) : graphe du réseau pour le produit de deux matrices
(b) : programme de la cellule exprimé à l'aide d'opérateurs vectoriels

Le programme SISYC qui simule ce réseau est:

```
PAR n = 3 ;
INDICE i , j ;
VECT  a(1:n+1,1:n) , b(0:n,1:n) , c(1:n,1:n+1) ;

ENTRE ( MAXT 3n-1 , POUR  j =1, n a(1,j) , POUR j=1,n  b(n,j) ) ;

POUR  i=1,n  FAIRE  c(i,1) = z  FIN ;          ? initialisation des ci,j   à  0 ?

POUR  i=1,n  FAIRE
        POUR  j=1,n  FAIRE

              a(i+1,j )  = O a(i,j) ;
              b(i-1,j )  = O b(i,j) ;                    ? cellule  (i,j)  ?
              c(i, j+1)  = O [ c(i,j) + a(i,j)*b(i,j) ]

                    FIN
            FIN ;

SORTIE ( POUR i=1,n  c(i,n+1) ) ;
```

Après avoir compilé ce programme et lu les données, on visualise sur l'écran les vecteurs sorties qui correspondent aux éléments de la matrice C. Nous avons récuperé la table des opérations irrégulières ci-dessous; où T(tps,nbr) est le nombre des opérations irrégulières effectuées par le réseau à l'instant t=tps.

tps:	1	2	3	4	5	6	7	8
nbr:	27	25	20	11	4	0	4	11

5.4.2. Inversion d'une matrice

L'exemple que nous traitons maintenant est dû à P. Common Y.Robert et [CoR 85]. Le réseau lit deux matrices A et B de tailles nxn et délivre en sortie le produit $A^{-1}B$. Cet algorithme, basé sur la méthode d'élimination de Gauss Jordan, est conçu pour la résolution de n systèmes linéaires (on peut prendre B de taille nxp et ne résoudre que p systèmes). Afin d'uniformiser le réseau, nous l'avons légèrement modifié et nous avons pris l'identité à la place de la matrice B pour calculer l'inverse de A.

Comme indiqué par la figure 5.6(a), ce réseau calcule $(-A^{-1})$. Les petits carrées représentent des cellules fictives d'entrées / sorties du réseau.

Chacune des n étapes de l'algorithme de Jordan est implémentée par une ligne du réseau. Après avoir lu toute la matrice A, le réseau agit comme un opérateur linéaire qui transforme toute matrice B en $-A^{-1}B$ (cet exemple correspond à B=Identité).

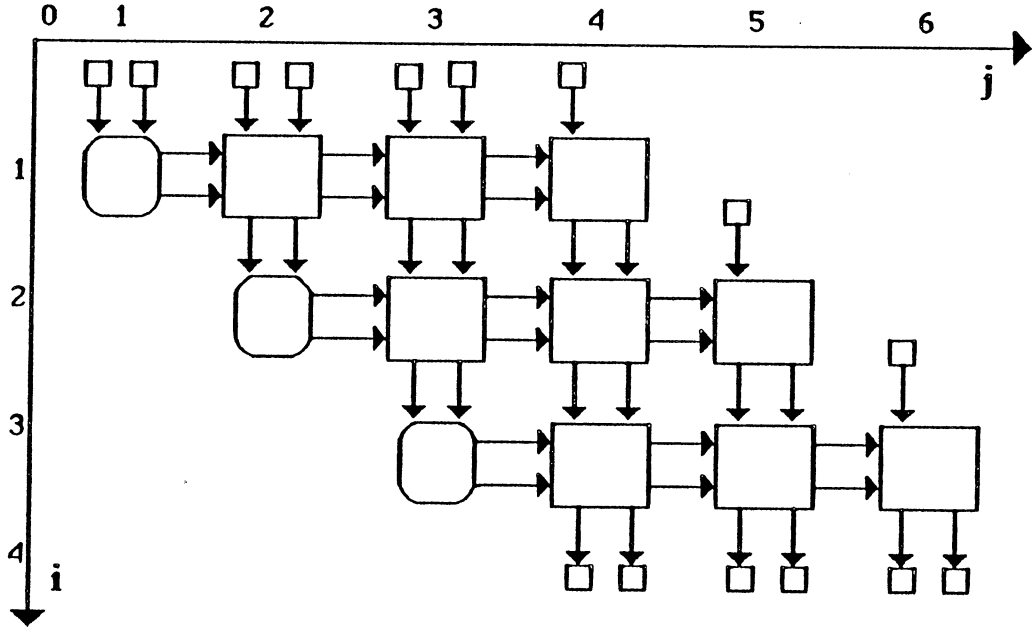
Le signal e vaut 1 lorsqu'il entre, dans le réseau, au même instant qu'un élément diagonal de la matrice A, autrement il vaut d.

Une cellule carrée fonctionne ainsi:

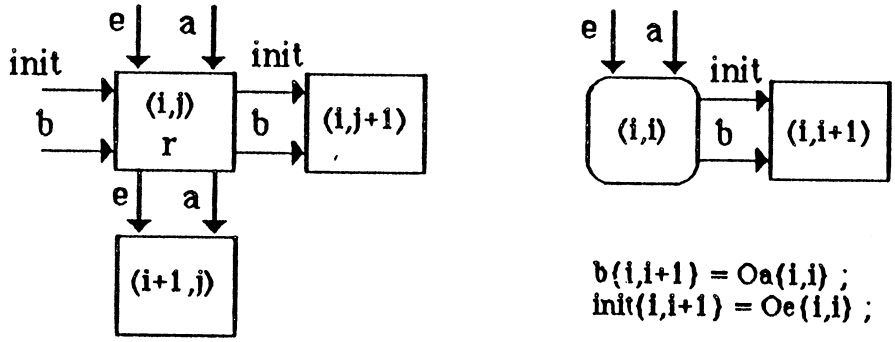
si le signal init est vrai alors elle génère un pivot qu'elle stocke dans son registre r (représenté par un arc à l'intérieur d'elle),

sinon elle applique le pivot, qu'elle a stocké auparavant dans r, au couple d'éléments qu'elle reçoit verticalement et horizontalement.

Le programme des cellules exprimé à l'aide d'opérateurs vectoriels est donné par la figure 5.6(b).



(a)




```
b(i,i+1) = Oa(i,i) ;
init(i,i+1) = Oe(i,i) ;
```

```
e(i+1,j) = O e(i,j) ;
init(i,j+1) = Oinit(i,j) ;
b(i,j+1) = Ob(i,j) ;
a(i+1,j) = O[ a(i,j) - r(i,j) * b(i,j) ] ;
SI( Oinit(i,j) = u ) { r(i,j) = O[ a(i,j) / b(i,j) ] ; } ;
SI( Oinit(i,j) ≠ u ) { r(i,j) = Or(i,j) ; } ;
```

(b)

Figure 5: (a) graphe du réseau systolique
(b) programme des cellules de base

Les cellules circulaires n'effectuent aucune modification et celles représentées par des petits carrés sont des cellules d'entrées/sorties du réseau.
Avec les données ci-dessous le réseau calcule $-A^{-1}$ en temps 5n-1 (14 dans ce cas).

```
a{1,1} =   a(1,1)   a(1,2)   a(1,3)   1       0       0   ...
a{1,2} =   d        a(2,1)   a(2,2)   a(2,3)  0       1       0   ...
a{1,3} =   d        d        a(3,1)   a(3,2)  a(3,3)  0       0       1   ...
a{1,4} =   d        d        d        1       0       0       0       0   0   ...
a{1,5} =   d        d        d        d       d       0       1       0   0
           0        0   ...
a{1,6} =   d        d        d        d       d       d       d       0   0
           1        0        0   ...
e{1,1} =   1   ...
e{1,2} =   d        1   ...
e{1,3} =   d        d        1   ...
```

Le programe en SISYC qui simule ce réseau est ci-dessous:

```
PAR n = 3 ;
INDICE i , j ;
VECT a( 1:n+1 ,1:2n ), b( 1:n , 1:2n+1), r(1:n , 1:2n), e(1:n+1 , 1:2n) , init(1:n , 1:2n ) ;

ENTRE ( MAXT 14, POUR j=1, n  a(1,j), POUR  i= 1,n a(i,n+i),POUR  i= 1,n e(1,i) );

POUR i=1 , n FAIRE

                b(i,i+1) = O a(i,i) ;
                init(i,i+1) = O e(i,i) ;

                POUR  j=i+1 , n+i  FAIRE

                        e(i+1,j)    = Oe(i,j) ;
                        b(i,j+1)    = Ob(i,j) ;
                        init( i,j+1) = O init(i,j) ;
                        a(i+1,j)    = O[ a(i,j) - r(i,j)*b(i,j) ] ;

                        SI ( Oinit(i,j) =true ) {  r(i,j) = O[ a(i,j) / b(i,j) ] ;  } ;
                        SI ( Oinit(i,j) ≠ true ) {  r(i,j) = Or(i,j)  ;  }

                                        FIN
        FIN ;

SORTIE ( POUR i=1 , n    a(n+1, i+n) ) ;
```

Commentaires:

• u désigne le vecteur unité. Il est prédéfini, il n'est pas donc nécessaire de le redéclarer. D'autre part, nous avons déclaré des tableaux rectangulaires de vecteurs a{1:n+1,1:2*n},... alors que nous n'avons besoin que de leurs moitiés supérieures. Pour remedier à cet inconvenient, une solution est de faire un changement du repére (i,j).

• init{i,j}(t)=1 si et seulement

$$t = 2i + j - 2,$$

donc on peut utiliser la variable temp à la place des deux tableaux de vecteurs e et init. Le programme devient alors:

```
PAR n = 3 ;
INDICE i , j ;
VECT  a( 1:n+1 ,1:2n )  , b( 1:n , 1:2n+1)  , r(1:n , 1:2n);

ENTRE ( MAXT 14, POUR  j=1, n   a(1,j), POUR   i= 1,n  a(i,n+i) );

 POUR i=1 , n  FAIRE

                b(i,i+1) = O a(i,i) ;

                POUR  j=i+1 , n+i  FAIRE

                        b(i,j+1) = Ob(i,j) ;
                        a(i+1,j)  = O[  a(i,j) - r(i,j)*b(i,j)  ] ;
                        SI ( temp = 2i+j-1) {  r(i,j) = O[ a(i,j) / b(i,j) ] ;  } ;
                        SI ( temp ≠ 2i+j-1 ) {  r(i,j) = Or(i,j)  ;  }

                                        FIN
                FIN ;

SORTIE ( POUR i=1 , n     a(n+1,i+n) ) ;
```

5.4.3. Résolution d'un système linéaire

Nombreuses sont les solutions systoliques proposées dans la littérature pour résoudre un système linéaire. Nous avons choisi de simuler le réseau de M.Cosnard, Y.Robert et M.Tchuente [CRT 86] car il a une forme triangulaire et n'est composé que de deux types de cellules.

Il s'agit de résoudre un système Ax=b;
où

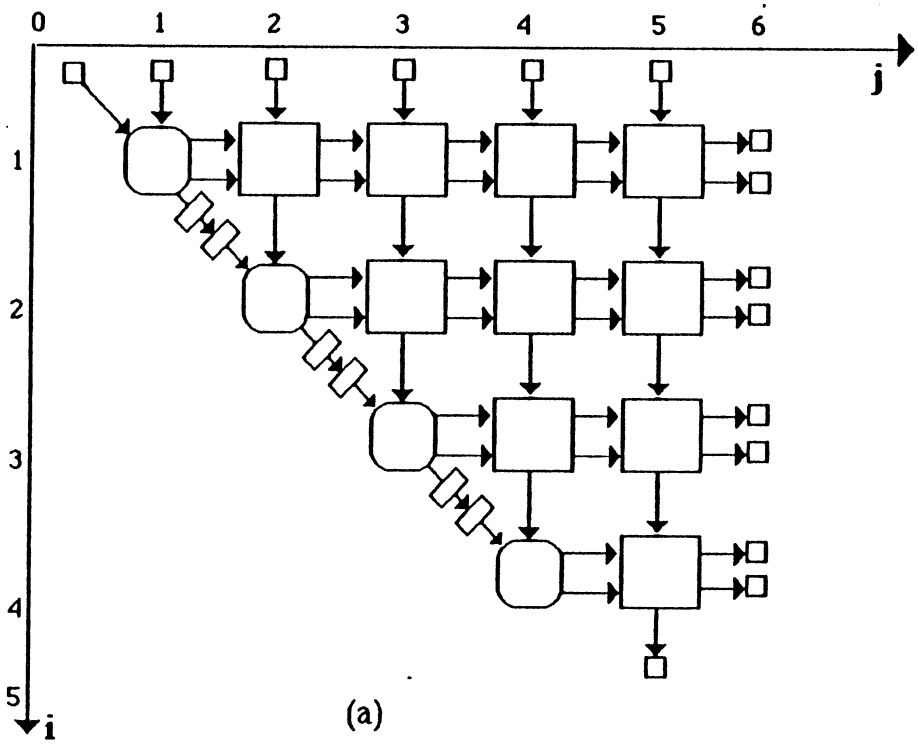
A est une matrice carrée d'ordre n,
x et b sont deux vecteurs de longueur n.

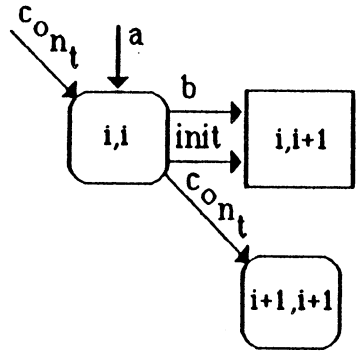
La solution proposée consiste à transformer le système initial en un système triangulaire Tx=b' et ensuite à diagonaliser la matrice T. Ces deux étapes reposent sur la méthode d'élimination de Gauss.

La figure 5.6(a), illustre cette solution pour n=4. La matrice A et le vecteur b sont délivrés au réseau par le haut et la solution du système est récupérée en sortie de la cellule (n+1,n). Le fonctionnement des cellules dépend de la valeur du signal contrôle.

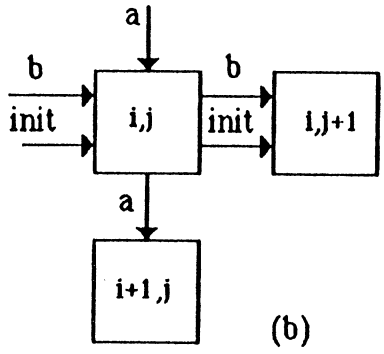
En terme d'opérateurs vectoriels, ce programme est illustré par la figure 5.6(b), où sont supprimées les bascules qui séparent la cellule (i,i) et (i+1,i+1). Ainsi,

contr(i+1,i+1) = O(3)contr(i,i).






```
cont(i+1,i+1)=O(3)cont(i,i) ;
init(i,i+1) = O cont(i,i) ;
SI( Ocont(i,i)=u ) ( r(i,i)=Oa(i,i) ;);
SI( Ocont(i,i)=2.0.u) (b(i,i+1)=O[-a(i,i)/r(i,i) ];);
SI( Ocont(i,i)=3.0.u) ( b(i,i+1)=O[ u/r(i,i) ];) ;
```




```
b(i,j+1)=Ob(i,j) ;
SI( Oinit(i,j)=u ) ( r(i,j) = Oa(i,j) ; ) ;
SI( Oinit(i,j) =2.0.u ) ( a(i+1,j)=O [ a(i,j) +r(i,j)*b(i,j) ] ; ) ;
SI( Oinit(i,j) =3.0.u ) ( a(i+1,j) = O[ r(i,j)*b(i,j) ] ; ) ;
```

Figure 6: (a) graphe du réseau de la résolution d'un système dense
(b) programme des cellules de base.

Avec les données ci-dessous le réseau, composé de n(n+1)/2+n cellules, résoud le système Ax=b en temps 4n.

```
contr{1}  =     1      2      2      2      3  ...
a{1,1}    =   a(1,1)   a(2,1)  a(3,1)  a(4,1)  ...
a{1,2}    =   d        a(1,2)  a(2,2)  a(3,2)  a(4,2)   ...
a{1,3}    =   d        d       a(1,3)  a(2,3)  a(3,3)   a(4,3)  ...
a{1,4}    =   d        d       d       a(1,4)  a(2,4)   a(3,4)  a(4,4)  ...
a{1,5}    =   d        d       d       d       b(1)     b(2)    b(3)   b(4) ...
```

Le programme SISYC qui simule le réseau de la figure 6 est ci-dessous:

```
PAR n=4 ;
INDICE i, j ;
VECT contr{1:n+1}, init{1:n+1,1:n+1}, r{1:n,1:n}, b{1:n,1:n+1}, a{1:n+1,1:n+1};

ENTRE( MAXT 4*n, contr{1}, POUR j=1,n+1 a{1,j} );

POUR i=1,n FAIRE

        contr{i+1} = O{3} contr{i} ;
        init{i,i+1} = O contr{i} ;
        SI( O contr{i} = u ) { r{i,i} = O a{i,i} ; } ;
        SI( O contr{i} = 2.0.u ) { b{i,i+1} = O[ - a{i,i} / r{i,i} ] ; } ;
        SI( O contr{i} = 3.0.u ) { b{i,i+1} = O[ u / r{i,i} ] ; } ;

        POUR j=i, n+1 FAIRE

                b{i,j+1} = O b{i,j} ;
                SI( Oinit{i,j}=u ) { r{i,j}= O a{i,j} ; } ;
                SI( Oinit{i,j}=2.0.u ) { a{i+1,j}=O[ a{i+1,j}=O[ a{i,j}+r{i,j}*b{i,j} ] ; } ;
                SI( Oinit{i,j}=3.0.u ) { a{i+1,j}=O[ r{i,j}*b{i,j} ] ; } ;

                        FIN ;
        FIN;

SORTIE( a{n+1,n+1} ) ;
```

5.4.4. Décomposition QR par la méthode de Givens.

Il s'agit de décomposer une matrice A de taille nxn à l'aide de la méhode d'élimination de Givens (cf. chap 3 pour des détails sur ce problème). Cet exemple traite le réseau de M.Cosnard et Y.Robert [CoR 86].

