Calcul formel et parallélisme : forme normale d'Hermite, méthodes de calcul et parallélisation

Françoise Roch

▶ **To cite this version:**

Françoise Roch. Calcul formel et parallélisme : forme normale d'Hermite, méthodes de calcul et parallélisation. Modélisation et simulation. Institut National Polytechnique de Grenoble - INPG, 1990. Français. NNT : . tel-00336486

HAL Id: tel-00336486

https://theses.hal.science/tel-00336486

Submitted on 4 Nov 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THESE

Présentée par Françoise ROCH

pour obtenir le titre de **DOCTEUR**

de l'**INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE**
(arrêté ministériel du 23 novembre 1988)

(Spécialité: **Mathématiques Appliquées)**

Calcul Formel et Parallélisme

FORME NORMALE D'HERMITE

METHODES DE CALCUL ET PARALLELISATION

Date de soutenance: 29 Janvier 1990

Composition du jury:

Président:	D. Lazard
Rapporteurs:	J.H. Davenport
	D. Trystram
Examinateurs:	J. Della Dora
	J.M. Dion

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

A mes Parents,

A Jean-Louis,

A toute notre famille.

REMERCIEMENTS

Je tiens à remercier le professeur Daniel Lazard, qui a bien voulu me faire l'honneur de présider le jury de cette thèse.

Je remercie monsieur James Davenport, professeur à l'université de Bath, pour avoir lu avec attention mon travail et pour avoir accepté de venir de Grande Bretagne pour participer à ce jury.

Je remercie Denis Trystram pour ses remarques et critiques constructives qui m'ont permis d'améliorer ce travail; j'apprécie en outre beaucoup sa gentillesse.

Je remercie monsieur Jean Michel Dion, directeur de recherche au CNRS, pour avoir bien voulu participer à ce jury.

Jean Della Dora m'a dirigée dans mon travail et je l'en remercie vivement. Ses conseils -et ses pâtisseries...-m'ont permis de mener à bien ce travail.

Pascale Sénéchaud et Gilles Villard (que je remercie pour son rôle déterminant dans la détermination du déterminant pour le calcul déterministe de la forme d'Hermite), compagnons de maîtrise, de DEA et de thèse, m'ont rendu agréables ces dernières années d'études. Je les en remercie, et je n'oublie pas tous les autres membres de l'équipe de calcul formel et algorithmique parallèle, en particulier Guoting Chen, Mustapha Daoudi, Abdel Barkatou, Jean-Laurent Philippe et Afonso Ferreira.

Merci à Joëlle Prévost, à Evelyne Tournier et à Claire Dicrescenso pour leur aide précieuse.

Je remercie Jacques Ballasi et Denis Berthet pour leur travail dans la réalisation de l'arithmétique polynomiale de PAC, sans laquelle une partie de ce travail n'aurait pu être réalisée.

Quant à Jean-Louis, sans son aide, ses suggestions et remarques, son arithmétique rationnelle et le système PAC (brique essentielle de ce travail), et surtout sa patience (!...), je me serais souvent découragée.

Enfin, je remercie tous les membres du service de scolarité de l'INPG et du service de reprographie de l'IMAG pour les photocopies de dernière minute et pour la réalisation finale de cette thèse.

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

46 avenue Felix Viallet
38031 GRENOBLE cedex

Tél. : 76.57.45.00

Année universitaire 1989

Président de l'Institut :
Monsieur Georges LESPINARD

Professeurs des Universités

BARIBAUD Michel	ENSERG	JAUSSAUD Pierre	ENSIEG
BARRAUD Alain	ENSIEG	JOST Rémy	ENSPG
BAUDELET Bernard	ENSPG	JOUBERT Jean-Claude	ENSPG
BEAUFILS Jean-Pierre	INPG	JOURDAIN Geneviève	ENSIEG
BLIMAN Samuel	ENSERG	LACOUME Jean-Louis	ENSIEG
BOIS Philippe	ENSHMG	LADET Pierre	ENSIEG
BONNETAIN Lucien	ENSEEG	LESIEUR Marcel	ENSHMG
BONNET Guy	ENSPG	LESPINARD Georges	ENSHMG
BRISSONNEAU Pierre	ENSIEG	LONGEQUEUE Jean-Pierre	ENSPG
BRUNET Yves	IUFA	LORET Benjamin	ENSHMG
CAILLERIE Denis	ENSHMG	LOUCHET François	ENSEEG
CAVAIGNAC Jean-François	ENSPG	LUCAZEAU Guy	ENSEEG
CHARTIER Germain	ENSPG	MASSE Philippe	ENSIEG
CHENEVIER Pierre	ENSERG	MASSELOT Christian	ENSIEG
CHERADAME Hervé	UFR PGP	MAZARE Guy	ENSIMAG
CHERUY Arlette	ENSIEG	MOHR Roger	ENSIMAG
CHOVET Alain	ENSERG	MOREAU René	ENSHMG
COHEN Joseph	ENSERG	MORET Roger	ENSIEG
COLINET Catherine	ENSEEG	MOSSIERE Jacques	ENSIMAG
CORNUT Bruno	ENSIEG	OBLED Charles	ENSHMG
COULOMB Jean-Louis	ENSIEG	OZIL Patrick	ENSEEG
COUMES André	ENSERG	PA ULEAU Yves	ENSEEG
CROWLEY James	ENSIMAG	PERRET Robert	ENSIEG
DARVE Félix	ENSHMG	PIAU Jean-Michel	ENSHMG
DELLA-DORA Jean	ENSIMAG	PIC Etienne	ENSERG
DEPEY Maurice	ENSERG	PLATEAU Brigitte	ENSIMAG
DEPORTES Jacques	ENSPG	POUPOT Christian	ENSERG
DEROO Daniel	ENSEEG	RAMEAU Jean-Jacques	ENSEEG
DESRE Pierre	ENSEEG	REINISCH Raymond	ENSPG
DOLMAZON Jean-Marc	ENSERG	RENAUD Maurice	UFR PGP
DURAND Francis	ENSEEG	ROBERT André	UFR PGP
DURAND Jean-Louis	ENSPG	ROBERT François	ENSIMAG
FAUTRELLE Yves	ENSHMG	SABONNADIERE Jean-Claude	ENSIEG
FOGGIA Albert	ENSIEG	SAUCIER Gabrièle	ENSIMAG
FONLUPTJean	ENSIMAG	SCHLENKER Claire	ENSPG
FOULARD Claude	ENSIEG	SCHLENKER Michel	ENSPG
GANDINI Alessandro	UFR PGP	SERMET Pierre	ENSERG
GAUBERT Claude	ENSPG	SILVY Jacques	UFR PGP
GENTIL Pierre	ENSERG	SIRIEYS Pierre	ENSHMG
GENTIL Sylviane	ENSIEG	SOHM Jean-Claude	ENSEEG
GREVEN Hélène	IUFA	SOLER Jean-Louis	ENSIMAG
GUEGUEN Claude	ENSIEG	SOUQUET Jean-Louis	ENSEEG
GUERIN Bernard	ENSERG	TROMPETTE Philippe	ENSHMG
GUYOT Pierre	ENSEEG	VINCENT Henri	ENSPG
IVANES Marcel	ENSIEG	ZADWORNY François	ENSERG

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
COURNIL M.
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
LACHENAL D.
LADET Pierre
LATOMBE Claudine
LE HUY H.
LE GORREC Bernard
MADAR Roland
MEUNIER G.
MULLER Jean
NGUYEN TRONG Bernadette
NIEZ J.J.
PASTUREL Alain
PLA Fernand
ROGNON J.P.
ROUGER Jean
TCHUENTE Maurice
VINCENT Henri
YAVARI A.R.

Chercheurs du C.N.R.S

DIRECTEURS DE RECHERCHE CLASSE 0

LANDEAU	Ioan
NAYROLLES	Bernard

Directeurs de recherche 1ère Classe

ANSARA Ibrahim
CARRE René
FRUCHART Robert
HOPFINGER Emile
JORRAND Philippe
KRAKOWIAK Sacha
LEPROVOST Christian
VACHAUD Georges
VERJUS Jean-Pierre

Directeurs de recherche 2ème Classe

ALEMANY Antoine
ALLIBERT Colette
ALLIBERT Michel
ARMAND Michel
AUDIER Marc
BERNARD Claude
BINDER Gilbert
BONNET Roland
BORNARD Guy
CAILLET Marcel
CALMET Jacques
CHATILLON Chritiant
CLERMONT Jean-Robert
COURTOIS Bernard
DAVID René
DION Jean-Michel
DRIOLE Jean
DURAND Robert
ESCUDIER Pierre
EUSTATHOPOULOS Nicolas
GARNIER Marcel
GUELIN Pierre
JOUD Jean-Charles
KAMARINOS Georges
KLEITZ Michel
KOFMAN Walter
LEJEUNE Gérard
MADAR Roland
MERMET Jean
MICHEL Jean-Marie
MEUNIER Jacques
PEUZIN Jean-Claude
PIAU Monique
RENOUARD Dominique
SENATEUR Jean-Pierre
SIFAKIS Joseph
SIMON Jean-Paul
SUERY Michel
TEODOSIU Christian
VAUCLIN Michel
VENNEREAU Pierre
WACK Bernard
YONNET Jean-Paul

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

E.N.S.E.E.G
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.M.G
ROWE Alain

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

C.E.N.G
CADET Jean
COEURE Philippe
DELHAYE Jean-Marc
DUPUY Michel
JOUVE Hubert
NICOLAU Yvan
NIFENECKER Hervé
PERROUD Paul
PEUZIN Jean-Claude
TAIEB Maurice
VINCENDON Marc

Laboratoires extérieurs :

C.N.E.T
DEVINE Rodericq
GERBER Roland
MERCKEL Gérard
PAULEAU Yves

Situation particulière

PROFESSEURS D'UNIVERSITE

DETACHEMENT

ENSIMAG	LATOMBE	J..Claude	Détachement	21/10/1989
ENSHMG	PIERRARD	J.Marie	Détachement	30/04/1989
ENSIMAG	VEILLON	Gérard	Détachement	30/09/1990
ENSIMAG	VERJUS	J.Pierre	Détachement	30/09/1989
ENSPG	BLOCH	Daniel	Recteur à c/	21/12/1988

SURNOMBRE

INPG	CHIAVERINA	Jean	30/09/1989
ENSHMG	BOUVARD	Maurice	30/09/1991
ENSEEG	PARIAUD	J.Charles	30/09/1991

UNIVERSITE Joseph FOURIER (GRENOBLE I)

Président de l'Université :
M. NEMOZ Alain

Année Universitaire 1988 - 1989

MEMBRES DU CORPS ENSEIGNANT DE SCIENCES ET DE GEOGRAPHIE

PROFESSEURS DE 1ère Classe

ADIBA Michel	Informatique
ANTOINE Pierre	Géologie I.R.I.G.M.
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
BRAVARD Yves	Géographie
CARLIER Georges	Biologie Végétale
CASTAING Bernard	Physique
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
FINKE Gerde	Informatique
GAGNAIRE Didier	Chimie Physique
GAUTRON René	Chimie
GENIES Eugène	Chimie
GERMAIN Jean-Pierre	Mécanique,
GIDON Maurice	Géologie
GUITTON Jacques	Chimie
HICTER Pierre	Chimie
IDELMAN Simon	Physiologie Animale
JANIN Bernard	Géographie
JOLY Jean René	Mathématiques Pures