Le réseau bidimentionnel qu'ils proposent (cas n=6) est illustré par la figure 5.7(a); où nous n'avons pas représenté tous les arcs. Il est composé de cellules ayant un fonctionnement assez complexe, ce qui entraîne un programme SISYC, simulant ce réseau, de même complexité.

Le réseau fonctionne en deux phases. Au cours de la première, les coefficients de la matrice circulent vers la droite.

Les cellules (i,j); $1 \leq i,j \leq n$, génèrent des rotations si les registres

x{i,j} et y{i,j} sont chargés,

autrement elles appliquent les rotations dont les paramètres circulent via les canaux c et s. Durant cette phase, la cellule(1,1) reçoit le signal

r{1,1}(t) = vrai; $1 \leq t \leq n$.

La deuxiéme phase débute lorsque la cellule (1,1) reçoit le signal

r{1,1}(n) = faux.

A partir de cet instant, les coefficients de la matrice A circulent vers la gauche. Ainsi, un réseau de taille $3n(n-2)/8$ cellules calcule la décomposition QR d'une matrice d'ordre nxn en temps 3n-1 qui est asymptotiquement optimal.

La figure 5.7(b) illustre la cellule de base.

- Les coefficients de la matrice à factoriser sont représentés par les vecteurs a et b.

- Le booléen r contrôle le sens des transferts de données (gauche/droite).

- Le signal (e,c,s) spécifie la nature de l'opération de chaque cellule (e est un booleen et (c,s) sont les paramétres de la rotation).

Le programme P(i,j) de la cellule (i,j) (cf. figure 5.7(b)), exprimé en terme de vecteurs, est ci-dessous:

```
debut :

        SI ( Oy(i,j) = nil ) { r(i+1,j) = vrai ; } ;
        SI ( Oy(i,j) ≠ nil ) { r(i+1,j) = faux ; } ;

        SI ( Og(i,j) = vrai ) {        ? génère la rotation ?

                                SI ( Zy(i,j) = z ) { c(i+1,j) = u ; s(i+1,j) = z ; } ;

                                SI ( Zy(i,j) ≠ z ) { t(i,j) = u + Z[ x(i,j)^2 / y(i,j)^2 ] ;
                                                     s(i+1,j) = u / RAC t(i,j) ;
                                                     c(i+1,j) = Z x(i,j) / [ Zy(i,j)*RAC t(i,j) ] ;
                                                   } ;

                                x(i,j) = c(i+1,j) * Zx(i,j) + s(i+1,j) * Zy(i,j) ;
                                y(i,j) = nil ;
                              } ;
        SI ( Og(i,j) ≠ faux ) {    ? applique la rotation ?

                                x(i,j) = Z [ x(i,j)*c(i,j) + y(i,j)*s(i,j) ] ;
                                y(i,j) = Z [ y(i,j)*c(i,j) - x(i,j)*s(i,j) ] ;
                                c(i+1,j) = Zc(i,j) ; s(i+1,j) = Zs(i,j) ;
                              } ;
        SI ( Or(i,j) ≠ faux ) {        ? envoie à droite ?

                                a(i+1,j) = y(i,j) ;
                                y(i,j) = x(i,j) ;
                                x(i,j) = Za(i,j) ;
                                b(i,j-1) = Zb(i,j) ;
                              } ;
        SI ( Or(i,j) = faux ) {        ? envoie à gauche ?

                                b(i,j-1) = x(i,j) ;
                                x(i,j) = y(i,j) ;
                                y(i,j) = Zb(i,j) ;
                                a(i+1,j) = Za(i,j) ;
                              } ;
      r(i+1,j) = Or(i,j) ;

fin_prg_cell
```

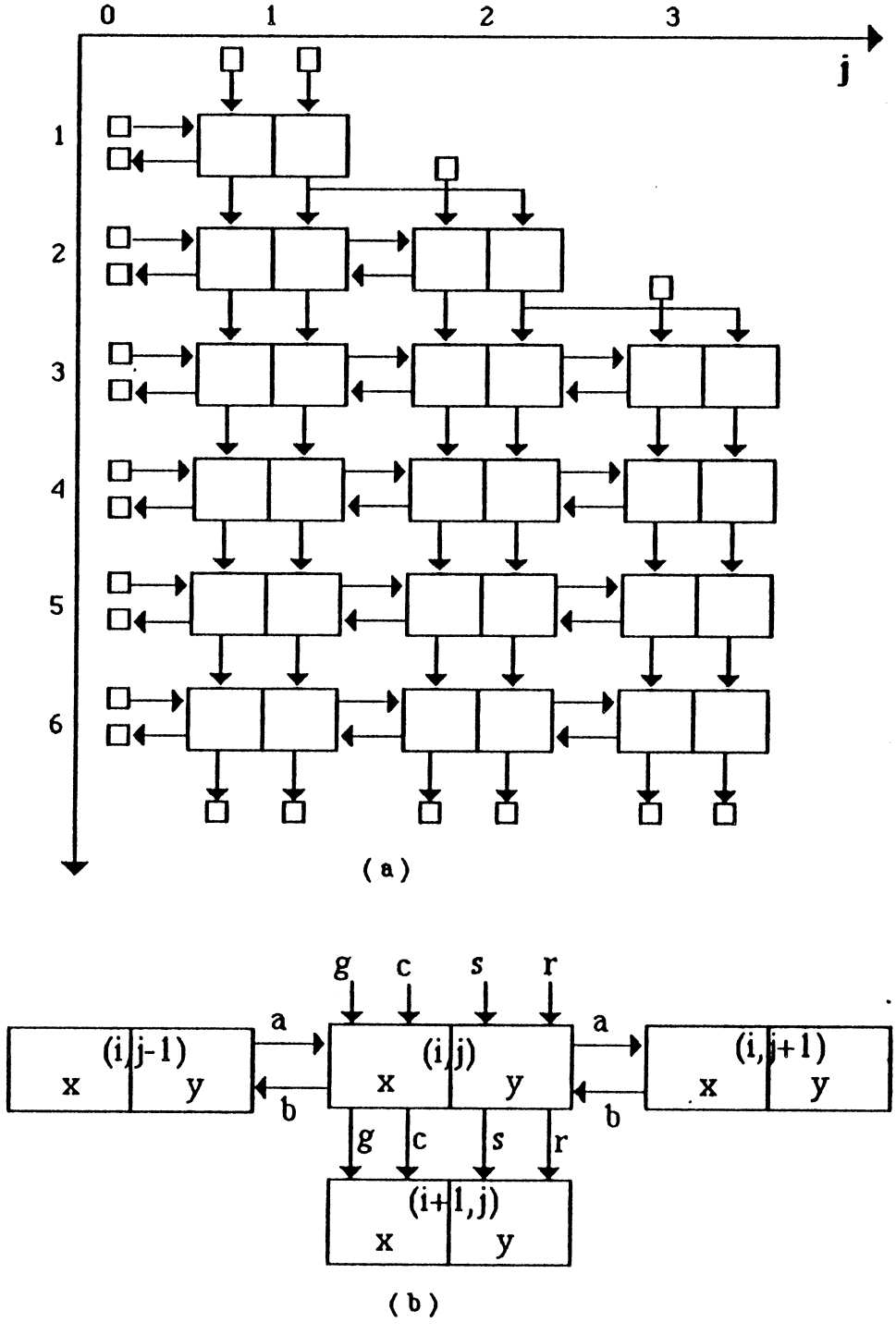


Figure 7 (a): graphe du réseau pour la décomposition QR
(b) cellule de base et couleurs des arcs.

Le programme en SISYC qui simule le réseau de la décomposition QR est :

```
PAR   p = 4 , n = 6 ;
INDICE i , j ;
VECT        a{1:n,1:p+1} , b{1:n,0:p} , g{1:n+1,1:p+1} , r{1:n,1:p+1} , c{1:n+1,1:p} ,
            s{1:n+1,1:p}, x{1:n,1:p} , y{1:n,1:p} , t{1:n,1:p}  ;

ENTRE ( MAXT 18 ,  POUR i = 1 , n  a{i,1}  , r{1,1} ) ;

POUR  i=1 , p  FAIRE

                r{i+1,i+1} = Or{i,i}  ;            ? cellule  (i,i )  ?
                g{i,i} = Z{i+1}vrai  ;

                FIN  ;

POUR  i=1 , n  FAIRE
              POUR  j=1, INF(p,i ) FAIRE        ? cellule (i,j)  ?

                    P(i,j)  ;
               FIN
FIN ;

SORTIE ( POUR  i=1 , n  b{i,0}  ) ;
```

Commentaires:

• Le programme des cellules (i,i) diffère de celui des cellules (i,j) i≠j. En effet, une cellule (i,i) reçoit de l'extérieur le signal g{i,i}=Z{i+1}vrai et envoie à la cellule (i+1,i+1) le signal Or{i,i}.

• Les variables vrai, faux et nil doivent être déclarées et initialisées par: vrai=u, faux=z et nil = d.

• SISYC n'étant pas structuré, il faut recopier intégralement P(i,j) dans le programme principal.

5.5. Conclusion

Nous avons présenté dans ce chapitre un outil logiciel pour la simulation pratique des algorithmes systoliques. Il est plus simple et plus général que que le système de Melhem. En effet, SISYC autorise les conditionnels, ceci a un double intérêt.

- La classe des réseaux systoliques qui peuvent être simulés en SISYC a été élargie.
- La programmation en SISYC est très simple.

Les exemples que nous avons traités, dans le paragraphe 5.4, témoignent de la simplicité et de la puissance de SISYC.

Pour élargir SISYC aux réseaux asynchrones, il faut le structurer et y introduire le constructeur alt qui existe dans Lustre et Occam, ... D'autre part, des opérateurs tels que le while, goto, ... dont on ne parle pas en systolique, n'existent pas dans SISYC. Nous n'avons pas essayé de les implémenter en SISYC, toutefois, il nous semble que c'est une tâche réalisable si de tels opérateurs s'avèrent utiles.

Chapitre 6

VALIDATION DES ALGORITHMES SYSTOLIQUES

6.1. Introduction

La preuve de correction des algorithmes séquentiels est depuis longtemps étudiée. Plusieurs méthodes ont été proposées dont les plus connues sont:

- La méthode de Floyd revue dans [CoC 87] pour la preuve des propriétés d'invariance

- Les techniques de la logique temporelle [AFE 88].

La généralisation de ces méthodes aux algorithmes systoliques (ou aux programmes parallèles) s'est heurtée à de sérieux problèmes:

- Ces techniques qui n'ont donné lieu à aucune réalisation pratique [AFE 88], sont basées sur des formalismes très lourds.

- Elles ne sont applicables qu'à des petits programmes séquentiels, et leur application aux réseaux systoliques n'est possible que sur des exemples triviaux.

Au vu de ces remarques, nous n'attaquons pas dans ce chapitre, le difficile problème de la preuve des algorithmes systoliques. Nous nous intéressons plutôt à leur validation. Plus précisément, nous montrons comment construire une formulation séquentielle équivalente à un algorithme systolique donné.

Il convient de noter que si un algorithme systolique (A') parallélise un algorithme séquentiel (A), il se peut que notre méthode produise non pas (A) mais un algorithme séquentiel (P) équivalent à (A). Nous supposons donc résolu le

problème de l'équivalence des algorithmes séquentiels. Cette supposition est raisonnable dans tous les cas usuels.

Ainsi, la preuve de correction d'un algorithme systolique se ramène à celle de l'algorithme séquentiel qui lui est équivalent.

Considérons donc, un concepteur d'algorithmes systoliques, qui a pris comme point de départ un algorithme séquentiel et a conçu une architecture systolique capable de l'implémenter. Son problème est de s'assurer que l'architecture exécute correctement l'algorithme séquentiel de départ. A ce niveau, l'approche par séquentialisation que nous proposons pour ce concepteur est un moyen de validation et non de preuve. Elle suppose néamoins, comme nous l'avons souligné plus haut, que le concepteur soit capable de reconnaitre des formes équivalentes des algorithmes séquentiels.

La séquentialisation fait le travail inverse de la synthèse automatique des algorithmes systoliques. En effet, dans leur thèse, B. Joinnault et P. Gachet [JoG 87], proposent une méthode qui permet de déduire de la formulation mathématique d'un problème, un système d'équations récurrentes uniformes (SERU), équivalent à cette formulation. La méthode permet de réécrire la formulation mathématique à l'aide de variables pipelinées. On obtient alors un système d'équations récurrentes linéaires (SERL). Ensuite, on uniformise le SERL pour obtenir un système d'équations récurrentes uniformes. La méthode de projection des dépendances [Quin 85], que nous avons exposée au chapitre 1, donne alors l'algorithme systolique qui résoud le problème de départ.

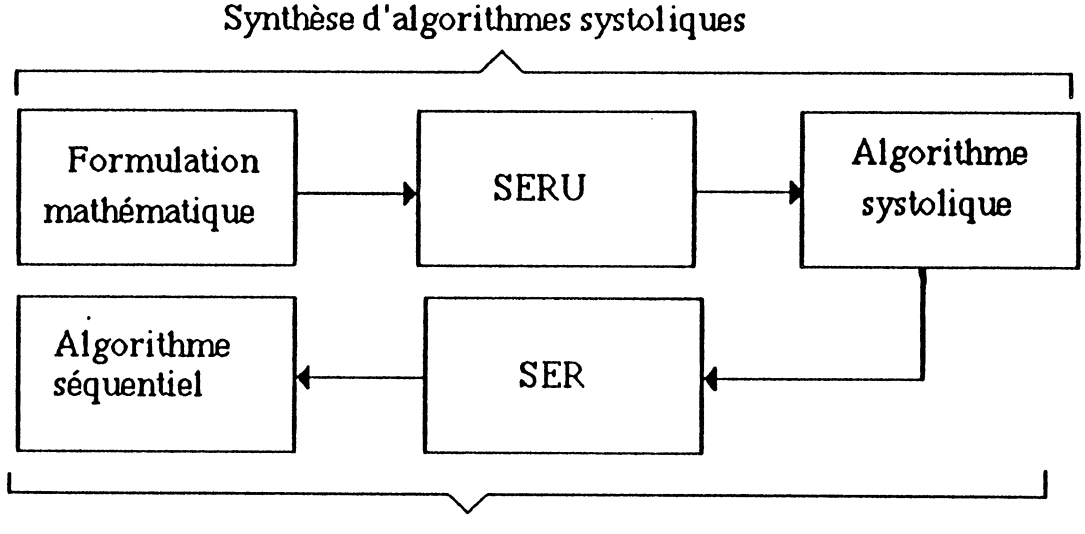


Figure 6.1.

La séquentialisation prend comme point de départ un algorithme systolique, décrit par un système d'équations récurrentes (SER), et le transforme en un algorithme séquentiel usuel. Contrairement à la synthèse des algorithmes systoliques , le travail inverse est facilement réalisable.

Exemple 6.1

Nous illustrons cette méthode sur l'exemple simple du produit de convolution.

Etant donné un vecteur (w_i , $0 \leq i \leq 2$), et une suite réelle (x_i , $i \geq 0$), un algorithme séquentiel du produit de convolution est (A):

```
        pour i = 0 , n faire
              pour k = 0, 2 faire
( A )                 y_i := y_i + w_k . x_{i-k}
        fin_pour.
```

Le réseau systolique qui effectue le produit de convolution est que nous voulons valider est donné par la figure 6.2 ci-dessous:

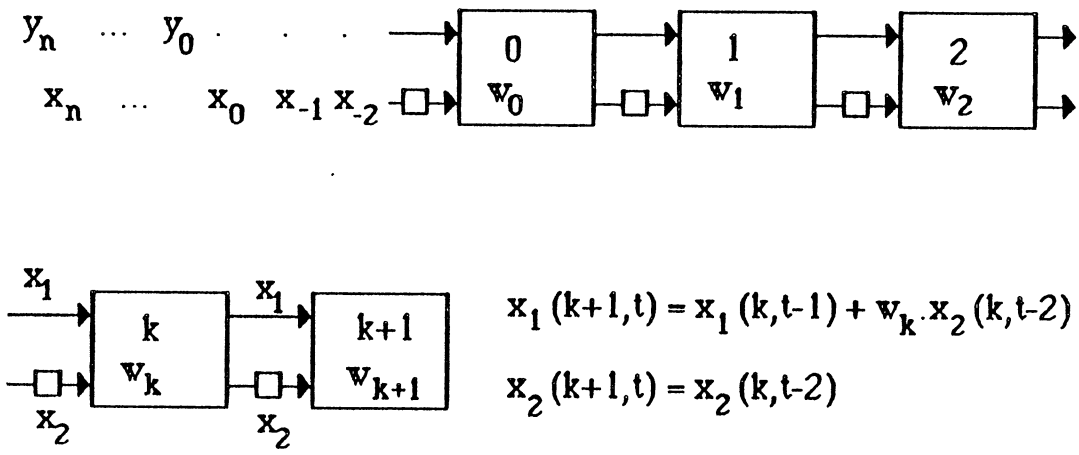


Figure 6.2. réseau systolique pour la convolution

Nous avons représenté les flux de données qui passent par le canal supérieur (resp. inférieur) par x_1 (resp. par x_2). Ce choix de coloration des arcs n'a aucune influence sur le résultat final.
Nous abordons maintenant les étapes successives de la méthode de séquentialisation.

- Etape 1: Description de l'algorithme systolique

Il est clair, qu'avec les notations ci-dessous, l'algorithme exécuté par le réseau de la figure est (A'):

pour $t = 1 , n + 6$ faire
pour $k = 0 , 2$ faire

(A') $x_1(k+1,t) = x_1(k,t-1) + w_k \cdot x_2(k,t-2)$;
$x_2(k+1,t) = x_2(k,t-2)$;

fin_pour

avec
$x_1(0,t) = y_{t-4} , \quad 4 \le t \le n + 4$
et $x_2(0,t) = x_{t-3} , \quad 1 \le t \le n + 3.$

L'algorithme systolique ci-dessus est valide, si on peut établir l'équivalence des algorithmes (A') et (A). Pour ce faire, nous identifions les variables $x_i(k,t)$, $i = 1, 2$, en fonction des données x_i, y_i et w_k.

- Etape 2: Identification des variables

Nous introduison pour cela, la fonction de nommination N. Cette dernière permet d'associer à tout signal sortant d'une cellule, un signal entrant, et de leur affecter le même nom. Nous la définissons comme suit:

$x_1(0,t) = y_{t-4}$ est une donnée de l'algorithme systolique, on pose alors
$N(x_1(0,t)) = y_{t-4} , \quad 0 \le t - 4 \le n$
$x_2(0,t) = x_{t-3}$ est une donnée de l'algorithme systolique, on pose alors
$N(x_2(0,t)) = x_{t-3} , \quad 1 \le t \le n$

Considérons maintenant le programme d'une cellule

$x_1(k+1,t) = x_1(k,t-1) + w_k \cdot x_2(k,t-2)$;
$x_2(k+1,t) = x_2(k,t-2)$;

C'est un système d'équations récurrentes triangulaires, où

$x_1(k+1,t)$ dépend de $x_1(k,t-1)$ et
$x_2(k+1,t)$ dépend de $x_2(k,t-2)$

Nous identifions $x_1(k+1,t)$ et $x_1(k,t-1)$ à la même variable, ce qui se traduit par:

$$N(x_1(k+1,t)) = N(x_1(k,t-1))$$

De même, nous identifions $x_2(k+1,t)$ et $x_2(k,t-2)$ à la même variable, ce qui se traduit par:

$$N(x_2(k+1,t)) = N(x_2(k,t-2))$$

Nous avons donc un système récurrent (S) qu'il faut résoudre pour avoir les $N(x_i(k,t))$, $i = 1, 2$, en fonction des données x_i, y_i et w_k.

$$(S)\quad \begin{cases} N(x_1(k+1,t)) = N(x_1(k,t-1)) & (1) \\ N(x_2(k+1,t)) = N(x_2(k,t-2)) & (2) \end{cases}$$

Avec les conditions initiales (I) ci-dessous:

$$(I)\quad \begin{cases} N(x_1(0,t)) = y_{t-4}\ , & 0 \le t-4 \le n, \\ N(x_2(0,t)) = x_{t-3}\ , & 1 \le t \le n, \end{cases}$$

La résolution du système (S) est très simple. En effet, on déduit de (1) que

$$N(x_1(k,t)) = N(x_1(0,t-k)) = y_{t-k-4}\ ,\ 0 \le t-k-4 \le n,$$

et de (2) , on déduit que

$$N(x_2(k,t)) = N(x_2(0,t-2k)) = x_{t-2k-3}\ ,\ 1 \le t-2k-3 \le n,$$

- Etape 3: Transformation de (A') en un algorithme (P)

La transformation consiste tout simplement remplacer chaque variable $x_i(k,t)$ dans (A') par $N(x_i(k,t))$, $i = , 2$. L'algorithme (P) obtenu est donc :

pour $t = 1 , n + 6$ faire
pour $k = 0 , 2$ faire

(P) si $0 \leq t-k-5 \leq n$ alors
$y_{t-k-5} = y_{t-k-5} + w_k \cdot x_{t-2k-5}$.

Dans le cas general, on retrouve, une formulation séquentielle comportant les instructions du type

$x := F(... , x , ...)$

dans lesquelles les nouvelles valeurs générées écrasent les anciennes. Cette technique est indispensable pour économiser la place mémoire en programmation séquentielle. Dans le cas particulier de signaux qui circulent de cellule en cellule sans changer de valeur, on obtient après nommination, des instructions du type

$a := a$

qui peuvent évidement être supprimées de la nouvelle formulation. D'autre part, les instructions vides disparaissent dans la formulation séquentielle.

- Etape 4: Transformation de (P) en (A)

Nous pouvons maintenant facilement établir l'équivalence entre l'algorithme (P) et l'algorithme (A). Il suffit pour cela, de faire le changement de variable

$t = i + k - 5$

En conclusion, nous avons montré, Par transformations successives, que l'algorithme systolique pour le produit de convolution (cf. figure. 6.2) est valide.

6.2. Transformation des algorithmes systoliques décrits par des SERT.

Nous nous limitons dans ce chapitre aux réseaux systoliques decrits par des systèmes d'equations récurrentes triangulaires (SERT). Nous traitons plus loin, un exemple de réseau systolique plus général. Il s'agit du réseau pour l'inversion d'une matrice [RoT 85].

Nous présentons maintenant, les étapes successives de la séquentialisation.

- Etape 1: description de l'algorithme systolique

Un réseau systolique est decrit par un SERT si chacune de ses cellules a un fonctionnement qui peut être représenté par un SERT comme ci-dessous:

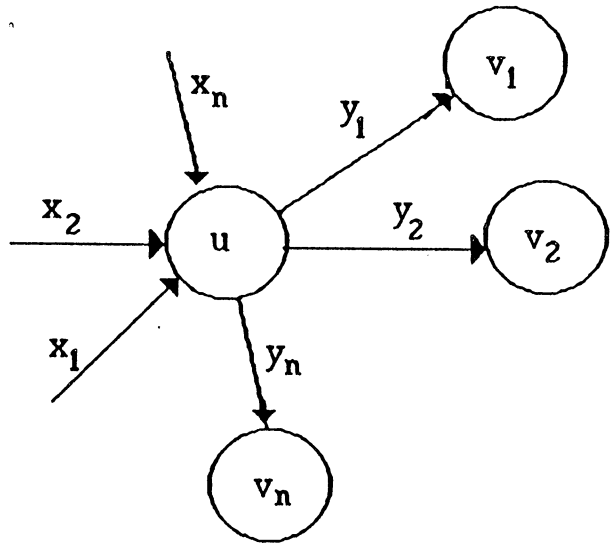


Figure 6.3

P(u) :
$$y_1(v_1,t) = f_1(\, x_1(u,t-h_1)\,,\, x_1(u,t-h_1),\ \ldots\ ,\, x_n(u,t-h_n)\,)$$
$$y_2(v_2,t) = f_2(\, x_2(u,t-h_2)\,,\ \ldots\ ,\, x_n(u,t-h_n)\,)$$
$$\ldots$$
$$y_n(v_n,t) = f_n(\, x_n(u,t-h_n)\,)$$

où pour tout i, $1 \le i \le n$,

- $x_i(u,t-h_i)$ = la donnée qui entre dans la cellule u par le canal de couleur x_i à l'instant $t - h_i$.
- h_i = le nombre de registres que doit traverser une donnée pour aller de la cellule u à la cellule v_i en passant par le canal de couleur i
- f_i = fonction de $n - i + 1$ variables.

Il est clair que le programme exécuté par le réseau au cours de T tops est:

```
        pour  t = 1 , T  faire
              pour toute cellule u du réseau faire
( A' )              exécuter  P ( u )
              fin_pour.
```

Appelons (A') cet algorithme. Il présente deux caractéristiques principales:

- C'est un programme à assignation unique, c'est à dire qu'au cours de son déroulement , chaque variable est modifiée au plus une fois.

- Une variable modifiée à la date t , ne sert à la modification d'autres variables qu'à une date t + h. C'est à dire, elle n'apparait en partie droite d'une équation qu'à l'instant t + h. C'est cette caractéristique de pipeline qui fait qu'un algorithme systolique est plus complexe qu'un simple programme à assignation unique.

- Etape 2: Identification des variables

Nous transformons l'algorithme (A') à l'aide de la fonction N. Comme nous l'avons expliqué dans le premier paragraphe, cette fonction permet d'identifier les variables $x_i(u,t)$. Dans le cas de réseaux systoliques décrits par des SERT, N est définie de manière simple comme ci-dessous:

- Si $y_i(v,t) = a$, est une donnée , de l'algorithme systolique , différente de d alors on pose
  $$N (y_i(v,t)) = a.$$
- Si $y_i(v,t)$ n'est pas une donnée de l'algorithme systolique, il existe forcément une équation (et une seule)
  $$y_2(v_2,t) = f_2(x_2(u,t-h_2) , \ ... \ , x_n(u,t-h_n))$$
  qui définie la variable $y_i(v,t)$. On pose alors,
  $$N (y_i(v,t)) = N (x_i(u,t-h_i)).$$

Nous avons ainsi formé un système récurrent avec des conditions initiales qu'il faut résoudre pour exprimer les $N (y_i(v,t))$ en fonction des données de l'algorithme systolique. Il convient de noter que N n'est pas définie pour les $y_i(v,t)$ qui valent d. De cette manière, lorsqu'on remplace , dans (A'), les variables $y_i(v,t)$ par leur noms effectifs, qui sont les $N ((y_i(v,t))$, les calculs nécessitant le delai d, disparaissent de la nouvelle formulation.

- Etape 3: Transformation de l'algorithme (A')

On remplace dans (A') les variables $y_i(v,t)$ par leurs nouveaux noms, qui sont les $N(y_i(v,t))$. L'algoritme (P) ainsi obtenu est équivalent à l'algorithme de départ (A'). D'autre part, tout instruction de l'algorithme (P) obtenu est de la forme

$$x := F (..., x , ...)$$

dans laquelle la nouvelle valeur générée écrase l'ancienne valeur. Dans le cas particulier de signaux qui circulent de cellule en cellule sans changer de valeur, on obtient après nommination, des instructions du type

$$a := a$$

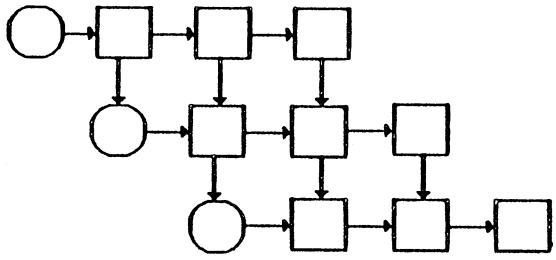
qui peuvent être évidement supprimées de la nouvelle formulation.

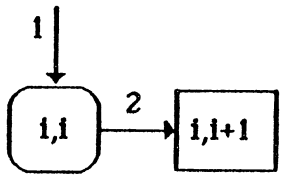
Maintenant, il suffit d'établir l'équivalence de (P) avec l'algorithme séquentiel (A) visé, pour pouvoir se prononcer sur la validité de l'algorithme systolique. Pour cela , il faut voir si on peut réordonner les calculs afin de pouvoir permuter la boucle sur t avec les autres boucles, ou bien, si on peut faire des changements de variable d'indice comme nous l'avons fait pour l'exemple 6.1.

Cette méthode de transformation peut être appliquée à des réseaux plus complexes que ceux représentés par des SERT. Toutefois, ils doivent être composés de cellules ayant aumoins autants d'arcs entrants que d'arcs sortants. Nous allons traiter un exemple, pour illustrer cet aspect.

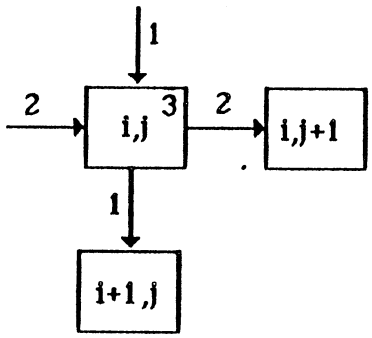
Exemple 6.2.

Le réseau systolique , effectuant l'inversion d'une matrice carrée A, d'ordre n , que nous allons valider est basé sur l'algorithme de Jordan [RoT 85]. Nous l'avons légèrement modifié afin de faire apparaître les conditionnels temporels (cf. chap. 5). Le réseau et les programmes des cellules qui le composent, sont donnés dans la figure 6.4.





$x_2(i,i+1,t) = x_1(i,i,t-1);$



Si (t = 2i+j-1)

$x_1(i+1,j,t) = x_1(i,j,t-1) - x_2(i,j,t-1).x_3(i,j,t-1)$

$x_2(i,j+1,t) = x_2(i,j,t-1)$

$x_3(i,j,t) = x_1(i,j,t-1) / x_3(i,j,t-1)$

Si (t ≠ 2i+j-1)

$x_1(i+1,j,t) = x_1(i,j,t-1) - x_2(i,j,t-1).x_3(i,j,t-1)$

$x_2(i,j+1,t) = x_2(i,j,t-1)$

$x_3(i,j,t) = x_3(i,j,t-1).$

Figure 6.4. réseau systolique et fonctionnement des cellules de base.

- Les données de l'algorithme sont:

$$
\begin{array}{ll}
 & x_1(1,j,t) = a_{j,t-j+1} \quad \text{si } j \le t < 2n+j \ , \\
 & \text{où} \quad \text{les } a_{i,j} \qquad\qquad 1 \le i,j \le n, \text{ sont les éléments de A} \\
(D) & a_{i,n+i} = 1 \quad \text{et} \quad a_{i,n+j}=0 \quad \text{pour } i \ne j. \\
 & x_1(j,j+n,t) = a_{j+n,t-n-2j+2} \quad \text{si } 2j+n-1 \le t < 3n+2j-1 \\
 & \text{où} \quad a_{j+n,j} = 1, \ a_{j+n,i} = 0 \ \text{pour} \ i \ne j
\end{array}
$$

Ceci permet de définir les conditions initiales du système récurrent définissant la fonction N, que nous allons construire .

- Les conditions initiales

$$
\begin{array}{ll}
 & N(x_1(1,j,t)) = a_{j,t-j+1} \quad \text{si} \quad j \le t < 2n+j \\
(I) & \\
 & N(x_1(j,j+n,t)) = a_{j+n,t-j-n} \quad \text{si} \quad 2j+n-1 \le t < 3n+2j-1.
\end{array}
$$

- Définition de la fonction N

Nous proposons de construire une fonction de nommination N par les équations récurrentes ci-dessous:

$$
\begin{array}{ll}
 & N(x_2(i,i+1,t)) = N(x_1(i,i,t-1)) \\
 & \text{si } t = 2i+j-1 \text{ alors} \\
 & \qquad N(x_1(i+1,j,t)) = N(x_3(i,j,t-1)) \ ; \\
 & \qquad N(x_2(i,j+1,t)) = N(x_2(i,j,t-1)) \ ; \\
 & \qquad N(x_3(i,j,t)) = N(x_1(i,j,t-1)) \ ; \\
(S) & \text{si } t \ne 2i+j-1 \text{ alors} \\
 & \qquad N(x_1(i+1,j,t)) = N(x_1(i,j,t-1)) \ ; \\
 & \qquad N(x_2(i,j+1,t)) = N(x_2(i,j,t-1)) \ ; \\
 & \qquad N(x_3(i,j,t)) = N(x_3(i,j,t-1)) \ ;
\end{array}
$$

La résoluton du système (S) avec les conditions initiales (I) donne la solutions ci-dessous:

$$
\begin{array}{ll}
N(x_1(i,j,t)) = a_{j,t-i-j+2} & \text{si } i+j-1 \le t \le 2n+i+j-2, \ 1 \le i \le n \text{ et } i \le j \le n+i \ ; \\
N(x_2(i,j,t)) = a_{i,t-i-j+2} & \text{si } i+j-1 \le t \le 2n+i+j-2 \ , \ 1 \le i \le n \text{ et } i < j \le n+i \ ; \\
N(x_3(i,j,t)) = a_{j,i} & \text{si } 2i+j-1 \le t \le 5n-2.
\end{array}
$$

Après avoir remplacer les variables $x_k(i,j,t)$, k=1, 2, 3, par leur valeurs effectives, nous obtenons l'algorithme (P) ci-dessous:

```
                pour    t = 2, 5n-2    faire
                    pour    i = 1 , n      faire
                            pour    j = i+1 , n+i    faire

( P )                           si   i+j-1 ≤ t ≤ 2n+i+j-2    alors
                                  si   t = 2i+j-1   alors   a_{j,i} = a_{j,i} / a_{i,i} ;
                                   sinon    a_{j,t-i-j+1} = a_{j,t-i-j+1} - a_{j,i} · a_{i,t-i-j+1} ;
```

Un simple changement de variable $t = k + i + j - 1$, dans l'algorithme (P) transforme ce dernier à un algorithme usuel pour l'inversion de la matrice A .

6.3. Le logiciel SISYC2

Nous avons développé un logiciel SISYC2 permettant de générer automatiquement la trace d'un algorithme systolique. Cette trace est définie comme l'ensemble ordonné, dans le temps, des calculs effectués par cet algorithme. Elle peut être exploitée de deux manières:

- Dans le cas général, elle peut être utilisée pour vérifier si les instructions éxecutées par les cellules à tout instant, correspondent bien aux spécifications de départ. Ce problème se pose principalement lorsque les cellules du réseau sont programmables et que l'instruction à exécuter à chaque instant est detérminée par les signaux de contrôle. La consultation de la trace permet de valider ces signaux de contrôle.

- Dans une certaine sous classe de problèmes, les solutions peuvent être exprimées comme des expressions formelles. C'est par exemple, le cas du produit de convolution, qui est équivalent aux calculs des expressions formelles:

$$y_i = w_0 \cdot x_i + w_1 \cdot x_{i-1} + \ldots + w_i \cdot x_0.$$

Pour cette classe de problèmes, on peut fournir à un système de calcul formel (Reduce par exemple) la trace délivrée par SISYC2, ce qui permet d'obtenir en sortie les expressions formelles, que l'on valide en les comparant aux expressions formelles définissant le problème.

L'environnement logiciel que nous avons mis en oeuvre est shématisé par la figure 6.5. Le programme, baptisé ici SISYC2, est écrit en langage C et tourne sous le système d'exploitation UNIX. Il est composé de deux modules principaux:

- Le module de compilation:

Comme pour SISYC (cf. chapitre 5), nous nous sommes servi des générateurs yacc [Joh 74] et lex [LeS 74] pour écrire les deux programmes qui le composent. Le rôle de ce module est double:

- Il vérifie la syntaxe de la description , sous forme d'un système d'équations vectorielles, du réseau systolique.
- Il forme l'arbre de réduction de cette description selon la grammaire de SISYC2.

Cette grammaire est donnée en annexe 1 .