JOSELEAU Jean Paul	Biochimie
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.
LONGEQUEUE Nicole	Physique
LUNA Domingo	Mathématiques Pures
MACHE Régis	Physiologie Végétale
MASCLE Georges	Géologie
MAYNARD Roger	Physique du Solide
OMONT Alain	Astrophysique
OZENDA Paul	Botanique (Biologie Végétale)
PANNETIER Jean	Chimie
PAYAN Jean-Jacques	Mathématiques Pures
PEBAY-PEYROULA Jean-Claude	Physique
PERRIER Guy	Géophysique
PIERRE Jean Louis	Chimie Organique
RENARD Michel	Thermodynamique
RIEDTMANN Christine	Mathématiques
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
VAN CUTSEM Bernard	Mathématiques Appliquées
VIALON Pierre	Géologie

PROFESSEURS de 2ème Classe

ARMAND Gilbert	Géographie
ATTANE Pierre	Mécanique
BARET Paul	Chimie
BERTIN José	Mathématiques
BLANCHI J.Pierre	STAPS
BLOCK Marc	Biologie
BLUM Jacques	Mathématiques Appliquées
BOITET Christian	Mathématiques Appliquées
BORNAREL Jean	Physique
BORRIONE Dominique	Automatique informatique
BOUVET Jean	Biologie
BROSSARD Jean	Mathématiques
BRUANDET J.François	Physique
BRUGAL Gérard	Biologie
BRUN Gilbert	Biologie
CASTAING Bernard	Physique
CERFF Rudiger	Biologie
CHIARAMELLA Yves	Mathématiques Appliquées
CHOLLET Jean Pierre	Mécanique
COLOMBEAU Jean François	Mathématiques (ENSL)
COURT Jean	Chimie
CUNIN Pierre Yves	Informatique
DAVID Jean	Géographie

DHOUAILLY Danielle	Biologie
DUFRESNOY Alain	Mathématiques Pures
GASPARD François	Physique
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
HACQUES Gérard	Mathématiques Appliquées
HERBIN Jacky	Géographie
HERAULT Jeanny	Physique
HERINO Roland	Physique
JARDON Pierre	Chimie
KERCKHOVE Claude	Géologie
MANDARON Paul	Biologie
MARTINEZ Francis	Mathématiques Appliquées
MOREL Alain	Géographie
NEMOZ Alain	Thermodymamique CNRS - CRTBT
NGUYEN HUY Xuong	Informatique
OUDET Bruno	Mathématiques Appliquées
PAUTOU Guy	Biologie
PECHER Arnaud	Géologie
PELMONT Jean	Biochimie
PELLETIER Guy	Astrophysique
PERRIN Claude	Sciences Nucléaires I.S.N.
PIBOULE Michel	Géologie
RAYNAUD Hervé	Mathématiques Appliquées
REGNARD Jean René	Physique
RICHARD Jean-Marc	Physique
RIEDTMANN Christine	Mathématiques Pures
ROBERT Danielle	Chimie
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
VINCENT Gilbert	Physique
VIVIAN Robert	Géographie
VOTTERO Philippe	Chimie

MEMBRES DU CORPS ENSEIGNANT DE L' IUT 1

PROFESSEURS de 1ère Classe

BUISSON Roger	Physique IUT 1
CHEHIKIAN Alain	E.E.A. I.U.T.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

BEE Marc	Physique IUT 1
BOUTHINON Michel	EEA. IUT 1
CHAMBON René	Génie Mécanique IUT 1
CHENAVAS Jean	Physique IUT 1

CHILO Jean	Physique IUT 1
CHOUTEAU Gérard	Physique IUT 1
CONTE René	Physique IUT 1
FOSTER Panayotis	Chimie IUT 1
GOSSE Jean-Pierre	EEA.IUT 1
GROS Yves	Physique IUT 1
HAMAR Roger	Chimie IUT 1
KUHN Gérard, (Détaché)	Physique IUT 1
LEVIEL Jean Louis	Physique IUT 1
MAZUER Jean	Physique IUT 1
MICHOULIER Jean	Physique IUT 1
MONLLOR Christian	EEA.IUT 1
PERRAUD Robert	Chimie IUT 1
PIERRE Gérard	Chimie IUT 1
TERRIEZ Jean-Michel	Génie Mécanique IUT 1
TOUZAIN Philippe	Chimie IUT 1
TURGEMAN Sylvain	Génie civil
VINCENDON Marc	Chimie IUT 1
ZIGONE Michel	Physique 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	Pneumophtisiologie	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.

SOMMAIRE GENERAL

INTRODUCTION

PREMIERE PARTIE :

PRESENTATION THEORIQUE
DU CALCUL DE LA FORME NORMALE D'HERMITE
A COEFFICIENTS DANS UN ANNEAU EUCLIDIEN

.....Introduction
Chapitre I ..... Quelques Notions de Base d'Algèbre
Chapitre II.....Forme d'Hermite sur un Anneau Euclidien
Chapitre III.....Réseaux
Chapitre IV.....Equations Diophantiennes
Chapitre V.....Forme d'Hermite sur les Entiers
Chapitre VI.....Forme d'Hermite sur les Polynômes

DEUXIEME PARTIE :

PARALLELISATION
DU CALCUL DE LA FORME NORMALE D'HERMITE

Chapitre I .....Résultats Théoriques
Chapitre II.....Algorithmes Parallèles
Chapitre III..... Implantation
.....Conclusion

Bibliographie

INTRODUCTION

La résolution exacte d'équations à coefficients entiers est un problème complexe qui préoccupe les mathématiciens depuis plus de 2000 ans.
Le 10ième problème d'Hilbert, formulé par Hilbert en 1900 en est une illustration [HIL00]: "existe-t-il une procédure générale qui permette de décider si une équation Diophantienne admet une solution?"
Ce problème n'a été résolu qu'en 1971 par Matyasevich [MAT71] qui montre qu'il n'existe pas de telle procédure.
La forme normale d'Hermite, définie en 1851 par Hermite [HER51], permet d'obtenir une solution, lorsqu'elle existe, à un système d'équations Diophantiennes linéaires. La méthode directe pour le calcul de cette forme est liée à l'algorithme d'Euclide. Cette méthode a été améliorée en 1970 par Bradley [BRA70]. Le meilleur algorithme actuellement connu pour le cas entier, qui borne la taille des coefficients intermédiaires par le déterminant, date de 1985 [Ill85].

Cette thèse est consacrée à l'étude de la forme normale d'Hermite et à la conception d'algorithmes parallèles pour son calcul. Nous avons examiné deux cas particuliers: le calcul de la forme normale d'Hermite de matrices à coefficients entiers et le calcul de la même forme pour des matrices dont les coefficients sont des polynômes sur les rationnels ($\mathbb{Q}[x]$).

Dans un premier temps nous avons considéré des matrices à coefficients entiers. L'algorithme de Gauss sur les rationnels et le calcul de la forme normale d'Hermite sur les entiers, en séquentiel, sont du même ordre de complexité en nombre d'opérations arithmétiques en précision infinie ($O(n^3)$ multiplications). En réalité la forme normale d'Hermite est un problème beaucoup plus complexe de par la taille des coefficients intermédiaires. Pour le meilleur algorithme séquentiel que nous connaissons, la taille des coefficients intermédiaires est bornée par la taille du déterminant de la matrice initiale. Pour la méthode de Gauss il existe des algorithmes modulaires bien plus performants, utilisant notamment des développements p-adiques [DIX82] [VIL88]. Concernant le calcul de la forme normale d'Hermite, qui nécessite une relation d'ordre sur les coefficients, il n'existe pas actuellement de méthode modulaire adaptée.
Ainsi, le calcul de la forme d'Hermite est d'un facteur quadratique ($O(n^2)$) plus complexe que le calcul de la factorisation de Gauss.

Le calcul de la forme d'Hermite est un problème typique, puisqu'il présente les difficultés classiques rencontrées en calcul formel, avec des opérations tout ce qu'il y a de plus exactes, et des coefficients intermédiaires tout ce qu'il y a de plus "gros" [DST86].
Il nous a donc semblé intéressant, vu la complexité des algorithmes séquentiels, de s'orienter vers une approche parallèle du problème.

L'intérêt ici est double. D'une part, les problèmes d'algèbre linéaire sont souvent de bons candidats à la parallélisation. En effet une opération de ligne ou de colonne consiste à effectuer un certain nombre d'opérations indépendantes, et par là-même présente un parallélisme inhérent. De nombreuses études ont déjà été réalisées en analyse numérique (sur l'algorithme de Gauss [TOU89] et de Givens [DAO89] par exemple) et commencent à se développer en calcul formel.
D'autre part, l'algorithme d'Hermite nécessite du fait du grossissement des coefficients, un volume mémoire très important. L'utilisation d'une mémoire distribuée avec répartition des données est donc particulièrement adaptée.

La thèse est constituée de deux parties.
Dans un premier temps, la forme normale d'Hermite pour des matrices à coefficients dans un anneau Euclidien est présentée. Les concepts et propriétés sur lesquels sont basés les algorithmes sont décrits. Nous introduisons la théorie sur les réseaux et les problèmes qui lui sont attachés. Le calcul de la forme normale d'Hermite répond à l'un de ces problèmes, à savoir l'obtention d'une certaine forme canonique d'un réseau. Les différents algorithmes séquentiels sont présentés. Nous étudions et comparons leurs complexités en nous attachant tout particulièrement à examiner la croissance des coefficients intermédiaires, fondamentale dans le traitement de ce problème.

Dans un deuxième temps, une approche parallèle est considérée. Les résultats théoriques de NC1-réductibilité du problème du calcul de la forme normale d'Hermite au problème de la résolution d'un système linéaire (E. Kaltofen, M.S. Krishnamoorthy et D. Saunders) sont présentés.

L'étude de la parallélisation est décomposée en deux étapes.
Tout d'abord, nous abordons le cas entier. Deux modèles parallèles sont distingués : les modèles à mémoire partagée et à mémoire distribuée. Le premier modèle conduit à ordonnancer les tâches indépendantes (PRAM). Deux algorithmes sont proposés : ils adaptent au calcul formel de la forme normale d'Hermite, les techniques de parallélisation utilisées pour le calcul numérique de la factorisation orthogonale d'une matrice (Givens), proposées par A.H.Sameh et D.J.Kuck d'une part, et M.Cosnard et Y.Robert d'autre part. Les complexités théoriques de ces deux algorithmes sont étudiées.

Le modèle distribué conduit à partager les données : nous dégageons deux algorithmes parallèles. L'un correspond à une répartition par ligne de la matrice (algorithme HL), l'autre a une répartition par colonne (algorithme HC). L'algorithme HC présente une meilleure complexité théorique. En pratique, l'implantation sur le FPS T40 (32 processeurs) tend à prouver que l'algorithme HL est expérimentalement plus performant en raison des temps de synchronisation plus faibles entre processeurs.

Les résultats expérimentaux s'avèrent probants: le calcul de la forme normale d'Hermite d'une matrice 160x160 nécessite 3 heures sur 32 processeurs avec l'algorithme HL.

Enfin, nous traitons le cas des matrices dont les composantes sont des polynômes. Deux études sont distinguées. Dans la première, les coefficients des polynômes sont dans un corps fini (Z/pZ[x] avec p premier) : le coût des calculs sur les coefficients peut alors être supposé constant. Dans la deuxième étude, les coefficients sont rationnels (Q[x]) : le coût des opérations de base dépend alors de la taille des coefficients manipulés. Le calcul de la Forme Normale d'Hermite de matrices à coefficients polynomiaux est particulièrement couteux : la parallélisation sur le modèle proposé donne une complexité parallèle qui reste importante. Le déséquilibre dynamique des tâches lors de l'exécution des algorithmes montre les limites d'une approche parallèle statique et conduit à s'orienter vers une répartition dynamique des tâches.

Ce travail s'inscrit dans le cadre du projet massivement parallèle PAC(Parallel Algebraic Computing). Le calcul de la forme d'Hermite à coefficients polynomiaux nous a conduits à participer au développement de l'arithmétique polynomiale de ce système, et plus particulièrement aux primitives de communications de polynômes entre processeurs.