SISYC2

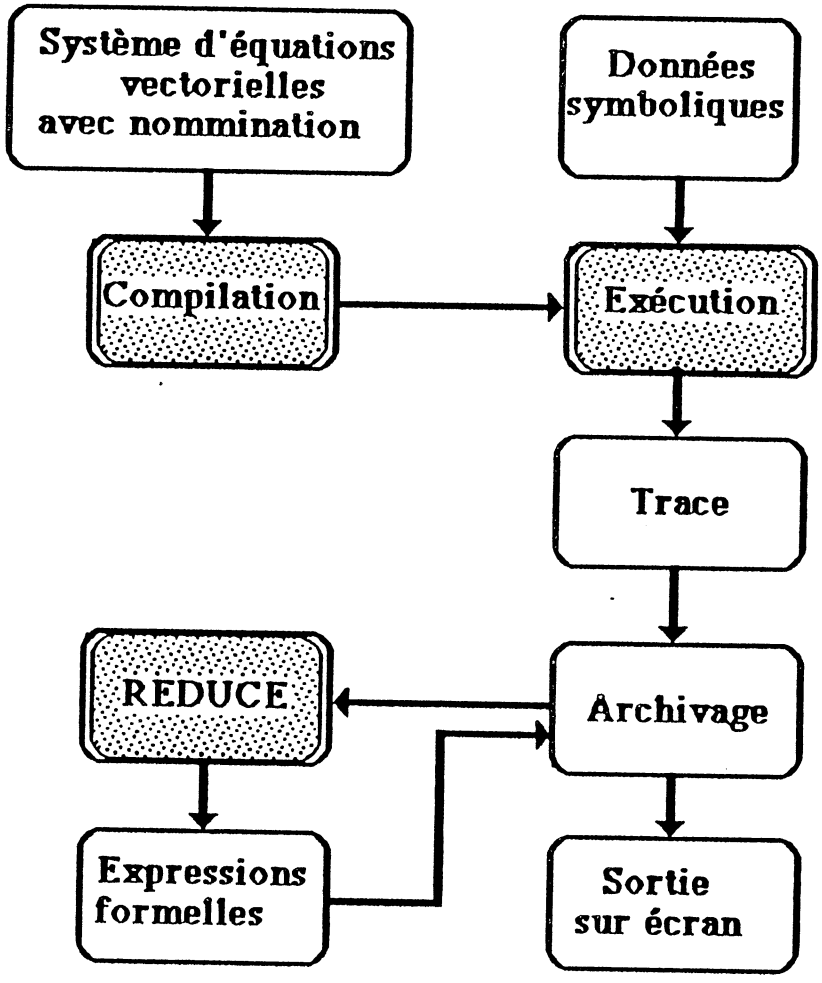


Figure 6.5a

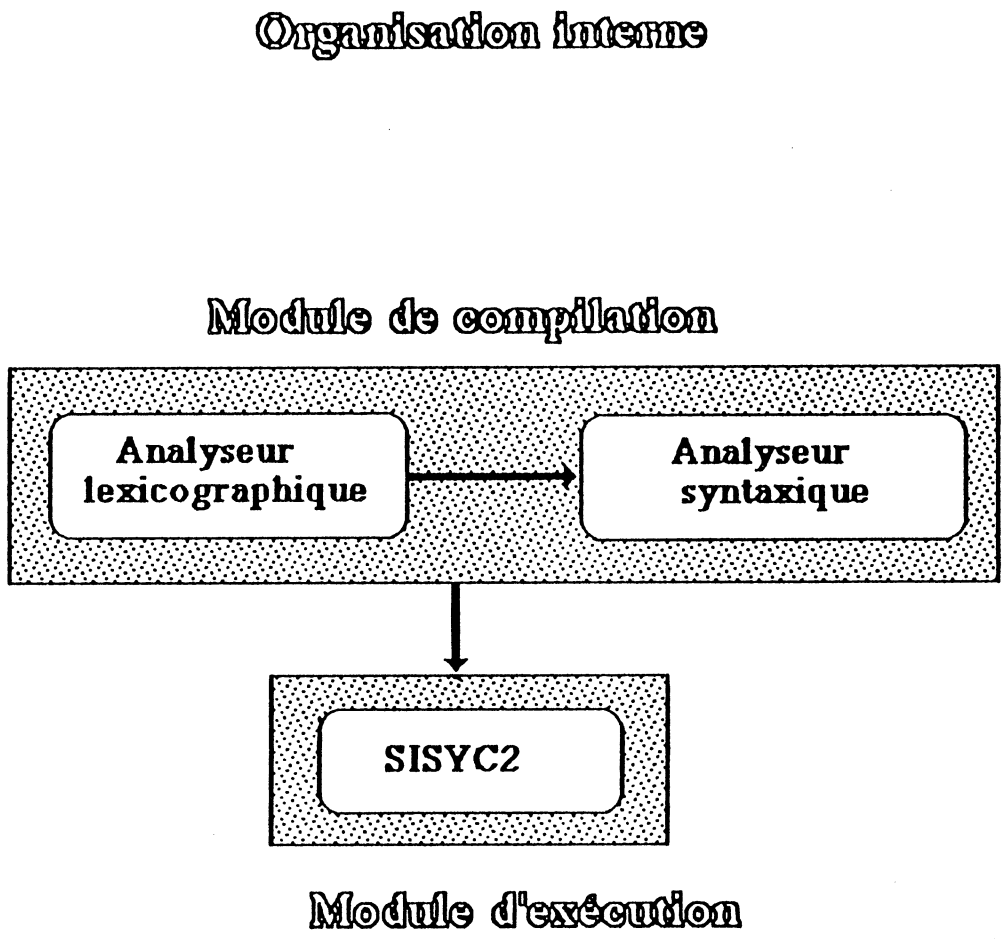

Organisation interne
Module de compilation
Analyseur lexicographique
Analyseur syntaxique
SISYC2
Module d'exécution

Figure 6.5b

6.3.1. Présentation générale

Un programme en SISYC2 est composé de quatres parties:

- les déclarations ,
- les initialisations ,
- les entrées ,
- le corps du programme.

Nous avons jugé inutile d'encombrer SISYC2 par l'introduction de la partie SORTIE (cf. chap. 5).

- Les parties déclarations et entrées sont identiques à celles de SISYC (cf. chapitre 5).

- Dans la partie initialisation on précise les vecteurs qui doivent être initialisés à l'instant t = 1 (start up). Toutes les variables sont initialisées par défaut à d.

- Le corps du programme est une suite d'équations et des conditionnels. Nous commençons par dire ce qu'est une équation, ensuite nous parlerons des conditionnels.

- Les équations

Toute équation dans SISYC2 est du type ci-dessous:

$x\{t\_ind\} = P(O\{k_1\}\ y_1\{t\_ind\} , \ldots , O\{k_i\}\ \hat{}y_i\{t\_ind\} , \ldots, O\{k_n\}\ y_n\{t\_ind\})$

où

- t_ind est un n-upplet qui représente en général le numéro de la cellule, i, j, k par exemple.
- k_i , i = 1, 2, 3, est un entier, et l'operateur O est défini comme suit:

si $y = (a_1, a_2, \ldots, a_n)$ alors

$O\{ k \}y = (\underbrace{d, d , \ldots, d}_{k \text{ fois}}, a_1, a_2 , \ldots, a_n)$,

Dans l'équation ci-dessus, le vecteur $y_i\{t\_ind\}$ est précedée par le symbol ' ^ '. SISYC2 l'interpréte

$$N(x \{t\_ind, t \}) = N (y_i \{ t\_ind, t - k_i \}),$$

pour tout ≥ 1. C'est à dire que x {t_ind} aura le même nom que $y_i\{t\_ind,t-k_i\}$ Par exemple, pour le réseau de la convolution, on fera

$$x_2\{ k + 1\} = O\{2\}\ \hat{}x_2\{ k \}.$$

Si $y_j \{ t\_ind, t - k_j \} = a_j , \quad 1 \leq j \leq n ,$

alors l'exécution de l'équation ci-dessus, à la date t est équivalente à

$$a_i = P(a_1 , \ldots , a_i , \ldots , a_n) \quad .$$

- Les conditionnels

Les conditionnels de SISYC2 sont du type ci-dessous:

```
SI ( condition )
    {
        une suite d'equations ;
    } ;
```

Naturellement, dans ce cas l'instruction n'est exécutée, à la date t, que si la condition est évaluée à vrai à cette date.

Une condition est évalée si:

- elle porte sur le temps (cf. chapitre. 5)
- elle porte sur des variables numériques,
- elle est un mélange des deux cas précedents.

Tout test sur le temps est autorisé. Par contre, les tests sur les variables symboliques sont soumis à des restrictions. En effet, en SISYC2, on ne peut tester que l'égalité de deux variables symboliques. Par exemple

SI (a {t_ind} = d) ou SI (a{t_ind} ≠d ET O{ 2 } b{t_ind} = a{t_ind})

En revanche le test ci-dessous n'est pas possible.

SI (a{t_ind} < O{ 2 } b{t_ind}) ...

6.3.2. Quelques remarques

• Une variable en SISYC2 représente un vecteur dont chaque terme peut être soit un entier soit une variable symbolique (voir le programme en annexe pour la définition d'une variable).

• Les opérateurs implementés dans SISYC2 sont tous réguliers , c'est à dire,
var op d = d op var = d ,
pour tout opérateur op de l'ensemble { + , - , * , / }.

• SISYC2 est à mémoire limitée, par conséquent le nombre de vecteurs utilisées dans un même programme est borné, et il en est de même pour le temps de simulation MAXT. Le nombre de suites utilisées dans une même equation est limité aussi.

• La régle d'évaluation en SISYC2 est la suivante:

expr1 op expr2

est évaluée si et seulement si expr1 et expr2 sont tout deux des entiers. Donc si a, b et c sont trois entiers alors l'expression

a + b + c

est évaluée, et sa valeur est la somme des trois entiers a, b et c.
Si a et b sont deux entiers et c une variable symbolique , alors l'expression

a + b + c

est évaluée, sa valeur est

ent + var ;

où ent est la somme de a et b. Par contre

a + c + b

n'est pas évaluée. Nous imposons donc une contrainte d'écriture où il faut écrire les variables entières les unes à côté des autres.

• Pour savoir s'il s'agit d'une variable symbolique ou d'un entier, SISYC2 test le premier caractère de la donnée en question. Si c'est un chiffre alors il la considère comme un entier, autrement elle est considèrée comme une variable symbolique.

• SISYC2 offre presque toutes les possibilités que SISYC quant à la manipulation de vecteurs d'entiers. Cependant, ce dernier est plus rapide et consomme moins de place mémoire. En effet, SISYC2 convertit tout entier en chaîne de caractères et il la reconvertit en entier pour les besoins de calculs. D'où le temps et la place requises pour effectuer ces opérations.

Ces conversions inutiles sont dues au fait que SISYC2 n'est pas typé. Nous n'avons pas de types de données car , ceci imposerait à un vecteur d'avoir tous ses composantes du même type, ce qui est assez restrictif. Une telle restriction ne permet pas de declarer par exemple le vecteur x ci-dessous:

$x = a_1 , a_2 , a_3 , 1 , 0 \ldots$

6.4. Exemples d'utilisation de SISYC2

Commençons par le réseau de la convolution traité dans le pragraphe précédent. Le programme à donner à SISYC2 pour le calcul de la trace symbolique de réseau systolique de la convolution est ci-dessous:

```
PAR k = 2 ;
INDICE i ;
VECT y(0:k+1) , x(0:k+1) , w(0:k) ;
ENTRE( MAXT 12 , y(0), x(0) )
INITIALE ( , w(0), w(1) , w(2)  ) ;
POUR i=0, k FAIRE
       x(i+1) = O(2) ^x(i) ;
       y(i+1) = O ^y(i) + O w(i)* O(2)x(i)
            FIN ;
```

```
Les donnees du reseau sont ci-dessous:

d  d  d  y(0) y(1) y(2) y(3) y(4) y(5) y(6) ...
x(-2) x(-1) x(0) x(1) x(2) x(3) x(4) x(5) x(6) ...
w(0)
w(1)
w(2)
```

```
La trace donnee par SISYC2 est ci-dessous:

y(0):=( y(0)+( w(0)*x(0) ) ) ;
y(1):=( y(1)+( w(0)*x(1) ) ) ;
y(0):=( y(0)+( w(1)*x(-1) ) ) ;
y(2):=( y(2)+( w(0)*x(2) ) ) ;
y(1):=( y(1)+( w(1)*x(0) ) ) ;
y(0):=( y(0)+( w(2)*x(-2) ) ) ;
y(3):=( y(3)+( w(0)*x(3) ) ) ;
y(2):=( y(2)+( w(1)*x(1) ) ) ;
y(1):=( y(1)+( w(2)*x(-1) ) ) ;
y(4):=( y(4)+( w(0)*x(4) ) ) ;
y(3):=( y(3)+( w(1)*x(2) ) ) ;
y(2):=( y(2)+( w(2)*x(0) ) ) ;
y(5):=( y(5)+( w(0)*x(5) ) ) ;
y(4):=( y(4)+( w(1)*x(3) ) ) ;
y(3):=( y(3)+( w(2)*x(1) ) ) ;
y(6):=( y(6)+( w(0)*x(6) ) ) ;
y(5):=( y(5)+( w(1)*x(4) ) ) ;
y(4):=( y(4)+( w(2)*x(2) ) ) ;
y(6):=( y(6)+( w(1)*x(5) ) ) ;
y(5):=( y(5)+( w(2)*x(3) ) ) ;
```

Resultat fourni par Reduce, apres initialisation
des y(i) a zero:

```
Y(0) := W(0)*X(0)

Y(1) := W(0)*X(1)

Y(0) := W(1)*X(-1) + W(0)*X(0)

Y(2) := W(0)*X(2)

Y(1) := W(1)*X(0) + W(0)*X(1)

Y(0) := W(2)*X(-2) + W(1)*X(-1) + W(0)*X(0)

Y(3) := W(0)*X(3)

Y(2) := W(1)*X(1) + W(0)*X(2)

Y(1) := W(2)*X(-1) + W(1)*X(0) + W(0)*X(1)

Y(4) := W(0)*X(4)

Y(3) := W(1)*X(2) + W(0)*X(3)

Y(2) := W(2)*X(0) + W(1)*X(1) + W(0)*X(2)

Y(5) := W(0)*X(5)

Y(4) := W(1)*X(3) + W(0)*X(4)

Y(3) := W(2)*X(1) + W(1)*X(2) + W(0)*X(3)

Y(6) := W(0)*X(6)

Y(5) := W(1)*X(4) + W(0)*X(5)

Y(4) := W(2)*X(2) + W(1)*X(3) + W(0)*X(4)

Y(6) := W(1)*X(5) + W(0)*X(6)

Y(5) := W(2)*X(3) + W(1)*X(4) + W(0)*X(5)
```

Appliquons SISYC2 pour la validation de l'algorithme de l'exemple 3.1 (cf. chap. 3). Rappelons qu'il s'agit d'effecteur le produit de deux matrices de taille 3x3, sur un réseau linéaire. Nous avons proposé trois solutions:

- **Solution 1**: réseau sans signaux de contrôle, données lues une seule fois.

Dans ce cas, tout point de calcul correspond à une accumulation

$$c := c + a.b$$

La cellule de base est donnée dans la figure 2.2. Le programme en SISYC2, qui calcul la trace de cet algoritme est ci-dessous:

```
PAR n =3 , x = 5 ;
INDICE i ;
VECT a(1:3*n-1) , b(1:3*n-1) , c(1:3*n-1)   ;
ENTRE( MAXT 37 , a(1) , b(1) , c(1) )
POUR i=1 , 3*n-2 FAIRE
    a(i+1) = O(x) ^a(i);
    b(i+1) = O(2) ^b(i) ;
    SI( O(x)a(i) != d ET O(2)b(i) != d )
                 ( c(i+1) = O ^c(i)+O(x)a(i)*O(2)b(i););
    SI( O(x)a(i) = d OU O(2)b(i) = d )
                 ( c(i+1) = O^c(i) ; )
                FIN ;
                .

Les donnees du programme sont:

a(3,3) d d a(2,3) a(3,2) d a(1,3) a(2,2) a(3,1) d a(1,2)
a(2,1) d d a(1,1) ...

d d d d d d d d d d d d d d d b(3,1) b(2,1) b(1,1) b(3,2)
b(2,2) b(1,2) b(3,3) b(2,3) b(1,3) ...

d d d d d d d d d d d d d d d d d d d c(1,1) c(2,1) c(3,1) d
c(1,2) c(2,2) c(3,2) d c(1,3) c(2,3) c(3,3) ...
```

```
Resultat de l'execution:

c(1,1):=( c(1,1)+( a(1,1)*b(1,1) ) ) ;
c(1,1):=( c(1,1)+( a(1,2)*b(2,1) ) ) ;
c(2,1):=( c(2,1)+( a(2,1)*b(1,1) ) ) ;
c(1,1):=( c(1,1)+( a(1,3)*b(3,1) ) ) ;
c(2,1):=( c(2,1)+( a(2,2)*b(2,1) ) ) ;
c(3,1):=( c(3,1)+( a(3,1)*b(1,1) ) ) ;
c(2,1):=( c(2,1)+( a(2,3)*b(3,1) ) ) ;
c(1,2):=( c(1,2)+( a(1,1)*b(1,2) ) ) ;
c(3,1):=( c(3,1)+( a(3,2)*b(2,1) ) ) ;
c(1,2):=( c(1,2)+( a(1,2)*b(2,2) ) ) ;
c(3,1):=( c(3,1)+( a(3,3)*b(3,1) ) ) ;
c(2,2):=( c(2,2)+( a(2,1)*b(1,2) ) ) ;
c(1,2):=( c(1,2)+( a(1,3)*b(3,2) ) ) ;
c(2,2):=( c(2,2)+( a(2,2)*b(2,2) ) ) ;
c(3,2):=( c(3,2)+( a(3,1)*b(1,2) ) ) ;
c(2,2):=( c(2,2)+( a(2,3)*b(3,2) ) ) ;
c(1,3):=( c(1,3)+( a(1,1)*b(1,3) ) ) ;
c(3,2):=( c(3,2)+( a(3,2)*b(2,2) ) ) ;
c(1,3):=( c(1,3)+( a(1,2)*b(2,3) ) ) ;
c(3,2):=( c(3,2)+( a(3,3)*b(3,2) ) ) ;
c(2,3):=( c(2,3)+( a(2,1)*b(1,3) ) ) ;
c(1,3):=( c(1,3)+( a(1,3)*b(3,3) ) ) ;
c(2,3):=( c(2,3)+( a(2,2)*b(2,3) ) ) ;
c(3,3):=( c(3,3)+( a(3,1)*b(1,3) ) ) ;
c(2,3):=( c(2,3)+( a(2,3)*b(3,3) ) ) ;
c(3,3):=( c(3,3)+( a(3,2)*b(2,3) ) ) ;
c(3,3):=( c(3,3)+( a(3,3)*b(3,3) ) ) ;
```

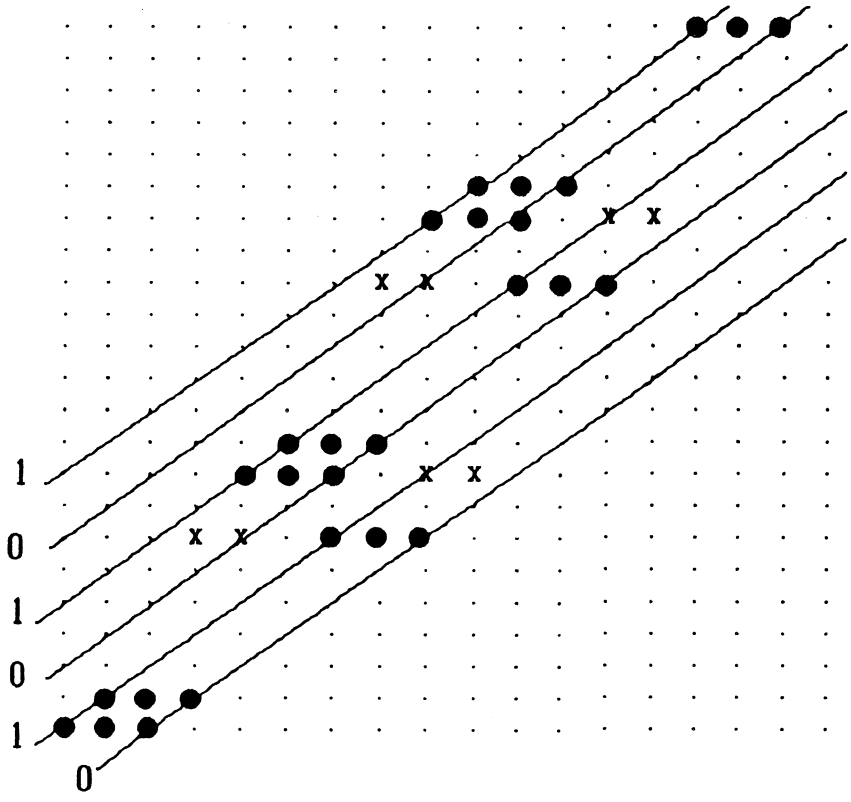
- **Solution 2**: réseau avec signaux de contrôle, données lues une seule fois.

Le diagramme de calcul correspondant à ce cas (n=3 , x=3) est donné par la figure 6.6 . Les croix représentent les points de calcul correspondant à des accumulations du type

$$c_{i,j} := c_{i,j} + a_{i',k'}.b_{k'',j''}$$

où les indices vérifient: $i \neq i'$ ou $j \neq j''$ ou $k' \neq k''$.

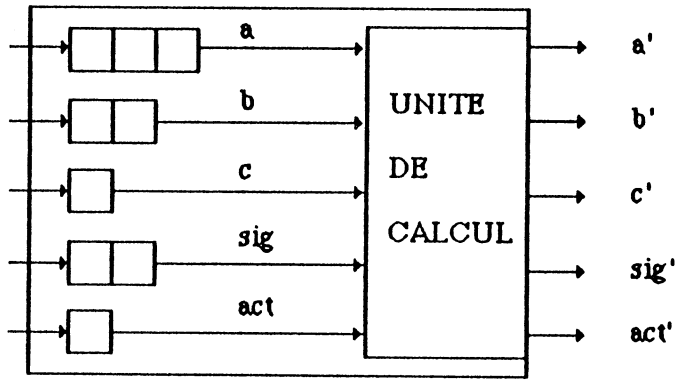
De tels calculs sont invalidés par les signaux de contrôle.



x : calcul indésirable
0 : signal de fin des accumulations
1 : signal de debut des accumulations.

Figure 6.6

La cellule de base qui compose le réseau est donnée par la figure 6.7 ci-dessous:



a' := a ; b' := b ; sig' := sig ;
Si(act ≠ vrai) c' := c ; act' := act ;
Si(a ≠ d et b ≠ d et c ≠ d et sig = vrai et act ≠ faux) act' := vrai ; c' := c + a.b ;
Si(sig = faux et act = vrai) act' = faux ; c' := c + a.b ;
Si(act = vrai et sig = d) act' := vrai ; c' := c + a.b ;

Figure 6.7

Le programme en SISYC2 qui calcul la trace de cet algorithme est ci-dessous. C'est une simple réecriture en terme de vecteurs du programme de la cellule de base (la nommination étant évidente dans ce cas).

```
PAR n =3 , x = 3 ;
INDICE i ;
VECT a(1:6*n) , b(1:6*n) , c(1:6*n),
     act(1:6*n), sign(1:6*n) , vrai , faux   ;

ENTRE( MAXT 55, a(1) , b(1) , c(1), sign(1), vrai , faux )

POUR i=1 , 6*n-1 FAIRE
    a(i+1) = O(x) ^a(i);
    b(i+1) = O(2) ^b(i) ;
    sign(i+1) = O(2) ^sign(i) ;
    SI(Oact(i)!=vrai) { c(i+1) = O^c(i); act(i+1)= O^act(i);};
    SI(O(x)a(i)!=d ET O(2)b(i)!=d ET Oc(i)!=d ET
          O(2)sign(i)=vrai ET Oact(i)!= faux) {
                        act(i+1) = ^vrai ;
                        c(i+1) = O^c(i)+O(x)a(i)*O(2)b(i);};
    SI(O(2)sign(i)=faux ET Oact(i)=vrai) {
                        act(i+1)=^faux;
                        c(i+1) = O ^c(i)+O(x)a(i)*O(2)b(i);};
    SI(Oact(i)=vrai ET O(2)sign(i)=d) {
                        act(i+1) = ^vrai ;
                        c(i+1) = O ^c(i)+O(x)a(i)*O(2)b(i);}
                FIN ;
```

Donnees du programme

```
a(3,3) d a(3,2) d  a(3,1) a(2,3) a(1,3) a(2,2) a(1,2) a(2,1)
a(1,1) ...

d d d d d d d d d b(3,1) b(2,1) b(1,1) d b(3,2) b(2,2) b(1,2)
d b(3,3) b(2,3) b(1,3) ...

d d d d d d d d d d d d d c(1,1) c(2,1) d  d  d  d  c(3,1) d
c(1,2) c(2,2) d  d  d  d  c(3,2) d c(1,3) c(2,3) d  d  d d
c(3,3) ...

d d d d d d d d d 0 d 1 d 0 d 1 d 0 d 1  ...

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
```

Afin de montrer l'interêt de la trace symbolique, on garde les éléments de la matrice B sous leur forme symbolique ($b_{i,j}$), par contre on prend pour ceux de la matrice A des coefficients pôlynomiaux. Ainsi, la matrice C cherchée est:

$$C = \begin{pmatrix} x^2+1 & x & x+1 \\ 1 & x & x+2 \\ x^2+2 & 0 & x \end{pmatrix} \begin{pmatrix} b(1,1) & b(1,2) & b(1,3) \\ b(2,1) & b(2,2) & b(2,3) \\ b(3,1) & b(3,2) & b(3,3) \end{pmatrix}$$

Pour vérifier que l'algorithme systolique effectue cette opération, il suffit de donner au système Reduce le programme ci-dessous, qui est composé des initialisations des variables et de la trace fournie par SISYC2.

```
a(1,1) := x**2+1 ;
a(1,2) := x ;
a(1,3) := x+1;
a(2,1) := 1;
a(2,2) := x;
a(2,3) := x+2;
a(3,1) := x**2+2;
a(3,2) := 0;
a(3,3) := x;
```

```
c(1,1):=( c(1,1)+( a(1,1)*b(1,1) ) ) ;
c(1,1):=( c(1,1)+( a(1,2)*b(2,1) ) ) ;
c(2,1):=( c(2,1)+( a(2,1)*b(1,1) ) ) ;
c(1,1):=( c(1,1)+( a(1,3)*b(3,1) ) ) ;
c(2,1):=( c(2,1)+( a(2,2)*b(2,1) ) ) ;
c(3,1):=( c(3,1)+( a(3,1)*b(1,1) ) ) ;
c(2,1):=( c(2,1)+( a(2,3)*b(3,1) ) ) ;
c(1,2):=( c(1,2)+( a(1,1)*b(1,2) ) ) ;
c(3,1):=( c(3,1)+( a(3,2)*b(2,1) ) ) ;
c(1,2):=( c(1,2)+( a(1,2)*b(2,2) ) ) ;
c(3,1):=( c(3,1)+( a(3,3)*b(3,1) ) ) ;
c(2,2):=( c(2,2)+( a(2,1)*b(1,2) ) ) ;
c(1,2):=( c(1,2)+( a(1,3)*b(3,2) ) ) ;
c(2,2):=( c(2,2)+( a(2,2)*b(2,2) ) ) ;
c(3,2):=( c(3,2)+( a(3,1)*b(1,2) ) ) ;
c(2,2):=( c(2,2)+( a(2,3)*b(3,2) ) ) ;
c(1,3):=( c(1,3)+( a(1,1)*b(1,3) ) ) ;
c(3,2):=( c(3,2)+( a(3,2)*b(2,2) ) ) ;
c(1,3):=( c(1,3)+( a(1,2)*b(2,3) ) ) ;
c(3,2):=( c(3,2)+( a(3,3)*b(3,2) ) ) ;
c(2,3):=( c(2,3)+( a(2,1)*b(1,3) ) ) ;
c(1,3):=( c(1,3)+( a(1,3)*b(3,3) ) ) ;
c(2,3):=( c(2,3)+( a(2,2)*b(2,3) ) ) ;
c(3,3):=( c(3,3)+( a(3,1)*b(1,3) ) ) ;
c(2,3):=( c(2,3)+( a(2,3)*b(3,3) ) ) ;
c(3,3):=( c(3,3)+( a(3,2)*b(2,3) ) ) ;
c(3,3):=( c(3,3)+( a(3,3)*b(3,3) ) ) ;
end ;
```

Resultat fourni par Reduce

$$C(1,1) := B(1,1)*(X^2 + 1)$$

$$C(1,1) := B(2,1)*X + B(1,1)*X^2 + B(1,1)$$

$$C(2,1) := B(1,1)$$

$$C(1,1) := B(3,1)*X + B(3,1) + B(2,1)*X + B(1,1)*X^2 + B(1,1)$$

$$C(2,1) := B(2,1)*X + B(1,1)$$

$$C(3,1) := B(1,1)*(X^2 + 2)$$

$$C(2,1) := B(3,1)*X + 2*B(3,1) + B(2,1)*X + B(1,1)$$

$$C(1,2) := B(1,2)*(X^2 + 1)$$

$$C(3,1) := B(1,1)*(X^2 + 2)$$

$$C(1,2) := B(2,2)*X + B(1,2)*X^2 + B(1,2)$$

$$C(3,1) := B(3,1)*X + B(1,1)*X^2 + 2*B(1,1)$$

$$C(2,2) := B(1,2)$$

$$C(1,2) := B(3,2)*X + B(3,2) + B(2,2)*X + B(1,2)*X^2 + B(1,2)$$

$$C(2,2) := B(2,2)*X + B(1,2)$$

$$C(3,2) := B(1,2)*(X^2 + 2)$$

$$C(2,2) := B(3,2)*X + 2*B(3,2) + B(2,2)*X + B(1,2)$$

$$C(1,3) := B(1,3)*(X^2 + 1)$$

$$C(3,2) := B(1,2)*(X^2 + 2)$$

$$C(1,3) := B(2,3)*X + B(1,3)*X^2 + B(1,3)$$

$$C(3,2) := B(3,2)*X + B(1,2)*X^2 + 2*B(1,2)$$

$$C(2,3) := B(1,3)$$

$$C(1,3) := B(3,3)*X + B(3,3) + B(2,3)*X + B(1,3)*X^2 + B(1,3)$$

$$C(2,3) := B(2,3)*X + B(1,3)$$

$$C(3,3) := B(1,3)*(X^2 + 2)$$

$$C(2,3) := B(3,3)*X + 2*B(3,3) + B(2,3)*X + B(1,3)$$

$$C(3,3) := B(1,3)*(X^2 + 2)$$

$$C(3,3) := B(3,3)*X + B(1,3)*X^2 + 2*B(1,3)$$

CONCLUSION

La formulation combinatoire présentée dans le chapitre 2, nous a permis de concevoir des algorithmes pour le produit matriciel sur les réseaux linéaires. Cette formulation varie selon les caractéristiques des réseaux considérés (sens de circulation des données, tailles des buffers, présence ou non de signaux de contrôle). En outre, elle nous a permis de concevoir des algorithmes dans les deux cas ci-dessous:

- données lues plusieurs fois par le réseau
- nombre variable de canaux d'entrée / sorties du réseau

A notre connaissance, ces deux difficiles problèmes n'ont pas été abordés par d'autres auteurs. D'autre part, tous les algorithmes, pour le produit matriciel, proposés par Varman, Fussell et Ramakrishnan, peuvent être retrouvés de manière évidente à l'aide de cette formulation.

Dans le cas de réseaux bidirectionnels où les données sont lues plusieurs fois, cette formulation est identique à la formulation combinatoire proposée par Melkemi et Tchuente, pour les réseaux rectangulaires.

Nous n'avons étudié l'optimalité de ces algorithmes que dans un cas simple. Le cas général est difficile, cependant il peut être traité à l'aide de cette formulation combinatoire.

La technique de séquentialisation, que nous avons présenté de manière très simplifiée, permet de retrouver l'un des algorithmes séquentiel implémenté par un réseau systolique donné. Il suffit donc, d'établir l'équivalence entre l'algorithme obtenu et l'algorithme pour lequel le réseau a été conçu, pour pouvoir se prononcer sur la validité de l'algorithme systolique consideré.

Le logiel SISYC2, basé sur cette méthode, prend en entrée une description d'un algorithme systolique et calcul la trace de ce dernier. Couplé à un module de calcul formel (Reduce par exemple), il peut être consideré comme un simulateur formel (ou symbolique) efficace des algorithmes systoliques. Ce logiciel et le langage de simulation numérique, SISYC, sont implementés à l'IRISA (Rennes) chez l'équipe de P. Quinton.

Pour cette étude, nous nous sommes limités au cas des réseaux systoliques décrits par des SERT, car dans ce cas la trace obtenue est directement interprétée par Reduce, et les expressions formelles obtenues sont celles effectuées par le réseau consideré. Dans le cas général, il faut modifier la trace en introduisant des variables de sauvgarde, afin d'obtenir les vrais calculs du réseau.

REFERENCES

[AhU 77] A.V. Aho & J.D. Ulman, " Priciples of compiler Design", Addition Wesley, Reading, Mass 1977.

[AAG 86] M. Annaratone, E. Arnold, T. Gross, H.T. Kung, M.S. Lam, O. Menzilciogla, K. Sarocky, J.A. Webb, "Warp architecture and implementation", the 13th Annual International symposium on computer architecture, june 1986, Tokyo, Jappan.

[AFE 88] E. Audureau, L. Frinas del cerro, P. Enjalbert " Théorie de la programmation et logique temporelle, validation des alghorithmes paralléles", TSI vol.7 n°2 1988

[BeE 87] M.P. Bekakos, D.J. Evans, " A rotating and folding algorithm using a two dimentional systolic communication geometry ", Parallel computing 4 (1987) 221-228, North Holland.

[BeT 88] A. Benaini, M. Tchuente, " Séquentialisation d'algorithmes systoliques ", R.R 718-M-, IMAG, TIM3, Avril 1988.

[Ber 86] J.L. Bergerand, " Lustre un langage declaratif pour le temps reel" , Thèse de l'Universite de Rennes I , Janvier 1986.

[BPS 88] J.C. Bermond, C. Peyrat, I. Sakho, M. Tchuente, "Parallelization of Gauss elimination algorithm on systolic arrays",

[BBK 81] A. Bojanczyk R.P. Brent H.T. Kung, " Numerically stable solution of dense systems of linear equations using mesh-connected processors", Tech. Rep, Carnegie Mellon University, 1981.

[Bro 84] S.D. Brookes, " Reasoning about synchronous systems ", CMU-CS-84-145, Dept. of computer science, Carnegie Mellon, Univ. Pittsburg, March 1984.

[BHM 85] D.S. Broomhead, J.G. Harp, J. Mcwhiter, K.J. Palmer, J.G.B. Robert, " Pratical comparaison of the systolic and wavefront array", ICASSP 85.

[BCC 87] B. Bruegge, C. Chang, R. Cohn, T. Gross, M. Lam, P. Lien, A. Noaman, D. Yam, "Programming warp", COMPCON Spring 87.

[CaS 83] P.R. Cappello, K. Steiglitz, " Unifying VLSI array design with geometric transformations" , IEEE Transaction on computer, 1983.

[CeM 82] M.C. Chen & C.A. Mead " Concurrent Algorithms as space-time recursion equations", in Kung S.Y. ed , Proc. of USC Workshop on VLSI & modern signal processing, Nov 1982.