Première Partie

PRESENTATION THEORIQUE DU CALCUL DE LA FORME NORMALE D'HERMITE D'UNE MATRICE A COEFFICIENTS DANS UN ANNEAU EUCLIDIEN

SOMMAIRE DE LA PREMIERE PARTIE

CHAPITRE I QUELQUES NOTIONS DE BASE D'ALGEBRE

I.1. Anneaux Intègres 10
I.2. Anneaux Factoriels 11
I.3. Anneaux Euclidiens 12

CHAPITRE II FORME D'HERMITE SUR UN ANNEAU EUCLIDIEN

II.1. Matrices sur un anneau euclidien 14
II.2. Forme normale d'Hermite 15
II.3. Forme normale de Smith 18

CHAPITRE III RESEAUX

III.1. Définitions 19
III.2. Bases 20
III.3. Lien avec le déterminant 21
III.4. Problèmes fondamentaux 21
III.5. Bases réduites 22
III.6. Problème du plus court vecteur 24
III.7. Approximation diophantienne simultanée 26
III.8. Conclusion 27

CHAPITRE IV EQUATIONS DIOPHANTIENNES

IV.1. Généralités 28
IV.2. Algorithmes 29
IV.2.1. Algorithme d'Euclide étendu 29
IV.2.2. Algorithme de Blankinship 31
IV.2.3. Algorithme de Bradley 33

CHAPITRE V FORME NORMALE D'HERMITE SUR LES ENTIERS

V.1. Définitions 36
V.2. Applications 37
V.2.1. Système d'équations diophantiennes 37
V.2.2. Forme normale de Smith 38
V.3. Algorithmes séquentiels 38
V.3.1. Algorithmes ligne par ligne 38
V.3.2. Algorithme mineur par mineur 46
V.3.3. Algorithme modulo d 49
V.4. Conclusion 53

CHAPITRE VI FORME NORMALE D'HERMITE SUR LES POLYNOMES

VI.1. Définition 54
VI.2. Algorithmes 54
VI.3. Complexité 55
VI.3.1. Nombre d'opérations arithmétiques 57
VI.3.2. Grossissement des coefficients intermédiaires 57
VI.3.3. Implantation 60
VI.4. Algorithme mineur par mineur avec cumul 61
VI.4.1. Principe de l'algorithme 61
VI.4.2. Algorithme 63
VI.4.3. Complexité 65
VI.4.4. Validité de l'algorithme et parallélisation 67
VI.5. Applications 68
VI.5.1. Formes normales de Smith et de Jordan 68
VI.5.2. Application à l'automatique 71

BIBLIOGRAPHIE

ANNEXE

INTRODUCTION

Dans cette première partie, nous présentons la forme normale d'Hermite pour des matrices à coefficients dans un anneau euclidien.

Dans un premier temps, nous rappelons les concepts et les propriétés sur lesquels sont basés les algorithmes que nous décrivons par la suite.
Nous introduisons alors la forme normale d'Hermite dans un anneau euclidien. Une preuve constructive de l'existence de cette forme est donnée. Deux cas particuliers sont ensuite développés.

Nous nous intéressons tout d'abord au cas des matrices à coefficients entiers. Dans le cadre de notre problème, nous décrivons brièvement la théorie des réseaux sur les entiers et les problèmes qui lui sont attachés. La forme normale d'Hermite sur les entiers répond à l'un de ces problèmes, à savoir l'obtention d'une certaine forme canonique du réseau engendré par les colonnes de la matrice initiale.

Les algorithmes séquentiels existants pour la résolution d'équations diophantiennes et pour le calcul de la forme normale d'Hermite sont décrits. Une étude comparative des complexités est effectuée, en tenant compte de la croissance des coefficients intermédiaires. Nous verrons qu'elle joue un rôle primordial dans le choix des algorithmes.
Nous étudions ensuite le cas des matrices à coefficients dans $\mathbb{Q}[x]$. Les algorithmes séquentiels sont calqués sur ceux proposés dans le cas entier. Deux paramètres importants entrent en jeu: la taille des rationnels mais aussi le degré des polynômes des coefficients intermédiaires. Ces deux quantités doivent être prises en compte dans le calcul des complexités. Nous verrons que le meilleur algorithme n'est pas forcément le même dans le cas entier et dans le cas polynomial.

CHAPITRE I

QUELQUES NOTIONS ET DÉFINITIONS DE BASE D'ALGEBRE

Dans ce chapitre nous rappelons quelques notions d'algèbre qui sont à la base des problèmes et des algorithmes que nous développerons par la suite [GED84], [NEW72]. Nous introduisons, en particulier, la notion d'anneau euclidien et décrivons quelques propriétés importantes associées à ces anneaux qui expliquent le choix des ensembles dans lesquels nous travaillons.

I.1. Anneaux Intègres

Définition:
Un anneau intègre et unitaire A est un anneau commutatif sans diviseur de zéro et possédant un élément unité (noté 1).

Les éléments inversibles:
Si u est un élément de A^* tel que uv = 1 pour un certain v dans A, alors u est un élément inversible de A. L'ensemble des éléments inversibles de A forme un groupe multiplicatif U(=U(A)=groupe des unités de A).

Exemples: dans $\mathbb{Z}$ les éléments inversibles sont 1 et -1; dans un corps $\mathbb{K}$ les éléments inversibles sont les éléments non nuls et dans $\mathbb{K}[x]$, les éléments inversibles sont les éléments non nuls de $\mathbb{K}$.

Un élément u de A est dit *associé* à un élément v de A si u = vw, où w appartient à U. L'ensemble des éléments associés à u forme une classe d'équivalence sur A pour la relation *est associé à*. La classe contenant 0, est constituée uniquement de 0 et la classe contenant 1 est U.

Un ensemble d'éléments de A constitué d'un élément et d'un seul de chaque classe est dit *système de représentants canoniques*. A chaque classe on associe un élément représentatif. Par convention, on choisit toujours 1 comme représentant de sa classe.

Divisibilité et pgcd:
Soient u et v des éléments de A, v non nul, on dit que u *divise* v (noté u/v) si v = uw pour un élément w de A. Les éléments inversibles sont des diviseurs triviaux de tout élément de A.

Soient u et v deux éléments de A non tous deux nuls, un élément g de A est appelé un *pgcd* de u et v si g/u et g/v et g est un multiple de tous les éléments qui divisent à la fois u et v.

Remarquons que le pgcd, s'il existe, n'est en général pas unique. Pour assurer l'unicité on impose à g d'appartenir à un certain système de représentants canoniques.

Théorème: soient u_1, u_2 deux éléments de A non tous deux nuls. Soient g_1 et g_2 deux éléments de A, si g_1 et g_2 sont deux pgcd de (u_1, u_2) alors g_1 et g_2 sont associés.

Preuve:
Si g_1 et g_2 sont deux pgcd distincts de (u_1, u_2) alors g_1 est un multiple de g_2 car g_2 divise u_1 et u_2; g_2 est un multiple de g_1 car g_1 divise u_1 et u_2.
Donc, il existe α et β tels que $g_1 = \alpha\, g_2$ et $g_2 = \beta\, g_1$.
D'où $\beta\, g_1 = \beta\, \alpha\, g_2 = g_2$ et $(\beta\, \alpha - 1)\, g_2 = (\beta\, \alpha - 1)\, g_1 = 0$. On en déduit β inversible (car A est intègre) et g_1 et g_2 sont donc associés.

Définition: si g est un pgcd de (u_1, u_2) et si g est le représentant canonique de sa classe, alors g est le pgcd de (u_1, u_2).

C'est la définition que nous prendrons pour la suite.

Congruence:
Soit m un élément non nul de A. On dit que u *est congru à* v modulo m (noté $u \equiv v \bmod m$) si m divise (u - v). La relation *est congru à* est une relation d'équivalence sur A. Un ensemble d'éléments de A formé d'un élément et un seul de chaque classe est dit système complet de résidus modulo m.

I.2. Anneaux Factoriels

Définition: un élément p non nul d'un anneau intègre A est dit premier (ou irréductible) si p n'est pas un élément inversible, et si p = uv implique u ou v est un élément inversible.

Définition: deux éléments u et v de A sont dits premiers entre eux si

$$\text{pgcd}(u,v) = 1.$$

Définition: un anneau intègre A est un anneau factoriel si pour tout u dans $A\backslash\{0\}$, soit u est un élément inversible, soit u peut être exprimé comme un produit fini d'éléments premiers (i.e. $u = p_1 p_2 \ldots p_n$ où les p_i , $1 \le i \le n$, sont premiers) tel que cette factorisation en éléments premiers est unique à réordonnement et à multiplication par des éléments inversibles près.

Remarquons qu'un anneau intègre n'est pas forcément un anneau factoriel et que dans un anneau intègre, le pgcd n'existe pas forcément pour tout couple d'éléments.

Exemple([GED84]): Soit S le sous ensemble de C défini par
$S = \{a + b\sqrt{5}\, i : a, b \in \mathbb{Z}\}$
S est un anneau intègre. Dans cet anneau, l'élément 147 peut se factoriser de deux façons différentes en produits d'éléments premiers.

$$147 = 7^2 . 3 = 7.(1 - 2\sqrt{5}\, i).(1 + 2\sqrt{5}\, i).$$

D'autre part l'élément $21 - 42\sqrt{5}\, i$ se factorise en

$$21 - 42\sqrt{5}\, i = 3.7.(1 - 2\sqrt{5}\, i)$$

D'après la première factorisation de 147, 21 serait le pgcd des deux éléments. Mais d'après la seconde factorisation, le pgcd serait $7.(1 - 2\sqrt{5}\, i)$.
Or 21 n'est pas un multiple de $7.(1 - 2\sqrt{5}\, i)$, et $7.(1 - 2\sqrt{5}\, i)$ n'est pas multiple de 21. Donc aucun de ces deux nombres n'est pgcd et 147 et $21 - 42\sqrt{5}\, i$ n'admettent pas de pgcd.

Théorème: si A est un anneau factoriel, et si u et v sont deux éléments non nuls de A, alors pgcd(u, v) existe et est unique.

I.3. Anneaux Euclidiens

Ces anneaux présentent un intérêt tout particulier de par leur propriété de divisibilité. Pour les problèmes que nous devrons traiter par la suite, nous nous restreindrons aux anneaux euclidiens.

Définition: un anneau intègre E est un anneau euclidien s'il existe une fonction v à valeur entière positive définie sur les éléments non nuls de E et ayant les propriétés suivantes:
(i) $v(ab) \geq v(a)$ pour tout a, b de E-{0}
(ii) si a, b sont deux éléments de E et $b \neq 0$, alors il existe q et r éléments de E tels que

$$a = qb + r$$

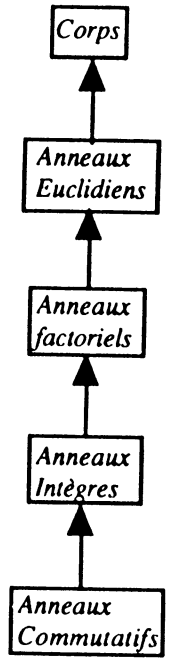
et soit r=0, soit $r \neq 0$ et $v(r) < v(b)$. v est encore appelée *valuation de E.*

Exemples: dans Z, on peut prendre $v(a) = |a|$, la valeur absolue de a;
pour les polynômes, la valuation peut être le degré.

Théorème: tout anneau euclidien est un anneau factoriel. Donc le pgcd existe et est unique.

Théorème: soit E un anneau euclidien, si a et b sont éléments de E et si g = pgcd(a , b) alors il existe s et t dans E tels que g = sa + tb.
Remarque: ce théorème se généralise au pgcd de n éléments.

En récapitulant, nous avons l'inclusion des ensembles suivante:



Nous ne nous intéresserons donc qu'aux corps et aux anneaux euclidiens afin de posséder suffisamment d'outils pour résoudre les problèmes que nous nous posons.