[ChS 87] K.H. Cheng, S. Sahni, " VLSI systems for band matrix multiplication", Parallel computing 4 (1987) 239-258 North-Holland.

[ComR 86] P. Common & Y. Robert "A systolic array for computing BA^{-1}", R.R, TIM3, IMAG, Grenoble n° 591, Janv 1986.

[CosR 86] M. Cosnard & Y. Robert " Systolic Givens factorization of dense rectangular matrices", RR, TIM3, IMAG, Grenoble, n°590, Janv 1986.

[CDM 86] M. Cosnard, M. Daoudi, J.M. Muller, Y. Rober, " On parallel and systolic Givens factorization of dense matrices ",
M. Cosnard, Y. Robert, P. Quinton , M. Tchuente, éditeurs, Proc. Parallel algorithms and architectures, Luminy, France, 1986, North-Holland.

[CRT 86] M. Cosnard, Y. Robert, M. Tchuente, "Matching parallel algorithms with architectures: a case study", IFIP working conf. on highly parallel computers for numerical and signal processing applications, IFIP W.G. 10-3, Nice, France , Mars 1986.

[CoC 87] P. Cousot, R. Cousot " Principes des méthodes de preuve de propriétés d'invariance et de fatalité des programmes paralléles", dans preuve des programmes paralléles, 129-148.

[CuY 85] K. Culik, S. Yu, " Translation of systolic algorithms between systems of different topology ", CS-85-06, May 1985, Dept. of computer science, Univ. of Waterloo, Canada.

[DoV 87] K. Doshi, P. Varman , " A modular systolic architecture for image convolutions" , ACM 1987

[Fos 81] M.J. Foster, " Syntax directed verfication of circuit functions, in VLSI systems and computations ", H.T. Kung , B spoull and G steele eds, comp. science press, Rockville, MD , 1981.

[Gau 84] T. Gautier " Conception d'un langage flot de données pour le temps réel", Thèse de l'université de Rennes, decembre 1984.

[GeK 81] W.M. Gentleman, H.T. Kung, " Matrix triangularisation by systolic arrays", Proc SPIE 298, Real-time Signal Processing IV, San Diego, California, 1981.

[GuB 86] P. Le Guernic, A. Benveniste, " Real-time, synchronous, data-flow programing: The langage SIGNAL and it's mathematical semantics ", publication interne n°298, Juin 1986, irisa, Rennes.

[GuL 82] L.J. Guibas, F.M. Liang, "Systolic stacks queues and counters", Proc. 1982, conf. of advanced reserch in VLSI, MIT, Boston, USA (1982), 155-164.

[HeI 83] D.E. Heller, I.C. Ipsen, " Systolic networks for orthogonal decompositions ", Siam J. on scientific and statical computing 4 (1983) 261-269.

[HuA 84] K.H. Huang, Abraham, " Algorithm based fault-tolerance for matrix operations ", IEEE Trans. on computer, June 1984.

[IKP 86] O.H. Ibarra, S.M. Kim, M.A. Palis, " Designing systolic algorithms using sequential machines ", IEEE Transactions on computers, vol. C-35, n° 6, June 86.

[IbK 84] O.H. Ibarra, S. Kim, " Characterization and computational complexity of systolic treillis automata ", Theoretical computer science 29 (1984), 123-153, North-Holland.

[Joh 74] S.C. Johnson " Yacc: yet another compiler compiler", computer science T.R n°32, 1975, Bell labs, Murray Hill, NJO 1974.

[JoG 87] B. Joinnault & P. Gachet , " conception d'algorithmes et d'architectures systoliques", Thèse de l' Université de Rennes I, sept 1987.

[JoA 85] J.Y. Jou, Abraham, " Fault-tolerant matrix arithmetique and signal processing on highty concurent computing structure, 1985

[Kat 86] E. Katona, " A lattice model for cellular (systolic) algorithms ", Parallel computing 3 (1986) 251-258, North Holland.

[Kun 80] H.T. Kung, " Special-purpose devices for signal and image processing: an opportinity in VLSI ", Proc. SPIE, vol 241, Real-time signal and processing 111, 1980

[Kun 82] H.T. Kung, " Why systolic architactures ? ", IEEE computer 15,1 (1982), p. 37-46.

[Kun 86] H.T. Kung, " Memory requirements for balanced computer architectures ", Proc. International sumposuim computer architecture, Tokyo 1986.

[KuL 84] H.T. Kung & M.S. Lam, " Wafer scal integration and two level pipelined implementation systolics arrays " , J. of parallel and distributed computing, 1984.

[KuL 78] H.T. Kung, C.E. Leiserson, " Systolic arrays (for VLSI) ", In Sparse Matrix Proc. 1978, p. 256-282, Society for industrial and applied Mathematics, 1978.

[KuL 83] H.T. Kung, W.T. Lin, " An algebra for VLSI algorithm design ", CMU-CS- 84-100, Dept. of computer science, Carnegie Mellon Univ. Pittsburg, Pennsylvania 15213, April 1983.

[KuS 85] H.T. Kung and S.W. Song, " A systolic 2-D convolution chip ", T.R. CMS-CS-81-110, Carnegie-Mellon Univ. USA, 1985.

[KAG 82] S.Y. Kung, K.S. Arun, R.J. Gal-ezer, D.V. Rao, " Wavefront array processor: langage, architecture and application ", IEEE T.C, vol. C-31, n° 11, Nov, 1982.

[LaM 85] M. Lam & J. Mostow, " A transformational model of VLSI systolic design", IEEE computer, Feb 1985.

[LeL 85] T. Leighton, C.E. Leiserson, " Wafer scale integration of systolic arrays ", IEEE Transactions on computers, vol. C-34, n° 5, May 1985.

[Lei 79] C. Leiserson " Systolic priority queues", dept of computer science, Carnegie mellon Univ. Pittsburg, Pennsylvania, 15213, April 1979.

[LeS 83] C. Leiserson & J. Saxe " Optimizing synchronous systems", J. VLSI and computer systems 1, 1983.

[LeS 74] M.E. Lesk & E. Schmidt, " Lex- a lexical analyser generator", Bell labs, Murray Hill, New jersy 07974.

[LiW 84] G.J. Li, B.W. Wah, " The design of optimal systolic algorithms ", IEEE Trans. on computer, 1984.

[Mel 85a] R.G. Melhem, " Formal analysis of systolic system for finite element stiffness matrices", J. of computer and system sciences, vol. 31.1, August 1985.

[Mel 85b] R.G. Melhem, "A langage for the simulation of systolic architectures", Proc. IEEE 12th. int. sym. on computer architectures, Boston, MA, USA, 1985.

[Mel 83] R.G. Melhem, "Simulation of systolic networks whith syntax directed solver for systems of sequence equations", T.R., ICMA, 83.58, Institute of computational mathematics and it's applications, Univ. Pittsburg, 1983.

[MeR 84] R.G. Melhem & W. Reinboldt, " A mathemetical model for the verification of systolic networks", SIAM. J. on computing, August 1984.

[MeT 88] L. Melkemi & M. Tchuente, " Complexity of matrix product on a class of orthogonally systolic arrays ", IEEE trans. on Computer 1988.

[MeT 86] L. Melkemi & M. Tchuente, " I/O Trade-offs matrix product on programmable arrays". Proc. Parallel Computing 85. M. Feilmein, G. Joubert, V. Schendel eds. Elsevier Publishers B.V. 187 -192 (1986).

[MeT 85] L. Melkemi & M. Tchuente, " Programmation du produit matriciel sur un réseau systolique rectangulaire", T.S.I.4. 1985

[MiC 81] J. Misra, K.M. Chandy, " Proofs of networks of processes ", IEEE Transactions on software engineering, vol SE 7, n° 4, July 1981.

[MiW 84] W.L. Miranker, A. Winkler, " Space-time representation of systolic computational structures ", Computing 32 (1984), p. 93-114.

[Mol 82] D.I. Moldovan, " On the analysis and synthesis of VLSI algorithm", IEEE Trans. computer, vol C-31,, Nov 1982.

[Mol 83] D.I. Moldovan, J.A. Fortes, " Partitioning of algorithms for fixed size VLSI architectures ", Technical report PPP 83-5, Dept. of electrical systems, Univ. of Southern California, Los Angelos.

[Mol 85] F. Moller, "A survey of systolic systems for solving the algebraic path problem", Rapport CS- 85-22, computer science dept. Univ. of Waterloo, Canada, 1985.

[Mon 85] C. Mongenet, " Une méthode de concepetion d'algorithmes systoliques, résultats théoriques et réalisation ", Thèse de l'INPL, Nancy 1985.

[Mun 83] T. Muntean, "Introducion a occam, langage parallele issu de CSP pour la programmation des systemes de transinateurs", RR LGI Grenoble 430, decembre 1983.

[Mur 71] Y. Muraoka, "Parallelism exposure and exploitation in programs", Ph.D, dept. computer science, Univ of Illinois, Urbana, feb, 1971.

[PrL 88] D.K. Probst, M.F. Li " Abstract specification of synchronous data types for VLSI and proving the correctness of systolic network implementation", IEEE transaction on computer, vol 37, n°6, june 1988.

[Qui 84] P. Quinton, " Automatic synthesis of systolic array from uniform recurrent equations", Proc, 11th Annual symp. on computer architecture, 1984.

[Qui 83] P. Quinton, " The systematic design of systolic arrays, R.R , 193 IRISA 1983

[QuG 84] P. Quinton & P. Gachet , " DIASTOL user's manual- preliminary version", T.R 233 Publication interne IRISA, Rennes, France , Aout 1984.

[QuR 88] P. Quinton & Y. Robert , " Algorithmes et architectures systoliques ", à paraître , Maccon éditeur, 1988.

[RaF 83] I.V. Ramakrishnan, D.S. Fussell, A. Silberschatz, " On mapping homogeneous graphs on a linear array-processor model ", IEEE 1983.

[Rob 86] Y. Robert, " Algorithmes et architectures systoliques", polycopié, INPG, Octobre 1986.

[RoT 85] Y. Robert & M. Tchuente " Résolution systoliques de systèmes linéaires denses", RAIRO Modélisation et analyse numérique, 19,2, 1985, 315-326.

[Rot 86] G. Rothe, " On the connection between hexagonal and unidirectional rectangular systolic arrays ", Bricht 86-71, May 1986.

[Sak 87] I. Sakho, " Synthèse et simulation d'algorithmes systoliques", Thèse de 3^{eme} cycle, INPG Grenoble, Avril 1987.

[Sam 82] A. Sameh, " Solving the linear least squares problem on a linear array of processors,", Proc. Purdue Workshop on algorithmically-specialized computer organisations, W. Lafayette, Indiana, Septembre 1982.

[Sch 84] U. Schendel, " Introduction to numerical methods for parallel computers", Ellis Horwood Series, J. Wiley & Sons, New York, 1984.

[Sch 87] R. Schreiber, " Systolic linear algebra machines: a survey ", Technical report, n°87-18, June 1987, Rensselaer Polytechnic Institute, Troy, New York 12180-3590.

[SuM 87] R. Suros, E. Montagne, " Optimizing systolic network by fitting diagonals ", Parallel computing 4 (1987), 167-174, North Holland.

[Tch 88] M. Tchuente, " Parallel computation on regular arrays ", Manchester University Press (to appear).

[TcB 88] M. Tchuente & A. Benaini, " Produit matriciel sur un réseau linéaire", RR 709 -I- , TIM3, IMAG, Grenoble , France, Mars 1988.

[TcC 88] M. Tchuente & M. Cosnard, " Systolisation par analyse descendante", RR TIM3, IMAG, Grenoble , France

[TRB 88] M. Tchuente, F. Robert, A. Benaini, " SISYC simulateur numérique d'algorithmes systoliques", RR 710 - I- ,TIM3, IMAG, Grenoble, France, Mars 1988

[Tid 84] E. Tiden, " Verification of systolic arrays, a case study", TR TRITA NA- 8403, Dept of numerical analysis and computer science, Royal Institue of technology 10044 Stockholm, Sweden 1984.

[VaF 82] P. J. Varman and D. Fussell , " Fault tolerant wafer scale architecture for VLSI " , Proc. 9^{th} ann. Symp. on computer architecture, 1982.

[VaR 84] P.J. Varman & I.V. Ramakrishnan, " Modular matrix multiplication on linear array", IEEE Trans. on Computer , vol C-33, Nov 1984.

[VaR 85] P.J. Varman & I.V. Ramakrishnan, " An optimal Family of Matrix Multiplication Algorithms on Linear Arrays". ACM Parallel Processing 1985.

[WoD 87] Y. Wong, J.M. Delosme, " Transformation of broadcasting into pipelining ", R.R, Yaleu / DCS/ RR - 544, June 1987.

ANNEXE 1

GRAMMAIRE DE SISYC2

```
%start prog
%token PAR INDICE VECT POUR FAIRE FIN ENTRE
%token INITIALE MAXT MAX MIN SI OU ET
%token Virgul P_virgul D_pt Egal Plus Moin
%token Mult Div Scalaire O_acolade F_acolade
%token Infegal Inf Diff Sup Sup_egal
%token O_parenthese F_parenthese O_crochet
%token F_crochet Scalaire Chap
%token O Z T V
%token Identificateur Reel_pos Ent_pos
%left Plus Moin
%left Mult Div
%left Scalaire MIN MAX
%left O Z T

%%
prog            : declaration entre initiale corp Scalaire
                ;
declaration     : decl_par decl_indice decl_vect
                ;
decl_par        : PAR list_par P_virgul
                |
                ;
list_par        : list_par Virgul un_par
                | un_par
                ;
un_par          : Identificateur Egal Ent_pos
                ;
decl_indice     : INDICE list_indice P_virgul
                |
                ;
list_indice     : list_indice Virgul Identificateur
                | Identificateur
                ;
decl_vect       : VECT list_vect P_virgul
                ;
list_vect       : list_vect Virgul dim_vect
                | dim_vect
                ;
dim_vect        : Identificateur O_acolade list_ind F_acolade
                | Identificateur
                ;
list_ind        : list_ind Virgul rang
                | rang
                ;
rang            : i_expr D_pt i_expr
                ;
i_expr          : i_expr Plus i_term
                | i_expr Moin i_term
```

```
                | i_term
                ;
i_term          : i_term Mult i_facteur
                | i_term Div i_facteur
                | i_facteur
                ;
i_facteur       : fact_simple
                | Moin fact_simple
                | MIN O_parenthese i_expr Virgul i_expr F_parenthese
                | MAX O_parenthese i_expr Virgul i_expr F_parenthese
                | O_parenthese i_expr F_parenthese
                ;
fact_simple     : Identificateur
                | Ent_pos
                ;
corp            : list_inst P_virgul
                ;
list_inst       : list_inst P_virgul instruction
                | instruction
                ;
instruction     : equation
                | boucle_equation
                | test
                ;
test            : SI O_parenthese cond F_parenthese O_acolade
                  block F_acolade
                ;
block           : block vect_spec Egal vect_expr P_virgul
                | block test P_virgul
                |
                ;
cond            : cond ET cond
                | cond OU cond
                | O_parenthese cond F_parenthese
                | cond1
                ;
cond1           : vect_expr Egal vect_expr
                | vect_expr Diff vect_expr
                | V Infegal i_expr
                | V Inf i_expr
                | V Egal i_expr
                | V Diff i_expr
                | V Sup i_expr
                | V Sup_egal i_expr
                | i_expr Sup V
                | i_expr Sup_egal V
                | i_expr Infegal V
                | i_expr Inf V
                | i_expr Egal V
                | i_expr Diff V
```

```
                | i_expr Inf V Inf i_expr
                ;
boucle_equation : POUR boucle_spec list_inst FIN
                ;
boucle_spec     : Identificateur Egal i_expr Virgul i_expr FAIRE
                ;
equation        : vect_spec Egal vect_expr
                ;
vect_spec       : Identificateur O_acolade l_indice F_acolade
                | Identificateur
                ;
l_indice        : l_indice Virgul i_expr
                | i_expr
                ;
vect_expr       : vect_expr Plus vect_term
                | vect_expr Moin vect_term
                | vect_term
                ;
vect_term       : vect_term Mult v_facteur
                | vect_term Div v_facteur
                | v_facteur
                ;
v_facteur       : Reel_pos Scalaire vect_fact
                | vect_fact
                ;
vect_fact       : vect_spec
                | Chap vect_spec
                | op_simple vect_fact
                | O_crochet vect_expr F_crochet
                ;
op_simple       : O argument
                | Z argument
                | T argument
                ;
argument        : O_acolade i_expr F_acolade
                |
                ;
entre           : ENTRE O_parenthese list_entre F_parenthese
                ;
list_entre      : list_entre Virgul spec_entre
                | MAXT Ent_pos
                ;
spec_entre      : vect_spec
                | POUR vect_pour spec_entre
                ;
vect_pour       : Identificateur Egal i_expr Virgul i_expr
                ;
initiale        : INITIALE O_parenthese list_initiale
                  F_parenthese P_virgul
                |
```

```
                ;
list_initiale   : list_initiale Virgul spec_initiale
                |
                ;
spec_initiale   : vect_spec
                | POUR vect_pour spec_initiale
                ;

%%
# include "lex.yy.c"
```