CHAPITRE II

FORME NORMALE D'HERMITE SUR UN ANNEAU EUCLIDIEN

Après quelques rappels sur les matrices à coefficients dans un anneau euclidien, nous définissons la forme normale d'Hermite et donnons une preuve d'existence constructive de cette forme.

II.1. Matrices sur un anneau euclidien

Soit E un anneau euclidien, on notera E_n l'anneau des matrices d'ordre n à coefficients dans E.

Les éléments inversibles de E_n :

Définition: une matrice M de E_n est un élément inversible si et seulement si il existe une matrice N de E_n telle que MN=NM=I (où I est la matrice nxn avec l'élément unité de E sur la diagonale et des zéros partout ailleurs).

Si MN=I, alors dét(M) dét(N) = 1, d'où dét(M) est un élément inversible de E.

Si dét(M) est inversible dans E, la matrice

$$N = (\text{dét}(M))^{-1} \; M^{adj}$$

appartient à E_n et satisfait MN = NM = I.

Théorème: M est un élément inversible de E_n si et seulement si dét(M) est un élément inversible de E.

Les matrices inversibles de E_n sont appelées *matrices unimodulaires* de E_n. Elles forment un groupe multiplicatif.

Quelques exemples:

(1) E=$\mathbb{Z}$, l'anneau des entiers. Les matrices unimodulaires sont les matrices de E_n de déterminant ± 1.
(2) E=K, un corps. Les matrices unimodulaires de E_n sont les matrices non singulières.
(3) E=K[x]. Les matrices unimodulaires de E_n sont les matrices de déterminant non nul élément de K.

Nous considérerons par la suite des matrices unimodulaires élémentaires correspondant aux *opérations élémentaires de colonnes* , sur une matrice M de E_n, suivantes:

1. Echange de deux colonnes
2. Multiplication d'une colonne par un élément unité u de E
3. Ajout d'un multiple m d'une colonne à une autre, où m est élément de E.

Sur les colonnes i et j elles correspondent à la multiplication à droite par une des matrices :

$$\begin{pmatrix} I_{p_1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & I_{p_2} & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & I_{p_3} \end{pmatrix} \quad \begin{pmatrix} I_{p_1} & 0 & 0 & 0 & 0 \\ 0 & u & 0 & 0 & 0 \\ 0 & 0 & I_{p_2} & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & I_{p_3} \end{pmatrix} \quad \begin{pmatrix} I_{p_1} & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & m & 0 \\ 0 & 0 & I_{p_2} & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & I_{p_3} \end{pmatrix}$$

où I_k est la matrice identité de dimension kxk et p_1= i-1, p_2= j-i-1 , p_3=n-j.

Toute matrice unimodulaire est un produit de matrices élémentaires de ce type.

Matrices équivalentes :

Définition:
soit A, B deux matrices de E_n, on dit qu'elles sont équivalentes à droite si il existe une matrice unimodulaire U de E_n telle que B=AU; équivalente à gauche s'il existe une matrice unimodulaire V de E_n telle que B=VA. B est équivalente à A si il existe U et V dans E_n telles que B=VAU.

II.2. Forme normale d'Hermite [HER51]

Définition:
Soit $H=(h_{ij})_{i=1,\ldots,n;j=1,\ldots,n}$ une matrice de E_n d'ordre n. H est dite sous Forme Normale d'Hermite si

(i) $h_{ij} = 0$, pour tout $j > i$,

(ii) tous les éléments diagonaux appartiennent à un système de représentants canoniques,

(iii) les éléments à gauche de la diagonale appartiennent à un système complet de résidus modulo les éléments diagonaux.

Remarque: les deux dernières conditions permettent d'assurer l'unicité.

Exemples:

- E=$\mathbb{Z}$, on impose que les éléments de la Forme Normale d'Hermite soient positifs
- E=$\mathbb{Q}$[x], on impose que tous les éléments diagonaux soient unitaires.

Définition: soit A une matrice de E_n. On appelle *rang* de A la dimension du plus grand mineur de A de déterminant non nul.

Théorème: toute matrice de E_n de rang n peut être mise sous forme normale d'Hermite par une suite de transformations unimodulaires élémentaires.

Preuve:

♦ Soit A une matrice de E_n. Montrons tout d'abord que A peut être rendue triangulaire inférieure par une suite de transformations élémentaires.

L'assertion est vraie pour n = 1. Supposons-la vraie jusqu'au rang n-1.

Considérons la première ligne de A et supposons qu'elle possède au moins un élément non nul. Choisissons un élément non nul de valuation minimale et positionnons le en position (1, 1) par une permutation de colonnes. En soustrayant des multiples de cette première colonne aux autres colonnes on peut rendre les n-1 derniers éléments de la première ligne soit nuls soit de valuations inférieures à celle de l'élément diagonale.
En répétant ce processus un nombre fini de fois on fait apparaître des zéros sur la première ligne, à l'exception de l'élément en position (1, 1). Soit A_1 la matrice obtenue.

D'après l'hypothèse de récurrence au rang n-1, la matrice privée de sa première ligne et de sa première colonne peut être rendue triangulaire inférieure par des transformations élémentaires. Notons V le produit de ces transformations. Considérons la matrice M de dimension nxn possédant un un en position (1, 1), des zéros partout ailleurs sur la première ligne et la première colonne, et V en mineur inférieur. Cette matrice est unimodulaire et A_1M est sous forme normale d'Hermite.
On en déduit que l'hypothèse est vérifiée au rang n.

♦ La matrice A est donc équivalente à droite à une matrice triangulaire inférieure T.
Pour que les éléments diagonaux appartiennent à un système de représentants canoniques, il suffit de multiplier T par une matrice diagonale $D = [\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n]$, pour des ε_i inversibles bien choisis dans E.

♦ On effectue finalement des transformations du type 2. pour que les éléments à gauche de la diagonale appartiennent à un système complet de résidus modulo l'élément diagonal, en parcourant les lignes de haut en bas et les colonnes de gauche à droite.

Corollaire: si A est une matrice de E_n non singulière, alors la forme normale d'Hermite de A est unique.

Preuve:
Supposons que A soit équivalente à droite à H et équivalente à droite à K avec:
$H = (h_{ij})_{i,j=1..n}$ et $K = (k_{ij})_{i,j=1..n}$ sous forme normale d'Hermite.
Alors, il existe U_1 et U_2 unimodulaires telles que $H = AU_1$ et $K = AU_2$. Par suite, il existe une matrice $U = (u_{ij})_{i,j=1..n}$ unimodulaire telle que $H = KU$.

♦ Montrons tout d'abord que les éléments diagonaux de U sont égaux à 1.
$dét(K).U = K^{adj}. H$
H et K étant non singulières et triangulaires inférieures, U est aussi triangulaire.
$dét(U) = u_{11}\, u_{22} \ldots u_{nn}$ est un élément inversible de E, chaque u_{ii} étant un élément inversible de E. Mais $h_{ii} = u_{ii}\, k_{ii}$, et h_{ii} est associé à k_{ii} . Donc, puisque l'élément diagonal doit être le représentant canonique, $u_{ii} = 1$ et $h_{ii} = k_{ii}$.

♦ Montrons que U = I. Soit s tel que $2 \le s \le n$.

$$h_{s+1,s} = \sum_{t=1}^{n} k_{s+1,t}\, u_{t,s} = \sum_{t=s}^{s+1} k_{s+1,t}\, u_{t,s}$$

car U et K sont triangulaires.
D'où

$$h_{s+1,s} = k_{s+1,s} + k_{s+1,s+1}\, u_{s+1,s} \text{ , car } u_{s,s} = 1.$$

On alors

$$h_{s+1,s} = k_{s+1,s} \mod k_{s+1,s+1}$$

et donc, puisque ils doivent appartenir au même système de résidus modulo $k_{s+1,s+1}$, on a $h_{s+1,s} = k_{s+1,s}$. D'où l'on déduit $u_{s+1,s} = 0$.
Supposons maintenant que $u_{s+i,s} = 0$, pour i tel que $1 \le i < j$, et montrons que $u_{s+j,s} = 0$.

$$h_{s+j,s} = \sum_{t=1}^{n} k_{s+j,t}\, u_{t,s} = \sum_{t=s}^{s+j} k_{s+j,t}\, u_{t,s} = k_{s+j,s}u_{s,s} + k_{s+j,s+j}\, u_{s+j,s}$$

D'où

$$h_{s+j,s} = k_{s+j,s} \mod k_{s+j,s+j}$$

Pour les mêmes raisons que précédemment on a donc

$$u_{s+j,s} = 0.$$

Remarque: la forme normale d'Hermite de A est une matrice équivalente à droite à A.

II.3. Forme normale de Smith [NEW72]

Nous introduisons ici une forme normale diagonale qui s'obtient par des transformations élémentaires sur les colonnes mais aussi sur les lignes de A.

Théorème: toute matrice A, non singulière de E_n, est équivalente à une matrice diagonale

$$S = S(A) = \text{diag}(s_1, s_2, \ldots, s_n)$$

où $s_1, s_2, \ldots, s_n$ sont des éléments non nuls de E appartenant à un système de représentants canoniques, et s_i / s_{i+1}, $1 \leq i \leq r-1$.
$s_1, s_2, \ldots, s_n$ sont appelés les *facteurs invariants* de A et S est appelée *forme normale de Smith de A*.

Théorème: deux matrices A, B de E_n sont équivalentes si et seulement si elles ont les mêmes facteurs invariants.

CHAPITRE III

RÉSEAUX

Ce chapitre est une brève introduction à la géomètrie des nombres de Minkowski et à ses applications.
Nous introduisons la notion de base d'un réseau et le lien qui existe entre le déterminant et une base du réseau avant de rappeler différents problèmes sur les réseaux. Nous décrivons ensuite une notion de réduction et des algorithmes permettant le calcul d'une base réduite, dûs à Lovasz [LOV86].
Ce chapitre est une introduction au chapitre sur la forme normale d'Hermite d'une matrice à coefficients entiers. Nous verrons que le calcul de cette forme normale permet d'obtenir une représentation canonique du réseau entier associé à la matrice initiale.

III.1. Définitions

$\mathbb{R}^n$ est muni de sa structure euclidienne. On notera ||a|| la norme de a et a.b le produit scalaire de a par b.

Définition: Soient $a_1, a_2, \ldots, a_m$, m ($m \leq n$) vecteurs linéairement indépendants de $\mathbb{R}^n$. L'ensemble des vecteurs

$$\lambda_1 a_1 + \lambda_2 a_2 + \ldots + \lambda_m a_m$$

où $\lambda_1, \lambda_2, \ldots, \lambda_m$ parcourent les entiers, est appelé *réseau* engendré par $(a_1, a_2, \ldots, a_m)$ et est noté $\mathfrak{R}(a_1, a_2, \ldots, a_m)$.

m est appelé *rang* ou *dimension* du réseau.

Soit $A = [a_1\ a_2\ \ldots\ a_m]$, $a_i \in \mathbb{R}^n$, i = 1, ..., m. Le réseau engendré par A est l'ensemble de toutes les combinaisons linéaires des colonnes de A; c'est à dire:

$\mathfrak{R}(A) = \{\ Ax$; x vecteur colonne de dimension n, à coefficients entiers $\}$.

Quelques propriétés [NEW72]:

♦ Un réseau de $\mathbb{R}^n$ de rang n est un $\mathbb{Z}$-module de $\mathbb{R}^n$ de rang n. C'est un sous-groupe discret de $\mathbb{R}^n$.

♦ Tout sous ensemble d'un réseau qui est un $\mathbb{Z}$-module non réduit à zéro est aussi un réseau.

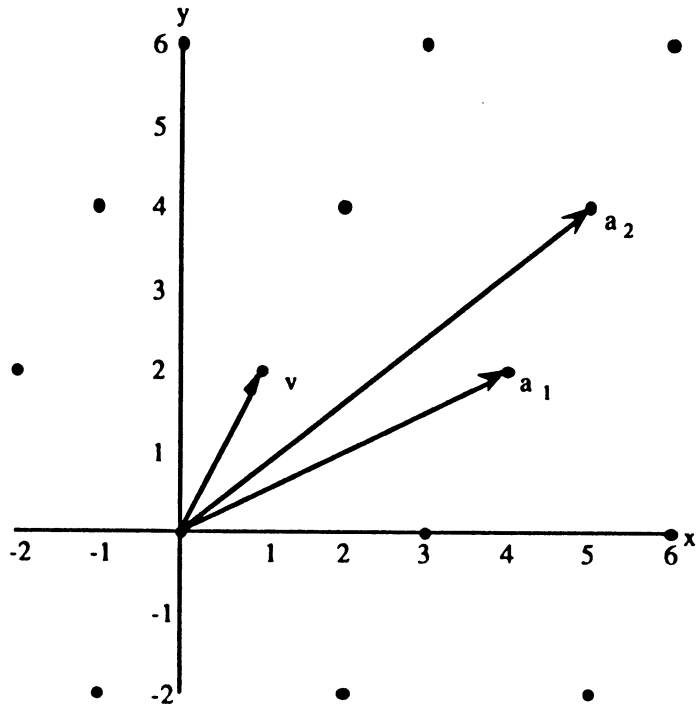


Figure III.1 : $\mathfrak{R}(a_1, a_2)$.

III.2. Bases

Définition: une base de $\mathfrak{R}$ est un sous ensemble de vecteurs de $\mathfrak{R}$ linéairement indépendants qui engendre $\mathfrak{R}$.

Soit $\mathfrak{R}$un réseau de dimension n, et $(b_1 , b_2 , \dots , b_n)$ et $(a_1, a_2 , \dots ,a_n)$ deux bases de $\mathfrak{R}$. Alors chaque b_i est une combinaison linéaire de $(a_1, a_2 , \dots ,a_n)$ et chaque a_i est une combinaison linéaire de $(b_1 , b_2 , \dots , b_n)$. Donc il existe deux matrices M et N d'ordre n telles que

$$B=AM \text{ et } A=BN,$$

où A est la matrice $[a_1, a_2 , \dots ,a_n]$ et B la matrice $[b_1 , b_2 , \dots , b_n]$.
Donc

$$A = AMN$$

et puisque A est non singulière, MN = I et dét(M) = dét(N) = $\pm$ 1

III.3. Lien avec le déterminant

On notera Δ = |déterminant ($\mathfrak{R}$)| . D'après le résultat précédent, Δ est un invariant de $\mathfrak{R}$.
Géométriquement, tous les parallépipèdes construits sur une base du réseau ont même volume. Ce volume est Δ.

Reprenons l'exemple précédent. Les trois surfaces hachurées sur la figure qui suit ont même aire et cette aire est égale au déterminant du réseau .

$$\Delta = \text{dét}([a_1, a_2]) = \begin{vmatrix} 4 & 5 \\ 2 & 4 \end{vmatrix} = 6$$

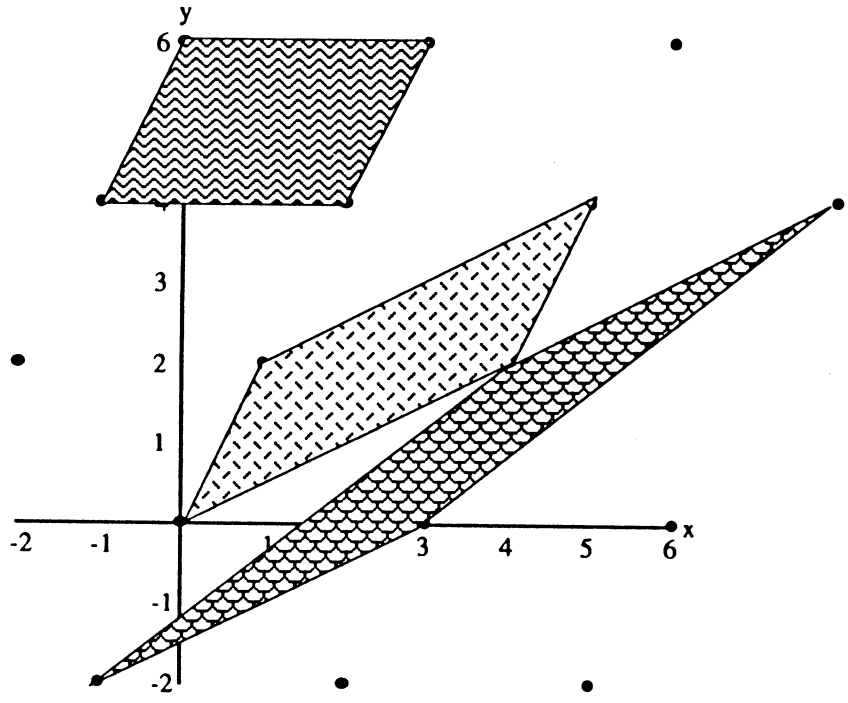


Figure III.2 : |dét($\mathfrak{R}$)|

III.4. Problèmes fondamentaux

Nous donnons ci-dessous les principaux problèmes sur les réseaux :

($\mathfrak{P}_1$) le problème de l'existence d'une solution entière pour l'équation Ax = b est équivalent à déterminer si b est un élément du réseau $\mathfrak{R}$(A). Trouver une solution de ce système revient à donner une représentation de b comme combinaison linéaire des colonnes de A.

($\mathcal{P}_2$) le problème du point le plus proche :

$$\text{trouver} \quad \underset{y \in \mathcal{R}(A)}{\text{Min}} \{\|b-y\|\}$$

($\mathcal{P}_3$) le problème du vecteur le plus court (cas particulier du problème précédent) :

$$\text{trouver} \quad \underset{y \in \mathcal{R}(A),\ y \neq 0}{\text{Min}} \{\|y\|\}$$

($\mathcal{P}_4$) approximation diophantienne simultanée:
Etant donnés n rationnels $\alpha_1, \alpha_2, \ldots, \alpha_n, \varepsilon > 0$, et un entier $Q > 0$, trouver, s'ils existent, $(n + 1)$ entiers $p_1, p_2, \ldots, p_n$ et $q > 0$, tels que

$$0 < q \leq Q \ , \ |p_i - q\,\alpha_i| \leq \varepsilon \quad (i = 1, \ldots, n).$$

Nous verrons que pour résoudre ce dernier problème nous sommes conduits à la résolution du problème ($\mathcal{P}_3$).

Tous ces problèmes sont directement liés à la recherche d'une représentation appropriée du réseau $\mathcal{R}(A)$.

III.5. Bases réduites [LOV86]

Nous construisons ici une base de $\mathcal{R}(A)$, appelée base réduite et nous indiquerons comment l'utiliser pour résoudre le problème du plus court vecteur.

Procédé d'orthogonalisation de Gram-Schmidt:

En général, un réseau ne possède pas de base orthogonale. Le procédé d'orthogonalisation de Gram-Schmidt associe à une base B d'un réseau de $\mathbb{R}^n$ une base orthogonale B^* du $\mathbb{Q}$-espace vectoriel engendré par B. Mais, en général, cette base n'appartient pas au réseau $\mathcal{R}(B)$.

Soit $B = (b_1, b_2, \ldots, b_n)$ une base de $\mathcal{R}(B)$.

Posons:

$$\begin{cases} b_1^* = b_1 \\ b_2^* = b_2 - \alpha_{21}\, b_1^* \\ \cdots \\ b_k^* = b_k - \sum_{j=1}^{k-1} \alpha_{kj}\, b_j^* \quad (k = 3, \ldots, n). \end{cases}$$

où $\alpha_{ji} = \dfrac{b_i^* \cdot b_j}{\|b_i^*\|^2}$ pour $i < j$.

Proposition:

(i) $B^* = (b_1^*, b_2^*, \ldots, b_n^*)$ est une base orthogonale pour $\mathbb{R}^n$

(ii) b_k^* est la projection de b_k sur l'orthogonal au sous espace engendré par $b_1^*, b_2^*, \ldots, b_{k-1}^*$

(iii) $|\det B| = |\det B^*| = \prod_{j=1}^{n} \| b_j^* \|$

(iv) $\Delta = |\det(b_1, b_2, \ldots, b_k)| = |\det(b_1^*, b_2^*, \ldots, b_k^*)| = \prod_{j=1}^{k} \|b_j^*\|$ pour $k < n$.

Remarque:

B^* n'est pas une base de $\mathfrak{R}$ car les α_{ij} ne sont pas entiers.

Le (ii) implique :

$$\|b_j^*\| \leq \|b_j\|$$

On en déduit la proposition suivante:

Proposition:

Pour toute base $B = (b_1, b_2, \ldots, b_n)$ de $\mathfrak{R}$, on a

$$\prod_{j=1}^{n} \|b_j\| \geq \Delta$$

Base faiblement réduite:

On cherche une base de vecteurs "assez orthogonaux". Nous allons rapprocher B de B^* en diminuant les coefficients α_{ij} sans modifier B^* ni le réseau $\mathfrak{R}(B)$.

Définition: il existe une base $(\bar{b}_1, \bar{b}_2, \ldots, \bar{b}_n)$ ayant la même orthogonalisation de Gram-Schmidt B^* que B et qui s'écrit

$$\bar{b}_i = \sum_{j=1}^{i} \bar{\alpha}_{ij} b_j^* \quad (i = 1, \ldots, n)$$

avec $|\bar{\alpha}_{ij}| \leq 1/2$ pour $1 \leq j < i \leq n$.
Une base ayant cette propriété est appelée *base faiblement réduite.*

Construction d'une base faiblement réduite:

Supposons que $B = (b_1, b_2, \ldots, b_n)$ ne soit pas faiblement réduite. Soient i et j $(1 \leq j < i \leq n)$, tels que $|\alpha_{ij}| > 1/2$ et j le plus grand possible. Soit m l'entier le plus proche de α_{ij}, alors $(b_1, \ldots, b_i, b_i - m\, b_j, b_{i+1}, \ldots, b_n)$ est une base de $\mathfrak{R}$ qui a la même orthogonalisation que B.
Si on exprime les vecteurs de base comme combinaison linéaire des vecteurs de base orthogonaux, seule la $i^{ème}$ expression change:

$$b_i - m\, b_j = \sum_{t=1}^{i} \alpha'_{it} b_t^*$$

où $\alpha'_{it} = \alpha_{it}$ pour $j < t \leq i$ et $\alpha'_{it} = \alpha_{it} - m\, \alpha_{jt}$ pour $1 \leq t \leq j$.

On a encore: $|\alpha'_{it}| \leq 1/2$ pour $j < t \leq i$, et de plus $|\alpha'_{ij}| = |\alpha_{ij} - m| \leq 1/2$.
En répétant ce processus, au plus $n(n-1)/2$ fois, on obtient une base faiblement réduite.

III.6. Problème du plus court vecteur

Etant donné un réseau L, nous noterons $\lambda(\mathfrak{R})$ la longueur du plus court vecteur non nul du réseau.

Lemme:
Si $B = (b_1, b_2, \ldots, b_n)$ est une base du réseau $\mathfrak{R}$ et B^* son orthogonalisation de Gram-Schmidt, alors

$$\lambda(\mathfrak{R}) \geq \min \{\|b_1^*\|, \ldots, \|b_n^*\|\}.$$

Preuve: soit b un élément de $\mathfrak{R}$, $b = \sum_{i=1}^{k} \lambda_i b_i$, $1 \leq k \leq n$, λ_i entier et $\lambda_k \neq 0$.