ANNEXE 2

PROGRAMME PRINCIPAL DE SISYC2

```
# include <stdio.h>

      /*       les constantes du programme          */

# define tail_prog  600
# define N_regle 97
    /* nombre de regles dans la grammaire                */
# define N_dums 29
    /* nombre de regles sans actions             */
# define N_symbol 40
    /* taille de la table des symboles           */
# define N_borne 30
    /* taille de la table des bornes             */
# define N_vect 300
    /* nombre maximal des sequences utilisees    */
# define maxtime 70
    /* borne sup du temp de simulation           */
# define tail_pile 40
    /* longeur de la pile de travail             */
# define par_eq  10
    /* longeur max d'une equation                */

    /*   les types de donnees                    */

typedef struct  var {  char  idf[6] ;
                       int   indice[4] ;
                       int   taille ; } ;
    /*  definition d'une variable */

typedef struct expr {  int    op ;
                       struct var variable ;
                       struct expr *droite ;
                       struct expr *gauche ; } ;
   /*  definition d'une expression formelle     */

typedef struct triplet { char action ;
                         int val ;
                         int top ; } ;

typedef struct typ_symb { char type ;
                          int entry1 ;
                          int entry2 ; } ;

int deborde[6] = { tail_prog, tail_pile, N_symbol,
                   N_borne,  N_vect, maxtime + 1};

FILE *fopen() ,*fp ;
      /*  fichier contenant le code du programme */
```

```
struct triplet prog[tail_prog];
     /*  tableau pour le stockage */
     /*  du code du programme     */

int location ;   /* indice courrant de prog  */

/* contient le nombre de mot de chaque regle
   de la grammaire */

int ajust[N_regle]  =
{
-4, -2, -2,  1, -2,  0, -2, -2,  1, -2,  0, -2, -2,  0, -3,  0, -2,  0,
-2, -2, -2,  0, -2, -2,  0,  0, -1, -5, -5, -2,  0,  0, -1, -2,  0,  0,
 0,  0, -6, -4, -2,  1, -2, -2, -2,  0, -2, -2, -2, -2, -2, -2, -2, -2,
 0, -2, -2, -2, -2, -2, -4, -3, -5, -2, -3,  0, -2,  0, -2, -2,  0, -2,
-2,  0, -2,  0,  0, -1, -1, -2, -1, -1, -1, -2,  1, -3, -2, -1,  0, -2,
-4, -4, 1, -2, 1, 0, -2
} ;

/* dums[] contient les numeros des regles sans action */

int dums[N_dums] =
 {
3, 4, 5, 6, 8, 9, 12, 13, 14, 22, 25, 26, 32, 34, 35, 36, 37, 38, 41,
42, 46, 71, 74, 76, 87, 92,93, 94, 95
 } ;

int f_etape[5] = { 0, 2, 86, 33, 1} ;

int stack[tail_pile];          /* pile de travail entiere  */
struct expr fstack[tail_pile]; /* pile de travail formelle */
int condition[tail_pile] ;

/* table des symboles :
   type V ->vect, P -> parametre, I -> indice */
struct { char type;
         int entry1;
         int entry2;} sym_tab[N_symbol];

int inf_borne[N_borne]; /* borne inf de la tab des vecteurs      */
int sup_borne[N_borne]; /* borne sup de la tab des vecteurs      */
int ran_ptr = -1 ;      /* pointeur vers la table des symboles   */
int compteur = 1 ;      /* nbr de vect a lire du fichier de donnees */
int d_flag ;            /* test le type des entrees                 */
int initial[maxtime] ;   /* test si le vecteur doit etre initialiser */
int  nb ;

struct var vect_stock[N_vect][maxtime+1];
    /* stockage des valeurs formelles */
int seq_ptr = 0 ;    /* pointeur vers vecteurs       */
```

```
int tail_vect ;       /*  taille actuelle du vecteur  */
int fin_etape ;       /* controle la fin de 1 etape   */
int booleen ;
struct expr vide ;    /* expression vide              */
struct var v[maxtime] ;  /* contient la variable d'initi */
struct var  Qc , d , un , zero , varvide ;

/*------------------      programme main      ----------------------*/

main()
{
 char action ;
 int val, top= -1, i, j, k, vue, etape, seq_ptr=0 ;

 for(j=0 ; j<100 ; j++ ) initial[j] = 1 ;
 d.taille = 0 ; zero.taille = 0 ;
 un.taille =0 ;  varvide.taille = 0 ;
 d.idf[0] = 'd' ; zero.idf[0] = '0' ;
 un.idf[0] = '1' ;varvide.idf[0] = ' ' ;

 for(j=1 ; j<6 ; j++) { d.idf[j]= ' ' ;
                        zero.idf[j]= ' ' ;
                        un.idf[j]= ' ' ;
                        varvide.idf[j] = ' ' ; }
 fp = fopen("F_prog_code","r");
 seq_ptr = 0;
 Qc = d ;
 for ( j = 0 ; j < 3 ; j++) {
                        sym_tab[j].type = 'V';
                        sym_tab[j].entry1 = 0 ;
                        sym_tab[j].entry2 = -1;
                        seq_ptr++ ;
                      }
 for ( j = 3 ; j < N_symbol ; j++ )
                        sym_tab[j].type = ' ' ;
 for ( j = 1 ; j < maxtime ; j++) {
                        vect_stock[1][j].idf[0]= '0' ;
                        vect_stock[2][j].idf[0]= '1' ;
                        vect_stock[0][j].idf[0]='d' ;
                        vect_stock[1][j].taille = 0;
                        vect_stock[2][j].taille = 0;
                        vect_stock[0][j].taille = 0;
                        for(i=1; i<6; i++) for( k=0; k<3; k++ )
                                 vect_stock[k][j].idf[i]=' ' ;
                      }
for(j=0; j<tail_pile ; j++) condition[j] = 1 ;

 /*----------------------fin d' initialisation------------------*/

for ( etape=1 ; etape<= 4 ; ++etape)
```

```
 {
   location = 0 ;
   fin_etape = 1;
   while( fin_etape )
   { fscanf(fp,"%c",&action);
     fscanf(fp,"%d\n",&val);
     switch (action)
       {
         case 'R' : {
                      vue = 1; j= 0 ;          /* reduction    */
                      while ( vue && j<N_dums )
                      if( val == dums[j++] ) vue = 0;
                      if( vue ) stock_triple(action,val,top);
                      top += ajust[val-1] ;
                      if( val == f_etape[etape] ) fin_etape= 0;
                      break;
                    }
         case 'C' : {
                      ++top;                  /* shift entier    */
                      stock_triple(action,val,top);  /* shift idf */
                      break;
                    }
         case 'S' : {
                      ++top;
                      break;                   /* shift         */
                    }
         case 'A' : {
                      ++top;                   /* accept        */
                      stock_triple(action,val,top);
                      break;
                    }
       }
   }
location = -1;
fin_etape = 1;

/*--------------   Execution des routines  semantiques  ----- */

while ( fin_etape )
  {
   ++location;
   val = prog[location].val;
   top = prog[location].top;
   switch(prog[location].action)
     {
      case 'C' : {
                   stack[top] = val ; break ;
                 }
      case 'R' : {
                   semantic(val,top) ; break ;
```

```
                    }
      case 'A' : fin_etape = 0 ;
    }
 }
}
fclose(fp) ;
}
/*--------------       fin du programme main     ----------------*/

/*          routine qui stocke un triplet dans prog[]          */

stock_triple(a,v,t) char a; int v,t;
{
 prog[location].action = a;
 prog[location].val = v;
 prog[location].top = t;
 ++location ;
 return(0);
}

/*---------------  les routines semantiques       ------------  */

int declare = 1 , causal = 0 , TIME = 1 ,t , i = 0 ;

semantic(regle,top) int regle, top ;
{
   int to, t2, lecture , j ;
   struct expr qt[par_eq], qg[par_eq] ;

   if( causal )
         {
           switch(regle)
             {
               case 79 :{ --causal; return(0) ; }
               case 81 :
               case 82 :
               case 83 :{ causal++ ; return(0) ; }
               default : return(0);
             }
         }
   switch(regle)
   {
    case 1 : {
               fin_etape=0; return(0) ; /*  fin de l etape 4      */
             }
    case 2 : {
               declare= 0;
               fin_etape = 0;
               return(0);
             }
```

```
    case 7 : {
                t2 = stack[top - 2];
                if(t2>N_symbol) erreur(13) ;
                if( ( t2 > 2 ) && ( sym_tab[t2].type !=' ' ) )
                                                    erreur(11);
                sym_tab[t2].entry1 = stack[top];
                sym_tab[t2].type = 'P';
                return(0);
             }
    case 10 :
    case 11 : {
                to = stack[top];
                if( ( to > 2 ) && ( sym_tab[to].type != ' ' ) )
                                                    erreur(11);
                sym_tab[to].entry1 = 0;
                sym_tab[to].entry2 = 0;
                sym_tab[to].type = 'I';
                return(0);
             }
    case 15 : {
                if(++ran_ptr>N_borne) erreur(14);
                inf_borne[ran_ptr] = 0;
                sup_borne[ran_ptr] = 0;
                seq_ptr = seq_ptr + stack[top-1];
                if(seq_ptr>N_vect) erreur(15);
                return(0);
             }
    case 16 : {
                to = stack[top];
                if( ( to > 2 ) && ( sym_tab[to].type != ' ' ) )
                                                    erreur(11);
                sym_tab[to].type = 'V';
                sym_tab[to].entry1 = seq_ptr;
                sym_tab[to].entry2 = -1;
                if(++seq_ptr>N_vect) erreur(15);
                return(0);
             }
    case 17 : {
                stack[top - 2] = stack[top] * stack[top - 2];
                return(0);
             }
    case 18 : {
                t2 = stack[top - 2];
                if(t2>N_symbol) erreur(13) ;
                if( (t2 > 2) && (sym_tab[t2].type !=' ') )
                                                    erreur(11);
                sym_tab[t2].entry1 = seq_ptr;
                sym_tab[t2].entry2 = ran_ptr;
                sym_tab[t2].type = 'V';
                return(0);
```

```
            }
   case 19 : {
               if(++ran_ptr>N_borne) erreur(14);
               sup_borne[ran_ptr] = stack[top];
               inf_borne[ran_ptr] = stack[top - 2];
               stack[top - 2] = stack[top] - stack[top-2]+1;
               return(0);
             }
   case 20 : {
               stack[top - 2] = stack[top - 2] + stack[top];
               return(0);
             }
   case 21 : {
               stack[top - 2] = stack[top - 2] - stack[top];
               return(0);
             }
   case 23 : {
               stack[top - 2] *= stack[top];
               return(0);
             }
  case 24 : { stack[top-2] /= stack[top] ;
              return(0) ; }
   case 27 : {
               stack[top - 1] = -stack[top];
               return(0);
             }
  case 28 : { if( stack[top-1] < stack[top-5] )
                                  stack[top-5]=stack[top-1];
              return(0) ;
            }
  case 29 : { if( stack[top-5] < stack[top-1] )
                                  stack[top-5] = stack[top-1];
              return(0) ;
            }
   case 30 : {
               stack[top - 2] = stack[top - 1];
               return(0);
             }
   case 31 : {
               to = stack[top];
               if(to>N_symbol) erreur(13) ;
               if( declare && ( sym_tab[to].type != 'P' ) )
                                                  erreur(3);
               if((sym_tab[to].type=='P')||(sym_tab[to].type=='I'))
                    { stack[top] = sym_tab[to].entry1; return(0); }
               erreur(4);
             }
/*-------------corps du programme    ----------------------*/

  case 33 : {
```

```
              if( ++TIME <= stack[1] ) {location = -1;
                                        return(0); }
              fin_etape= 0;            /* fin de l'etape 3 */
              return(0);
             }
 case 39 : { condition[top-4] = 1 ;
             return(0) ; }
 case 40 : { t2 = stack[top-3] ; i=0 ;
             if( condition[top-7] && condition[top-13])
                                       condition[top-7]=1;
             else condition[top-7] = 0 ;
             if( condition[top-7] )
               { vect_stock[t2][TIME] = Qc ;
                 if( var_expr(fstack[top-1]))
                 { if(d_egal(fstack[top-1].variable) && d_egal(Qc))
                                                    return(0);
                   if(d_egal(fstack[top-1].variable) && diff(Qc))
                      { vect_stock[t2][TIME] = d ; Qc = d ;
                        return(0); }
                   if(egal(fstack[top-1].variable,Qc))
                      { Qc=d ; return(0); }
                  }
                 var_ecrit(&Qc) ;
                 printf(":=");
                 exp_ecrit(&fstack[top-1]) ;
                 printf(" ; \n"); Qc=d ;
               }
           return(0) ;
            }
 case 43 : { if(condition[top] && condition[top-2])
                                  condition[top-2]=1 ;
             else condition[top-2] = 0 ;
             return(0) ;
           }
 case 44 : { if( condition[top-2] || condition[top] )
                                   condition[top-2] = 1;
             else condition [top-2] =0 ;
             return(0) ;
           }
 case 45 : { condition[top-2] = condition[top-1] ;
             return(0) ;
           }
 case 47 : { if(egal(fstack[top].variable,fstack[top-2].variable))
               condition[top-2] = 1 ;
             else condition[top-2] = 0 ;
             fstack[top-2].variable=fstack[top].variable=varvide ;
             fstack[top-2].droite = fstack[top].gauche = NULL ;
             fstack[top].droite = fstack[top].gauche = NULL ;
             return(0) ;
            }
```

```
case 48 : { if(egal(fstack[top-2].variable,fstack[top].variable))
              condition[top-2] = 0;
            else condition[top-2] = 1 ;
            fstack[top-2].variable=fstack[top].variable=varvide ;
            fstack[top-2].droite = fstack[top-2].gauche = NULL ;
            fstack[top].droite = fstack[top].gauche = NULL ;
            return(0) ; }
case 49 : { if( TIME <= stack[top] ) condition[top-2] = 1 ;
            else condition[top-2] = 0 ;
            return(0) ;
          }
case 50 : { if( TIME < stack[top] ) condition[top-2] = 1 ;
            else condition[top-2] = 0 ;
            return(0) ;
          }
case 51 :
case 59 : { if( TIME == stack[top] ) condition[top-2] = 1 ;
            else condition[top-2] = 0 ;
            return(0) ;
          }
case 52 :
case 60 : { if( TIME != stack[top] ) condition[top-2] = 1 ;
            else condition[top-2] = 0 ;
            return(0) ;
           }
case 53 : { if( TIME > stack[top] ) condition[top-2] = 1 ;
            else condition[top-2] = 0 ;
            return(0) ;
          }
case 54 : { if(TIME >= stack[top]) condition[top-2] = 1 ;
            else condition[top-2] = 0 ;
            return(0) ;
          }
case 55 : { if(stack[top] >TIME ) condition[top-2] = 1 ;
            else condition[top-2] = 0 ;
            return(0) ;
          }
case 56 : { if( stack[top] >= TIME ) condition[top-2] = 1 ;
            else condition[top-2] =0 ;
            return(0) ;
          }
case 57 : { if( stack[top] <= TIME ) condition[top-2] =1 ;
             else condition[top-2] = 0 ;
             return(0) ;
           }
case 58 : { if( stack[top] < TIME ) condition[top-2] = 1;
            else condition[top-2] = 0 ;
            return(0) ;
          }
case 61 : { if( stack[top-4] < TIME && TIME < stack[top])
```

```
                                        condition[top-2]=1;
            else condition[top-2] = 0 ;
            return(0) ;
          }
  case 62 : {
            to = stack[top - 2];
            if( sym_tab[to].entry1 >= sym_tab[to].entry2 )
                                        return(0);
            ++sym_tab[to].entry1;
            location = stack[top - 3];
            return(0);
          }
  case 63 : {
            t2 = stack[top - 5];
            if( sym_tab[t2].type != 'I' ) erreur(5);
            sym_tab[t2].entry1 = stack[top - 3];
            sym_tab[t2].entry2 = stack[top - 1];
            stack[top - 6] = location;
            return(0);
          }
  case 64 : {
            t2 = stack[top - 2];  to=stack[top] ;
            if(t2>N_vect) erreur(15) ;
            i=0 ;
            vect_stock[t2][TIME] = Qc ;
            if( var_expr(fstack[top] ) )
            {
              if( d_egal(fstack[top].variable)&&d_egal(Qc))
                                                  return(0);
              if( d_egal(fstack[top].variable) && diff(Qc) )
                { vect_stock[t2][TIME] = d ; Qc=d ;
                  return(0) ; }
             if( egal(fstack[top].variable,Qc))
                { Qc=d ;  return(0) ; }
            }
            var_ecrit(&Qc);
            printf(":=");
            exp_ecrit(&fstack[top]);
            printf(" ;\n") ; Qc=d ;
            return(0);
          }
  case 65 : {
            t2 = stack[top - 3]; to = stack[top - 1];
            if(inf_borne[++ran_ptr] || sup_borne[ran_ptr] )
                                                erreur(12);
            seq_ptr = sym_tab[t2].entry1 + to;
            stack[top - 3] = seq_ptr;
            return(0);
          }
  case 66 : {
```

```
            to = stack[top];
            if( sym_tab[to].type != 'V' ) erreur(9);
            if( sym_tab[to].entry2 != -1 ) erreur(10);
            stack[top] = sym_tab[to].entry1;
            return(0);
          }
 case 67 : {
            to = stack[top];
            ran_ptr++ ;
            if(to<inf_borne[ran_ptr] || to>sup_borne[ran_ptr] )
                                                    erreur(1);
            stack[top - 2] = stack[top-2]*(sup_borne[ran_ptr] -
            inf_borne[ran_ptr]+1)+(stack[top]-inf_borne[ran_ptr]);
            return(0);
          }
 case 68 : {
            t2 = stack[top - 2]; to = stack[top];
            if(t2>N_symbol) erreur(13) ;
            if( sym_tab[t2].type != 'V' || sym_tab[t2].entry2 < 0 )
                                                    erreur(8);
            ran_ptr = sym_tab[t2].entry2;
            if( to<inf_borne[ran_ptr] || to>sup_borne[ran_ptr])
                                                    erreur(1);
            stack[top] = stack[top] - inf_borne[ran_ptr];
            return(0);
         }
 case 69 : { to = stack[top]  ; t2 = stack[top-2] ;
            if( d_egal(fstack[top].variable) ||
                                d_egal(fstack[top-2].variable))