Ou encore $b = \sum_{i=1}^{k} \lambda'_i b_i^*$.

or $b_k^* = b_k - \sum_{j=1}^{k-1} \alpha_{kj} b_j^*$, donc $\lambda_k = \lambda'_k$ est un entier non nul.

D'où :

$$\|b\|^2 = \sum_{i=1}^{k} |\lambda_i'|^2 \|b_i^*\|^2 \geq |\lambda_k|^2 \|b_k^*\|^2 \geq \|b_k^*\|^2$$

Nous utiliserons le résultat précédent pour obtenir une borne inférieure sur le plus court vecteur.

Construisons, dans un premier temps, une base formée de vecteurs "assez courts" et dont l'orthogonale est constitué de vecteurs "raisonnablement longs".

Définition:
soit $\mathfrak{R}$ un réseau, B= (b_1 , b_2 , ... , b_n) une base de $\mathfrak{R}$, et B^* son orthogonalisation de Gram-Schmidt. B est une base réduite si b_1 est le plus court des b_j pour j = 1 à n, si elle est faiblement réduite et si

$$\|b_{j+1}^* + \alpha_{j,j+1}\, b_j^*\|^2 \geq \frac{3}{4}\|b_j^*\|^2 \quad \text{pour } j = 1, \ldots, n-1.$$

Une base réduite nous fournit alors une borne supérieure sur le plus court vecteur.

Théorème:
Soit B= (b_1 , b_2 , ... , b_n) une base réduite pour $\mathfrak{R}$. Alors

(i) $\|b_j^*\|^2 \leq 2\,\|b_{j+1}^*\|^2$

(ii) $\|b_1\| \leq 2^{(n-1)/4}\,(\text{dét}(\mathfrak{R}))^{1/n}$

(iii) $\|b_1\| \leq 2^{(n-1)/2}\,\lambda(\mathfrak{R})$

(iv) $\prod_{j=1}^{n} \|b_j\| \leq 2^{\,n(n-1)/4}\,\text{dét}(\mathfrak{R})$

Soit base_f_réduite(B) un algorithme qui calcule une base faiblement réduite de B et permute(B, j, j+1) un algorithme qui permute le $j^{ème}$ et le $(j+1)^{ème}$ vecteur de B. base_non_réduite est une variable booléenne qui indique si la réduction est ou n'est pas terminée.

Algorithme de construction d'une base réduite (Lovasz)
Entrée : une base $B = (b_1 , b_2 , \dots , b_n)$ de $\mathfrak{R}$
Début
base_non_réduite := 1
Tant que base_non_réduite **faire**
Début

Calculer B^* par le procédé d'orthogonalisation de Gram-Schmidt

B := base_F_réduite(B)

Si $\left\| b_{j+1}^* + \alpha_{j,j+1}\, b_j^* \right\|^2 < \frac{3}{4} \left\| b_j^* \right\|^2$ pour un certain j **alors**
B := permute(B, j, j+1)
Sinon base_non_réduite := 0
Fin
Fin.

On montre que l'algorithme s'arrête aprés un nombre polynomial d'étapes [LOV86].

Théorème (Lenstra, Lovasz): pour une base entière $(b_1 , b_2 , \dots , b_n) \in \mathbb{Z}^n$, de longueur bornée par $\|b_1\|^2 , \|b_2\|^2 , \dots , \|b_n\|^2 \le K$, l'algorithme de Lovasz nécessite au plus $O(n^4 \log K)$ opérations arithmétiques sur des entiers codés sur $O(n \log K)$ bits.

Remarquons que cet algorithme peut être utilisé pour trouver un plus petit vecteur d'un réseau.

Si $B = (b_1 , b_2 , \dots , b_n)$ est une base réduite et si $z = \sum_{i=1}^{n} \lambda_i b_i \in \mathbb{R}^n$, $\lambda_i^{n(n+1)/4} \varepsilon^{-n} \in \mathbb{Z}^n$, alors on peut montrer que $|\lambda_i| < 3^n \frac{\|z\|}{\|b_1\|}$ [LOV86].

Or nous ne sommes intéressés que par les vecteurs z tels que $\| z \| \le \| b_1 \|$. Il suffit donc d'examiner tous les vecteurs z tels que $\lambda_i \in \{-3^n , \dots, 3^n\}$.

III.7. Le problème d'approximation diophantienne simultanée

Soit $\alpha_1, \alpha_2, \dots, \alpha_n \in \mathbb{Q}$, $\varepsilon \in \mathbb{R}$, tel que $0 < \varepsilon < 1$ et $Q > 0$. Nous allons voir que le problème d'approximation diophantienne simultanée peut être transformé en un problème de plus court vecteur.

Considérons la matrice :

$$A = \begin{pmatrix} 1 & & 0 & \alpha_1 \\ & \ddots & & \vdots \\ 0 & & 1 & \alpha_n \\ & & & \varepsilon/Q \end{pmatrix}$$

Soit $b \in \mathfrak{R}(A)$. Alors $b = Ap$ où $p = {}^t(p_1, p_2, \ldots, p_{n+1}) \in \mathbb{Z}^n$. Supposons $b \neq 0$ et $\|b\| \leq \varepsilon$. On a alors $p_{n+1} \neq 0$, et on peut supposer $p_{n+1} < 0$. Soit $q = -p_{n+1}$, alors

$$|b_i| = |p_i - \alpha_i q| \leq \varepsilon \ (i = 1, \ldots, n)$$

$$|b_{n+1}| = \varepsilon\, q/Q \leq \varepsilon.$$

$q \leq Q$, donc "un vecteur court" de $\mathfrak{R}(A)$ donne une bonne approximation de $\alpha_1, \alpha_2, \ldots, \alpha_n$.

Si on choisit $Q = 2^{n(n+1)/4}\varepsilon^{-n}$, on peut alors trouver, en un temps polynomial, un vecteur $b \in \mathfrak{R}(A)$ tel que $b \neq 0$ et

$$\|b_1\| \leq 2^{n/4} \left|\text{dét}(\mathfrak{R}(A))\right|^{1/(n+1)} = 2^{n/4}\left(\frac{\varepsilon}{Q}\right)^{1/(n+1)}$$

et donc $\|b\|_\infty \leq \|b\| \leq \varepsilon$.

D'où le théorème:

Théorème: étant donnés $\alpha_1, \ldots, \alpha_n \in \mathbb{Q}$, $0 < \varepsilon < 1$, on peut trouver, en un temps polynomial, des entiers $p_1, p_2, \ldots, p_n$ et q tels que

$$\begin{cases} |p_i - q\alpha_i| \leq \varepsilon \\ 0 < q \leq 2^{n(n+1)/4}\varepsilon^{-n}. \end{cases}$$

III.8. Conclusion

Nous avons construit dans ce chapitre une base B de $\mathfrak{R}(A)$ appelée base réduite et nous avons indiqué comment l'utiliser pour résoudre le problème du plus court vecteur et le problème d'approximation diophantienne simultanée.

Dans les deux chapitres qui suivent, après une présentation des équations diophantiennes nous définissons une forme normale d'Hermite de $\mathfrak{R}(A)$ ou encore base d'Hermite de $\mathfrak{R}(A)$, où A est une matrice à coefficients entiers, qui permet de résoudre un système d'équations diophantiennes.

CHAPITRE IV

EQUATIONS DIOPHANTIENNES

Nous nous intéresserons dans ce chapitre aux algorithmes qui permettent la résolution d'équations diophantiennes. Nous verrons par la suite que ces algorithmes peuvent être repris pour certaines étapes du calcul de la forme normale d'Hermite des matrices à coefficients entiers.
Nous montrerons également dans le chapitre suivant que les deux problèmes sont directement liés: le calcul de la forme normale d'Hermite d'une matrice A permet de résoudre, si une solution existe, le système d'équations diophantiennes associé aux coefficients de A.

IV.1. Généralités

Soit l'équation

$$(1)\quad a_1 x_1 + a_2 x_2 + \ldots + a_n x_n = b$$

où $a_1, a_2, \ldots, a_n$, b sont des entiers. Cette équation admet une solution entière si et seulement si le pgcd des a_i divise b.

Considérons l'application linéaire:

$$\begin{aligned} \phi : \quad \mathbb{Z}^n &\rightarrow \mathbb{Z} \\ (x_1, x_2, \ldots, x_n) &\mapsto a_1 x_1 + a_2 x_2 + \ldots + a_n x_n \end{aligned}$$

ϕ est un homomorphisme de $\mathbb{Z}$-module. Le noyau Kerϕ de ϕ est un sous-module de $\mathbb{Z}^n$ de dimension n-1.

Il existe n-1 solutions indépendantes, $s_1, s_2, \ldots, s_{n-1}$ dans $\mathbb{Z}^n$, de l'équation homogène telles que :

$$\text{Ker } \phi = \{\lambda_1 s_1 + \lambda_2 s_2 + \ldots + \lambda_{n-1} s_{n-1} / \lambda_1, \lambda_2, \ldots, \lambda_{n-1} \in \mathbb{Z}\}.$$

$s_1, s_2, \ldots, s_{n-1}$ forment une base du $\mathbb{Z}$-module Ker ϕ. Ker $\phi = \mathfrak{R}(s_1, s_2, \ldots, s_{n-1})$.

Théorème: soit b=d le pgcd des a_i (i= 1, ..., n). Soit $x_0 \in \mathbb{Z}^n$ une solution particulière de (1). La solution générale $x \in \mathbb{Z}^n$ de (1) peut être mise sous la forme :

$$x = x_0 + f \quad f \in \text{Ker } \phi.$$

Preuve:
Si x est une solution de (1), on a $\phi(x - x_0) = 0$. Donc, $x - x_0 \in \text{Ker}\,\phi$.

Théorème [SCH85]: si b = cd, avec c dans $\mathbb{Z}$, alors la solution générale de (1) est:

$$x = (x_0.c) + \lambda_1 s_1 + \lambda_2 s_2 + \ldots + \lambda_{n-1} s_{n-1}$$

où x_0 est une solution particulière de l'équation (1).

IV.2. Algorithmes

IV.2.1. Algorithme d'Euclide étendu [KNU81]

Soit a et b deux entiers $|a| \geq |b|$. On cherche deux entiers x et y satisfaisant l'équation:

$$(2) \qquad a\,x + b\,y = \text{pgcd}(a, b)$$

Supposons a et b positifs. On notera $\left\lfloor \frac{a}{b} \right\rfloor$ le quotient de la division euclidienne de a par b, soit l'entier positif tel que $a = b\,q + r$ avec $0 \leq r < b$.

Considérons la matrice :

$$A_0 = \begin{pmatrix} a & 1 & 0 \\ b & 0 & 1 \end{pmatrix}$$

On définit par récurrence une suite de matrices $\left(A_k = \begin{pmatrix} a_k & c_k & e_k \\ b_k & d_k & f_k \end{pmatrix}\right)_{k \in N}$ par :

$$(I) \begin{cases} A_{2k+1} = \begin{bmatrix} 1 & -\left\lfloor \frac{a_{2k}}{b_{2k}} \right\rfloor \\ 0 & 1 \end{bmatrix} A_{2k} \\ A_{2(k+1)} = \begin{bmatrix} 1 & 0 \\ -\left\lfloor \frac{b_{2k+1}}{a_{2k+1}} \right\rfloor & 1 \end{bmatrix} A_{2k+1} \end{cases}$$

Ce processus est itéré jusqu'à obtention d'une matrice A_N telle que a_N ou b_N soit nul.
Supposons A_N telle que $b_N = 0$:

$$A_N = \begin{pmatrix} g & c_N & e_N \\ 0 & d_N & f_N \end{pmatrix}$$

Alors g est le pgcd de a et b.
N.B.: g et 0 sont intervertis dans le cas $a_N = 0$.