            {
                fstack[top-2].variable = d ;
                fstack[top-2].op = ' ' ;
                fstack[top-2].droite = fstack[top-2].gauche = NULL ;
                return(0);
               }
            if(reel_expr(fstack[top])&&reel_expr(fstack[top-2]))
             { somme(fstack[top-2].variable.idf,
                                    fstack[top].variable.idf);
               return(0) ; }
             qt[i]= fstack[top-2] ; qg[i]= fstack[top] ;
             fstack[top-2].op = '+' ;
             fstack[top-2].variable = varvide ;
             fstack[top-2].droite = &qg[i] ;
             fstack[top-2].gauche = &qt[i] ;
             i++ ;
             fstack[top].variable = varvide ;
             fstack[top].op = ' ' ;
             fstack[top].droite = fstack[top].gauche = NULL ;
             return(0);
           }
```

```
  case 70 : { to = stack[top]  ; t2 = stack[top-2] ;
              if( d_egal(fstack[top].variable) ||
                                  d_egal(fstack[top-2].variable))
               {
                   fstack[top-2].variable = d ;
                   fstack[top-2].op = ' ' ;
                   fstack[top-2].droite=fstack[top-2].gauche=NULL;
                   return(0);
                  }
               if( reel_expr( fstack[top] ) &&
                                        reel_expr( fstack[top-2] ) )
                { moin(fstack[top-2].variable.idf,
                                           fstack[top].variable.idf);
                  return(0) ; }
                qt[i]= fstack[top-2] ; qg[i]= fstack[top] ;
                fstack[top-2].op = '-' ;
                fstack[top-2].variable = varvide ;
                fstack[top-2].droite = &qg[i] ;
                fstack[top-2].gauche = &qt[i] ;
                i++ ;
                fstack[top].variable = varvide ;
                fstack[top].op = ' ' ;
                fstack[top].droite = fstack[top].gauche = NULL ;
                return(0);
             }
  case 72 : { to = stack[top]  ; t2 = stack[top-2] ;
              if( d_egal(fstack[top].variable) ||
                                   d_egal(fstack[top-2].variable))
               {
                  fstack[top-2].variable = d ;
                  fstack[top-2].op = ' ' ;
                  fstack[top-2].droite = fstack[top-2].gauche = NULL ;
                  return(0);
                 }
               if( reel_expr( fstack[top] ) &&
                                        reel_expr( fstack[top-2] ) )
                { mult(fstack[top-2].variable.idf,
                                           fstack[top].variable.idf);
                  return(0) ; }
                qt[i]= fstack[top-2] ; qg[i]= fstack[top] ;
                fstack[top-2].op = '*' ;
                fstack[top-2].variable = varvide ;
                fstack[top-2].droite = &qg[i] ;
                fstack[top-2].gauche = &qt[i] ;
                i++ ;
                fstack[top].variable = varvide ;
                fstack[top].op = ' ' ;
                fstack[top].droite = fstack[top].gauche = NULL ;
                return(0);
             }
```

```
  case 73 : { to = stack[top]  ; t2 = stack[top-2] ;
              if( d_egal(fstack[top].variable) ||
                                  d_egal(fstack[top-2].variable))
               {
                 fstack[top-2].variable = d ;
                 fstack[top-2].op = ' ' ;
                 fstack[top-2].droite = fstack[top-2].gauche = NULL ;
                 return(0);
               }
               if( reel_expr( fstack[top] ) &&
                                    reel_expr( fstack[top-2] ) )
                { div(fstack[top-2].variable.idf,
                                    fstack[top].variable.idf);
                  return(0) ; }
               qt[i]= fstack[top-2] ; qg[i]= fstack[top] ;
               fstack[top-2].op = '/' ;
               fstack[top-2].variable = varvide ;
               fstack[top-2].droite = &qg[i] ;
               fstack[top-2].gauche = &qt[i] ;
               i++ ;
               fstack[top].variable = varvide ;
               fstack[top].op = ' ' ;
               fstack[top].droite = fstack[top].gauche = NULL ;
               return(0);
             }
  case 75 : {
               return(0) ;
             }
  case 77 : {
               to = stack[top] ;
               if(to>N_vect) erreur(15) ;
               fstack[top].variable = vect_stock[to][TIME] ;
               fstack[top].op = ' ' ;
               fstack[top].droite = fstack[top].gauche = NULL ;
               stack[top] = 0 ;
               return(0) ;
             }
  case 78 : {
               to = stack[top] ; t2 = stack[top-1] ;
               Qc = vect_stock[to][TIME] ;
               fstack[top-1].variable = Qc ;
               fstack[top-1].op = ' ' ;
               fstack[top-1].droite=fstack[top-1].gauche=NULL ;
               stack[top-1] = stack[top]  ;
               return(0) ;
             }
  case 79 : {
               TIME = stack[top - 1] ;
               fstack[top-1] = fstack[top] ;
               stack[top - 1] = stack[top] ;
```

```
                return(0) ;
              }
    case 80 : {
                fstack[top-2] = fstack[top-1] ;
                stack[top - 2] = stack[top -1] ;
                return(0) ;
              }
    case 81 : {
                to = stack[top] ;
                if( TIME <= to ) {
                          causal = 1 ;
                          stack[top-1] = 1 ;
                          fstack[top-1].variable= d ;
                          fstack[top-1].op = ' ' ;
                          fstack[top-1].droite = NULL ;
                          fstack[top-1].gauche = NULL ;
                          return(0) ;
                          }
                stack[top - 1] = TIME ;
                TIME = TIME - to ;
                return(0) ;
              }
    case 82 : {
                to = stack[top] ;
                if( TIME <= to ) {
                          causal = 1 ;
                          stack[top-1] = 0 ;
                          fstack[top-1].variable = zero ;
                          fstack[top-1].op = ' ' ;
                          fstack[top-1].droite = NULL ;
                          fstack[top-1].gauche = NULL ;
                          return(0) ;
                          }
                stack[top - 1] = TIME ;
                TIME = TIME - to ;
                return(0) ;
              }
    case 83 : {
                to = stack[top] ;
                t2 = (TIME + to) % (1 + to) ;
                if( t2 ) {
                      causal = 1 ;
                      stack[top-1] = 1 ;
                      fstack[top-1].variable= d ;
                      fstack[top-1].op = ' ' ;
                      fstack[top-1].droite = NULL ;
                      fstack[top-1].gauche = NULL ;
                      return(0) ;
                       }
                t2 = (TIME + to) / (1 + to) ;
```

```
                stack[top - 1] = TIME ;
                TIME = t2 ;
                return(0) ;
              }
    case 84 : {
                stack[top - 2] = stack[top - 1] ;
                return(0) ;
              }
    case 85 : {
                stack[top + 1] = 1 ;
                return(0) ;
              }
   /*---------------- les entrees   --------------------------*/

    case 86 : {
                fin_etape = 0 ;  /* fin de l'etape 2 */
                return(0) ;
              }
    case 88 : {
                stack[1] = stack[top] ;
                if(stack[top]>maxtime) erreur(16) ;
                return(0) ;
              }
    case 89 : {
                to = stack[top] ;
                lecture = 1 ;
                for (j=1 ; j <= stack[1] ; j++)
                     if( lecture )
                     {
                      getvar(&vect_stock[to][j],&d_flag) ;
                      if( d_flag ==1 ) vect_stock[to][j] = d ;
                      if(d_flag==-1)
                           { lecture = 0 ;
                             for( nb=j ; nb<=stack[1] ; nb++)
                             vect_stock[to][nb] = d ;
                            }
                    }
                 return(0) ;
               }
    case 90 : {
                to = stack[top - 1] ;
                if( sym_tab[to].entry1 >= sym_tab[to].entry2 )
                                                  return(0) ;
                ++sym_tab[to].entry1 ;
                location = stack[top - 2] ;
                return(0) ;
              }
    case 91 : {
                t2 = stack[top - 4] ;
                if(t2>N_symbol) erreur(13) ;
```

```
                if( sym_tab[t2].type != 'I' ) erreur(5) ;
                sym_tab[t2].entry1 = stack[top - 2] ;
                sym_tab[t2].entry2 = stack[top] ;
                stack[top - 5] = location ;
                return(0) ;
              }
  /*-------------------les initialisations----------------------*/
   case 96 : {
                to = stack[top] ;
                if( initial[to] ) {
                        getvar(&v[to],&d_flag) ;
                        initial[to] = 0 ;
                        for(j=0; j< stack[1] ; j++ )
                          vect_stock[to][j] = v[to] ;
                         }
                return(0) ;
              }
   case 97 : {
                to = stack[top-1] ;
                if( sym_tab[to].entry1 >= sym_tab[to].entry2 )
                                                  return(0) ;
                ++sym_tab[to].entry1 ;
                location = stack[top-2] ;
                return(0) ;
              }
  }
}
/*------------------ fin des routines --------------------*/


var_ecrit(x) struct var *x ;

{ int i ;
  if( x != NULL )
        if( x->taille > 0 )
            { for(i=0; i<6 && x->idf[i]!=' '  ; i++)
                              printf("%c",x->idf[i]) ;
              printf("(%d",x->indice[0]);
              for(i=1;i<x->taille-1;i++)
                              printf(",%d",x->indice[i]);
              if( x->taille > 1 )
                    printf(",%d)",x->indice[x->taille-1]);
              else printf(")");
            }
    else for(i=0;i<6 && x->idf[i]!=' ' ;i++)
              printf("%c",x->idf[i]) ;
return(0) ;
}

exp_ecrit(x) struct expr *x ;
```

```
{ if( x != NULL )
        if( x->droite != NULL  && x->gauche != NULL )
              { printf("( ") ;
                exp_ecrit(x->gauche) ;
                var_ecrit(&x->variable) ;
                if( x->op != ' ' )  printf("%c",x->op) ;
                exp_ecrit(x->droite) ;
                printf(" )") ; }
        else if(x->gauche != NULL )
               { var_ecrit(&x->variable) ;
                 if(x->op !=' ' )
                 printf("%c",x->op) ;
                 exp_ecrit(x->gauche) ;  }
     else if( x->droite != NULL )
            { exp_ecrit(x->droite) ;
              var_ecrit(&x->variable) ;
              if( x->op != ' ' ) printf("%c",x->op);
            }
else { var_ecrit(&x->variable) ;
       if( x->op != ' ' ) printf("%c",x->op) ; }
return(0) ;
}

/*la fonction getvar lit les donnees du fichier de donnees */

 getvar(z,r)  struct var *z ;
                int *r ;
   {
    int c, i , j , tail , valeur , neg ;
    i=0 ;

    while ((c=getchar()) == ' ' || c == '\n' ) ;
    if( c == 'd' ) { *r = 1 ;
                     *z = d ;
                     return(0) ;}
    if(c == '.'&&( getchar() == '.' )&&(getchar() == '.' ))
                     { *r = -1 ;
                       *z = d ;
                     return(0) ;}
    while( c!='(' && c!=' ' && c!= '\n' )
                    { z->idf[i++]=c ; c=getchar() ;}
    for(j=i ; j < 6 ; j++ ) z->idf[j] = ' ' ;
    tail = 0 ; neg = 0 ;
    if( c=='(' ) while ( c!=')')
            {
                c=getchar() ;
                if( c == '-' ) { c = getchar() ;  neg = 1 ; }
                valeur = 0 ;
                while( c!=',' && c!=')')
```

```
                {
                  if( isdigit(c))
                       valeur=valeur*10+(c-'0');
                  else  erreur(0) ;
                  c=getchar() ;
                }
                if( neg ) z->indice[tail++] = -1*valeur ;
                else z->indice[tail++] = valeur ;
                neg = 0 ;
              }
        z->taille = tail ;
        *r = 0 ;
        return(0) ;
}

  isdigit(d) int d ;
  {
   if( d <= '9' &&  d >= '0' ) return(1) ;
   return(0) ;
  }

/*---------d_egal(x)  test si la variable  x == nil --------*/
d_egal(x) struct var x  ;

  { if( x.taille == 0 && x.idf[0] == 'd' && x.idf[1] == ' ' )
                                                    return(1) ;
    return(0) ;
}

/*   egal(x,y) test si les variables x et y sont egales */
egal(x,y) struct var x , y ;

{ int tt = 1 , i ;
  if( x.taille != y.taille ) return(0) ;
  for(i=0 ; i<5 && y.idf[i]!= ' ' && x.idf[i] != ' ' ; i++ )
                          if( x.idf[i] != y.idf[i] ) tt = 0 ;
  if ( tt==0 || x.idf[i]!=y.idf[i] ) return(0) ;
  for( i=0 ; i<x.taille ; i++ ) if(x.indice[i]!=y.indice[i])
                                                    tt=0 ;
  if( tt==0 ) return(0) ;
  return(1) ;
}

/* v_diff(x,y) test si les variables x et y sont distinctes */
v_diff(x,y) struct var x,y ;
{ if( egal(x,y) ) return(0) ;
  return(1) ;
}

diff(x) struct var x ;
```

```
{ if( d_egal(x)  ) return(0) ;
  return(1) ;
}

/* var_expr(x) test si x est une variable  */
var_expr(x) struct expr x ;
{ if( x.op == ' ' && x.droite == NULL && x.gauche == NULL )
                                                    return(1) ;
  return(0) ;
}
/* reel_expr(x) test si x est un reel  */
reel_expr(x) struct expr x ;
{ char c ;
  c = x.variable.idf[0] ;
  if( var_expr(x) && c!='-' && (c=='0' || c=='1' || c=='2'
  || c=='3' || c=='4'|| c=='5' || c=='6' || c=='7' ||
     c=='8' || c=='9') ) return(1) ;
  if( var_expr(x) && c=='-') {
   c = x.variable.idf[1] ;
   if( c=='0' || c=='1' || c=='2' || c=='3' || c=='4' || c=='5'
   || c=='6' || c=='7' || c=='8' || c=='9' ) return(1) ; }
  return(0) ;
}

somme(x,y) char x[6] , y[6] ;
{ int n1 , n2 ;
  n1= convint(x) ;
  n2= convint(y) ;
  n1 = n1+n2 ;
  convchar(n1,x) ;
  return(0) ;
}
 moin(x,y) char x[6] , y[6] ;
{ int n1 , n2 ;
  n1 = convint(x) ;
  n2 = convint(y) ;
  n1 = n1-n2 ;
  convchar(n1,x) ;
  return(0) ;
}

mult(x,y) char x[6] , y[6] ;
{ int n1 , n2 ;
  n1= convint(x) ;
  n2= convint(y) ;
  n1 = n1*n2 ;
  convchar(n1,x) ;
  return(0) ;
}
```

```
div(x,y) char x[6] , y[6] ;
{ int n1 , n2 ;
  n1 = convint(x) ;
  n2 = convint(y) ;
  n1=n1/n2 ;
  convchar(n1,x) ;
  return(0) ;
}

convint(x) char x[6] ;
{ int i, j, n ;
  n=0 ; j=0 ;
  if( x[0]=='-' ) j=1 ;
  for(i=j ; i<6 && x[i]!=' ' ; i++ ) n=10*n+x[i]-'0' ;
  if(j) return(-1*n) ; else return(n) ;
}

convchar(n,x) int n ; char x[6] ;
{ int i , j , k ; char s[6] ;
  j=0 ; k=1 ;
  if( n<0 ) { j=1 ; n = -1*n ; }
  for(i=j; k>0 && i<6 ; i++ ) { s[i]=n%10 ; n=n/10 ;
                                if (n==0) k=0 ;}
  if(j) x[0]='-' ;
  for(k=j; k<i ; k++ ) x[k]= s[i-k-1+j]+'0'  ;
  for(k=i; k<6; k++) x[k] =' ' ;
  return(0) ;
}

 /*  MESSAGE D'ERREURS  */

 erreur(i) int i ;
 { switch(i)
   {
    case 0 : {printf(" erreur : donnee incorrecte  \n") ;
              exit(0) ; }
    case 1 : {printf(" tableau de vecteurs hors bornes \n");
              exit(0) ;}
    case 2 : {printf("arguments d' operateurs incorrects ou\n");
              printf("indices des vecteurs incorrects \n");
              exit(0) ;}
    case 3 : {printf("seul les parametres peuvent etre utilises
                                                    dans\n ");
              printf(" les declarations des bornes des tableaux
                                                   de vect \n");
              exit(0) ;}
    case 4 : {printf("manque d'indice du vecteur  \n");
              exit(0) ;}
    case 5 : {printf("l' indice de POUR doit etre declare en
                                                     INDICE \n");
```

```
            exit(0) ;}
    case 6 : {printf("donnee incorrect ds le fichier de donnees
                                                          \n");
            exit(0) ;}
    case 7 : {printf("donnees insuffisantes ds le fichier de
                                                  donnees \n");
            exit(0) ;}
    case 8 : {printf("tableau de vecteurs non declare \n");
             exit(0) ;}
    case 9 : {printf("vecteur non declare \n");
             exit(0) ;}
    case 10 : {printf("manque  d'arguments   \n");
             exit(0) ;}
    case 11 : {printf("variable declaree plusieurs fois   \n");
             exit(0) ;}
    case 12 : {printf(" arguments de l' operateur incorrect \n");
             exit(0) ; }
    case 13 : {printf("debordement de la pile de travail \n");
             exit(0) ; }
    case 14 : {printf("debordement de la table des bornes \n");
             exit(0) ; }
    case 15 : { printf("vecteur hors des bornes declarees \n");
             exit(0) ; }
    case 16 : { printf(" MAXT doit etre <= %d\n", maxtime) ;
             exit(0) ;}
  }
 }
```

Résumé

Cette thèse comporte deux parties complémentaires. La première partie qui est de nature algorithmique, propose une formulation combinatoire pour la conception d'algorithmes de produit matriciel sur les réseaux systoliques linéaires. Cette approche est utilisée pour conçevoir divers algorithmes sur de tels réseaux, en jouant sur divers paramètres: directions de circulation des données, existence ou non de signaux de contrôle, caractère combinatoire ou programmable de la cellule de base. Dans la deuxième partie, consacrée à la validation des algorithmes systoliques, nous proposons deux logiciels: le premier, SISYC, est un simulateur numérique d'algorithmes systoliques. Le second, SISYC2, calcule la trace symbolique des algorithmes systoliques et permet lorsqu'il est couplé avec un système de calcul formel tel que Reduce, de réaliser une simulation formelle. Nous montrons en quoi cet outil permet de faire la validation d'algorithmes systoliques.

Mots clés:

algorithme, architecture systolique, calcul formel, formulation combinatoire, élimination de Gauss, produit matriciel, réseau linéaire , simulation, validation.