A_N est obtenue en multipliant A_0 à gauche par une suite de matrices 2x2 de déterminant ± 1. Le produit de ces matrices est la matrice :

$$\begin{pmatrix} c_N & e_N \\ d_N & f_N \end{pmatrix}$$

où c_N, d_N, e_N, f_N vérifient

$$\begin{cases} c_N a + e_N b = g \\ d_N a + f_N b = 0 \end{cases}$$

On a donc une solution (c_N , e_N) de l'équation (2) et une solution (d_N , f_N) de l'équation homogène associée.
La solution générale de l'équation (2) s'écrit:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} c_N + \lambda d_N \\ e_N + \lambda f_N \end{pmatrix}$$

avec λ dans $\mathbb{Z}$
Remarque:
Supposons que la matrice initiale choisie pour (I) soit:

$$A_0 = \begin{pmatrix} a & 1 \\ b & 0 \end{pmatrix}$$

on obtient alors les coefficients e_N et f_N :

$$\begin{cases} e_N = \dfrac{g - c_N a}{b} \\ f_N = -\dfrac{d_N a}{b} \end{cases}$$

Supposons maintenant a et b de signes quelconques. Soit (c_N , e_N, d_N , f_N) les coefficients obtenus pour |a| et |b|. La solution générale de l'équation (2) s'écrit:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \text{sign}(a) (c_N + \lambda d_N) \\ \text{sign}(b) (e_N + \lambda f_N) \end{pmatrix}$$

Théorème[ROC89]:

$$\text{Max}(|c_n|, |e_n|) \leq \text{Max}(|a|, |b|)$$

Complexité

Si l'algorithme présenté ci-dessus nécessite N itérations, de l'ordre de 2N multiplications, N divisions et 2N additions sont effectuées.

Théorème [LAM44]: Le nombre d'itérations de l'algorithme d'Euclide sur deux entiers positifs est borné par le nombre de bits du plus petit multiplié par la constante $\log_\phi 2$ (où $\phi = (1 + \sqrt{5})/2$).

Le pire cas est obtenu lorsque les deux entiers sont des itérés successifs de la suite de Fibonacci: la borne donnée ci-dessus est alors atteinte.

On a donc un algorithme en O(log (n)) étapes, où n est le plus petit des deux opérandes.

Une petite parenthèse en liaison avec le chapitre précédent

Trouver le pgcd de deux entiers a et b revient à résoudre le problème d'approximation rationnelle:
chercher une solution de l'équation

$$\min\{p: a/b - q/p = 0, p, q > 0\}.$$

Le pgcd est alors b/p.

IV.2.2. Algorithme de Blankinship [BLA66]

Cet algorithme généralise l'algorithme précédent au cas de n coefficients.

Considérons l'équation (1) du IV-1. On pose:

$$A_0 = \begin{pmatrix} a_1 & 1 & 0 & 0 & . & 0 \\ a_2 & 0 & 1 & 0 & . & 0 \\ a_3 & 0 & 0 & 1 & . & 0 \\ . & . & . & . & . & . \\ a_n & 0 & 0 & 0 & . & 1 \end{pmatrix}$$

On suppose tout d'abord les a_i positifs.

L'algorithme de Blankinship consiste à annuler les éléments de la première colonne par des opérations de lignes du même type que précédemment. Les opérations sont effectuées jusqu'à n'obtenir plus qu'un élément non nul sur la première colonne.

On notera $(A_0)_i$ la ième ligne de A_0 et $(A_0)_{i,j}$ son $j^{ième}$ coefficient, pour i=1,..., n et j=1,..., n.

Algorithme :
Début

Tant que (il existe i , $1 \le i \le n$, tel que $(A_0)_{i,1} \neq 0$) **faire**

soit k, $1 \le k \le n$, tel que $(A_0)_{k,1} = \min \{(A_0)_{j,1} / 1 \le j \le n, (A_0)_{j,1} \neq 0\}$

si (il existe j , $1 \le j \le n$, $j \neq k$, tel que $(A_0)_{j,1} \neq 0$) **faire**

$$(A_0)_j := (A_0)_j - \left\lfloor \frac{(A_0)_{j,1}}{(A_0)_{k,1}} \right\rfloor (A_0)_k$$

Fin.

On obtient finalement une matrice de la forme :

$$\begin{pmatrix} 0 & L_1 \\ . & \dots \\ g & L_k \\ . & \dots \\ 0 & L_n \end{pmatrix}$$

où L_1 , L_2 , ..., L_n sont n vecteurs ligne de dimension n et où g est le pgcd des a_i (pour i= 1 à n).
De plus, si on note $A = (a_1, a_2, \dots, a_n)$, alors on a :

$A^t . L_i^t = 0$ pour tout i dans $\{1, \dots, k-1, k+1, \dots, n\}$
et $A^t . L_k^t = g$.

$L_1, \dots, L_{k-1}, L_{k+1}, \dots, L_n$ sont donc n-1 solutions indépendantes de l'équation homogène qui fournissent une base du noyau de A.
Si g divise b, (b/g). L_k est une solution particulière de l'équation (1).

Si g divise b, on obtient alors la solution générale de l'équation (1) qui est :

$$x = \frac{b}{g} L_k + \sum_{i \neq k} \lambda_i L_i \quad , \quad (\lambda_i)_{\substack{i=1..n \\ i \neq k}} \in \mathbb{Z}$$

Complexité
Supposons que le terme non nul le plus petit soit en première position.
Supposons que le calcul de pgcd(pgcd($a_1, a_2, \dots, a_{i-1}$), a_i) nécessite K_i itérations et soit $K = \sum_{i=2}^{n} K_i$. Alors l'algorithme de Blankinship nécessite de l'ordre de 2K multiplications, K divisions et 2K additions.

Théorème [BRA70]: le nombre d'itérations pour calculer le pgcd de n entiers en appliquant l'algorithme d'Euclide récursivement est borné par (n-2) + (nombre de bits du plus petit).

On en déduit la complexité de l'algorithme de Blankinship:
Théorème: le coût de l'algorithme de Blankinship appliqué à n entiers est

$$\boxed{(n-2)\ (\text{nombre de bits du plus petit})\ (2\ T_{mul} + T_{div} + 2\ T_{add})}$$

Où T_{mul}, T_{div}, T_{add} sont respectivement le coût de la multiplication, de la division et de l'addition de deux entiers bornés par le max des opérandes.

IV.2.3. Algorithme de Bradley [BRA70]

Nous nous intéressons maintenant au calcul du pgcd et d'une solution particulière de l'équation avec le pgcd comme terme de droite.
Dans un premier temps, appliquons récursivement l'algorithme d'Euclide:

Généralisation de l'algorithme d'Euclide:

étape 1:

$g := a_1$
pour i:=2 à n faire
Appliquer Euclide pour calculer pgcd(g , a_i) et x_i , y_i tels que
$x_i\ g + y_i\ a_i = pgcd(g , a_i)$
g:=pgcd(g,a_i)

étape 2:

$X_n := y_n$
cumul := x_n
pour i:=n-1 à 2 faire
X_i := cumul.y_i
cumul := cumul.x_i
X_1 := cumul

étape 3:

retourner (X_n, X_{n-1}, ..., X_1, g)

Théorème: g est le pgcd des coefficients de départ et (X_n, X_{n-1}, ..., X_1) est solution de l'équation:

$$(1)\quad a_1 x_1 + a_2 x_2 + \ldots + a_n x_n = g$$

Preuve:
g est le pgcd puisque $pgcd(a_1, a_2, a_3) = pgcd(pgcd(a_1, a_2), a_3)$.

$$pgcd(a_1, a_2, \ldots, a_n) = y_n a_n + x_n (y_{n-1} a_{n-1} + \ldots + x_3 (y_2 a_2 + x_2 a_1)\ldots)$$
$$= y_n a_n + x_n y_{n-1} a_{n-1} + \ldots + x_n \ldots x_3 y_2 a_2 + x_n \ldots x_2 a_1$$

Cet algorithme donne une solution particulière de l'équation diophantienne.

Définition:
Soit l'équation $a_1 x_1 + a_2 x_2 + \ldots + a_n x_n = \text{pgcd}(a_1, a_2, \ldots, a_n)$. La solution est dite minimale si et seulement si:

$$\prod_{j \in S} |x_j| \leq \left(\frac{1}{2}\right)^k \frac{|a_1|}{\text{pgcd}(a_1, a_2, \ldots, a_n)}$$

où $S = \{ j \in \{1,\ldots, n\} / x_j \neq 0 \}$ et $k = \text{card}(S)$.
N.B.: il n'y a bien sûr pas unicité de la solution minimale.

Exemple (Bradley 1970):

Pour $a_1 = 424$, $a_2 = 444$, $a_3 = 932$, $a_4 = 22\,347$, l'algorithme donne la solution (122 914, -117 327, 0, -1).

(37, 15, 0, -1) est une solution minimale.

Algorithme de Bradley (construction d'une solution minimale):
étape 1:

$g := a_1$
pour i:=2 à n faire
Appliquer Euclide pour calculer $\text{pgcd}(g, a_i)$ et x_i, y_i tels que
$x_i\, g + y_i\, a_i = \text{pgcd}(g, a_i)$
$v_i := g / \text{pgcd}(g, a_i)$
$g := \text{pgcd}(g, a_i)$
$u_i := - a_i / g$

étape 2:

$X_n := y_n$
$\text{cumul} := x_n$

pour i:=n-1 à 2 faire
$X_i := \text{cumul}.y_i - v_i\, [[\text{cumul}.y_i / v_i]]$
$\text{cumul} := \text{cumul}.x_i - u_i\, [[\text{cumul}.y_i / v_i]]$
$X_1 := \text{cumul}$

étape 3:

retourner $(X_n, X_{n-1}, \ldots, X_1, g)$

où [[x]] représente l'entier le plus proche de x.

Les nombres (u_i, v_i) construits par cet algorithme sont des générateurs de toutes les solutions de l'équation $\text{pgcd}(a_1, a_2, \ldots, a_n)\, y + a_i\, z = 0$. Toutes les solutions de l'équation sont donc des multiples entiers de cette solution.
On peut prouver aisément que cette algorithme calcule une solution minimale en remplacant la valeur des X_i dans l'équation.

Cet algorithme nécessite un nombre supplémentaire linéaire en n d'opérations arithmétiques par rapport à l'algorithme de Blankinship.

Conclusion

Nous avons donné ici des algorithmes qui permettent, en un temps polynomial, de décider si une équation diophantienne admet une solution et si oui de donner la solution générale.
Nous avons remarqué, avec l'algorithme de Bradley, qu'il était possible de trouver des solutions plus ou moins "simples".

CHAPITRE V

FORME NORMALE D'HERMITE SUR LES ENTIERS

Nous présentons dans ce chapitre une représentation canonique du réseau $\mathfrak{R}(A)$, dite forme normale d'Hermite. Nous fournirons par la même un algorithme polynomial pour le problème ($\mathfrak{P}_1$) sur les réseaux.

V.1. Définitions

Définition: une matrice $H=(h_{ij})_{i=1,\ldots,m;j=1,\ldots,n}$, $m \leq n$, à coefficients entiers, non singulière est dite sous forme normale d'Hermite si

$h_{ij} = 0$, pour tout $j > i$,
$h_{ii} > 0$, pour tout i,
$h_{ij} \geq 0$ et $h_{ij} < h_{ii}$, pour tout $j < i$.

$\mathbb{Z}$ étant un anneau euclidien, tous les résultats énoncés au chapitre II peuvent être appliqués.

Proposition: si A est une matrice mxn à coefficients entiers, et C une matrice unimodulaire nxn, alors $\mathfrak{R}(AC) = \mathfrak{R}(A)$.

Preuve:
Montrons que $\mathbb{Z}^n = \{ z \in \mathbb{Z}^n / Cz \in \mathbb{Z}^n \}$
Pour tout $z \in \mathbb{Z}^n$, $Cz \in \mathbb{Z}^n$ car C est à coefficients entiers. Réciproquement, C étant unimodulaire, C^{-1} est à coefficients entiers et $C^{-1}Cz=z \in \mathbb{Z}^n$.
D'où :

$$\mathfrak{R}(A) = \{ y \in \mathbb{Z}^m / y=Ax, x \in \mathbb{Z}^n \} = \{ y \in \mathbb{Z}^m / y=ACz, z \in \mathbb{Z}^n \}$$

Théorème: soit A et A' deux matrices à coefficients entiers, de rang m et de forme normale d'Hermite respectives H et H'. Alors les colonnes de A génèrent le même réseau que les colonnes de A', si et seulement si H=H'.

Preuve:

♦ Les colonnes de A et H engendrent le même réseau et de même pour A' et H'. Donc, si H = H', il est clair que A et A' engendrent le même réseau. La condition est donc suffisante.

On en déduit l'unicité de H:

♦ Supposons que les colonnes de A et de A' engendrent le même réseau L. Alors il en est de même des colonnes de H et H', puisqu'elles sont obtenues par des transformations élémentaires. Soient h_j et h'_j les $j^{èmes}$ colonnes de H et H'. Alors h_j et h'_j sont éléments de L, donc $h_j - h'_j$ est aussi élément de L et par conséquent est une combinaison linéaire des colonnes de H. Supposons que le premier coefficient non nul de ce vecteur soit en position i et que $h_{ii} \geq h'_{ii}$. H étant triangulaire inférieure, $h_j - h'_j$ est une combinaison linéaire des colonnes i à n. En particulier $h_{ij} - h'_{ij}$ est un multiple de h_{ii} ce qui est contradictoire avec le fait que $0 < |b_{ij} - b'_{ij}| < b_{ii}$ (puisque si j = i, alors $0 < b'_{ii} < b_{ii}$, et si j < i, alors $0 \leq b_{ij} < b_{ii}$ et $0 \leq b'_{ij} < b'_{ii} \leq b_{ii}$).

Corollaire: toute matrice de rang m admet une unique forme normale d'Hermite.

Preuve: prendre A = A' dans le théorème qui précède.

V.2. Applications

V.2.1. Résolution d'un système d'équations diophantiennes

Soit A une matrice d'ordre n non singulière à coefficients entiers et b un vecteur colonne de dimension n à coefficients entiers. Nous cherchons une solution entière, si elle existe, au système Ax = b.

Considérons la matrice 2nx(n+1) suivante:

$$\begin{bmatrix} A & -b \\ I & 0_n \end{bmatrix}$$

où 0_n est un vecteur colonne nul de dimension n.

On effectue les opérations suivantes:

Etape 1:

effectuer des opérations élémentaires de façon à faire apparaître la forme normale d'Hermite de A en mineur principal sans modifier la dernière colonne.

Etape 2:

tant que cela est possible, effectuer des opérations élémentaires pour faire apparaître des zéros en dernière colonne, sans permuter celle-ci.

La matrice obtenue est de la forme:

$$\begin{bmatrix} H & k \\ U & f \end{bmatrix}$$

où H est la forme normale d'Hermite de A, U une matrice de dimension nxn et k, f deux vecteurs colonne de dimension n.

Théorème: U est la matrice unimodulaire telle que H = AU et si le système Ax = b admet une solution alors $k = 0_n$ et f est solution du système.

Preuve: voir [CHO82].

V.2.2. Forme normale de Smith

Théorème: étant donnée une matrice non singulière entière A, il existe deux matrices unimodulaires, nxn, U et V telles que D = VAU soit une matrice diagonale avec des éléments positifs sur la diagonale tels que $d_{11}/ d_{22}/ \ldots / d_{nn}$.

Kannan et Bachem [KAN79] ont proposé un algorithme calculant la forme normale de Smith d'une matrice en temps polynomial. Dans cet algorithme on alterne des calculs de forme normale d'Hermite et des éliminations d'éléments sous-diagonaux.

V.3. Algorithmes séquentiels

Les algorithmes calculant la forme d'Hermite consistent à effectuer des opérations unimodulaires sur les colonnes de A. Ces opérations sont de deux types :

- les opérations pour la triangularisation
- les opérations pour la réduction

V.3.1. Algorithmes ligne par ligne [KAN80]

Les deux algorithmes décrits ci-dessous suivent le schéma suivant:

pour i:=1 **à** n **faire**
    faire apparaître n-i zéros en ligne i
    si i≥2 **alors**
        réduire les éléments à gauche de la diagonale sur la ligne i

Nous noterons A^i la matrice obtenue à l'étape i, avec A^0=A. Les algorithmes qui suivent construisent la suite $A^0, A^1, \ldots, A^n$.

A^i est obtenue à partir de A^{i-1} par une transformation unimodulaire U^i.

Considérons deux colonnes j et k de A^{i-1}, avec $i \leq j < k$. Leurs i-1 premiers coefficients sont nuls. Pour faire apparaître un zéro en position (i,k), on peut utiliser la transformation suivante:

$$\tau_{kj}^{i} = \begin{pmatrix} p & -\frac{a_{ik}^{i-1}}{g} \\ q & \frac{a_{ij}^{i-1}}{g} \end{pmatrix}$$

où p , q et g vérifient:

$$g = p\, a_{ij}^{i-1} + q\, a_{ik}^{i-1} = pgcd\,(a_{ij}^{i-1}, a_{ik}^{i-1})$$

On remarque que la transformation est de déterminant 1.

On effectue alors l'opération:

$$\begin{pmatrix} 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ a_{ij}^{i-1} & a_{ik}^{i-1} \\ x & x \\ x & x \\ a_{nj}^{i-1} & a_{nk}^{i-1} \end{pmatrix} \begin{pmatrix} p & -\frac{a_{ik}^{i-1}}{g} \\ q & \frac{a_{ij}^{i-1}}{g} \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ g & 0 \\ x' & x' \\ x' & x' \\ a_{nj}^{i-1} & a_{nk}^{i-1} \end{pmatrix}$$

Il est clair que l'on peut triangulariser la matrice A en $n^2/2$ opérations de colonnes de ce type et $n^2/2$ opérations de réduction..

Un premier algorithme

Algorithme

```
pour i:=1 à n faire
    étape triangularisation
        pour j:=i+1 à n faire
            pour l:=i à n faire
                ( a_li  a_lj ) := ( a_li  a_lj ) τ_ij^i
    étape réduction
        pour j:=1 à i-1 faire
            q := ⌊a_ij / a_ii⌋
            pour l:=i à n faire
                a_lj := a_lj - q*a_li
```

L'exemple qui suit est extrait d'une session Macsyma. "Hermite" est une fonction que nous avons développée sous ce système pour calculer la forme normale d'Hermite par l'algorithme précédent. Elle permet de calculer la forme d'Hermite, ainsi que la matrice unimodulaire qui lui est associée.
Les paramètres d'entrée de la fonction sont la matrice initiale et sa dimension. Les résultats intermédiaires sont donnés après chaque élimination de ligne.

```
(c11) m;

                          [ 10    - 47   8    59   ]
                          [                        ]
                          [ 121   - 563  96   716  ]
(d11)                     [                        ]
                          [ 139   - 777  129  839  ]
                          [                        ]
                          [ 191   - 552  94   - 220 ]

(c12) hermite(m,4);

                % Apres elimination sur la premiere ligne %

                [   1        0        0        0    ]
                [                                    ]
                [  - 5       57       136      1011 ]
                [                                    ]
                [  385     - 1237   - 2951   - 21876 ]
                [                                    ]
                [ - 1018    3457     8238     59842 ]
                [                                    ]
                [  - 14      47       112      826  ]
                [                                    ]
                [  - 3       10       24       177  ]
                [                                    ]
                [   0        0        1        0    ]
                [                                    ]
                [   0        0        0        1    ]
```

% Apres elimination sur la seconde ligne %

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ -5 & 1 & 0 & 0 \\ 385 & -16 & -25 & -5700 \\ -1018 & -73 & 586 & 133645 \\ -14 & -1 & 8 & 1837 \\ -3 & 2 & -8 & -1845 \\ 0 & 13 & -57 & -13143 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

% Apres reduction sur la seconde ligne %

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 305 & -16 & -25 & -5700 \\ -1383 & -73 & 586 & 133645 \\ -19 & -1 & 8 & 1837 \\ 7 & 2 & -8 & -1845 \\ 65 & 13 & -57 & -13143 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

% Apres elimination sur la troisieme ligne %

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 305 & -16 & 25 & 0 \\ -1383 & -73 & -586 & 37 \\ -19 & -1 & -8 & 13 \\ 7 & 2 & 8 & -21 \\ 65 & 13 & 57 & -147 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
% Apres reduction sur la troisieme ligne %

[   1      0      0      0   ]
[                             ]
[   0      1      0      0   ]
[                             ]
[   5      9     25      0   ]
[                             ]
[ 5649   - 659  - 586    37   ]
[                             ]
[  77     - 9    - 8     13   ]
[                             ]
[ - 89     10     8     - 21  ]
[                             ]
[ - 619    70     57    - 147 ]
[                             ]
[   0      0      0      1   ]

% Apres reduction sur la quatrieme ligne %

[   1      0      0      0   ]
[                             ]
[   0      1      0      0   ]
[                             ]
[   5      9     25      0   ]
[                             ]
[   25     7      6     37   ]
[                             ]
[ - 1899   225    200    13   ]
[                             ]
[  3103  - 368  - 328   - 21  ]
[                             ]
[ 21725  - 2576 - 2295  - 147 ]
[                             ]
[ - 152    18     16     1    ]

                [ 1   0  0   0 ]  [  10   - 47    8     59  ]
                [               ]  [                        ]
                [ 0   1  0   0 ]  [ 121   - 563   96    716 ]
(d12)    list([                ], [                        ])
                [ 5   9  25  0 ]  [ - 40   181   - 31  - 236 ]
                [               ]  [                        ]
                [ 25  7  6   37 ]  [ - 18   94    - 16  - 143 ]
```

Complexité de l'algorithme:
Supposons tout d'abord que le temps d'une multiplication, et d'un calcul de pgcd sont des constantes T_{mul} et T_{pgcd}.
En ignorant le temps des additions, le coût de l'algorithme est alors:

$$\sum_{k=1}^{n-1} 4\,(n-k)^2\,T_{mul} + \frac{n^2}{2}\,T_{pgcd} + \sum_{k=1}^{n-1} k\,(n-k)\,T_{mul}$$

donc de l'ordre de $O(n^3)$ opérations arithmétiques.
Nous avons cependant négligé un aspect fondamental du calcul sur des entiers: la croissance des coefficients intermédiaires.
Nous avons vu dans l'exemple précédent que les coefficients intermédiaires grossissent très rapidement alors que la matrice finale a des coefficients bornés par la valeur absolue du déterminant.

A une étape d'élimination, nous pouvons seulement garantir:

$$\max(|p|, |q|) \leq \max(|a_{ii}|, |a_{ij}|)$$

A chaque opération, un coefficient est susceptible d'être élevé au carré. On ne connaît pas de borne sur les coefficients intermédiaires meilleure que B^{2^n}, si B est une borne sur les coefficients de A [FRU77].
On est donc confronté à une croissance des coefficients exponentielle, et les opérations arithmétiques précision infinie sont alors bien loin de pouvoir être ramenées à des constantes !!!!

Ce problème a motivé la construction de nouveaux algorithmes pour l'obtention d'une borne correcte sur la taille des coefficients intermédiaires.

Examinons de plus près l'algorithme décrit précédemment.

Plaçons nous à l'étape i et notons $c_1, c_2, \ldots, c_{n-i+1}$ les n-i+1 derniers coefficients de la ligne i. Pour éliminer n-i de ces coefficients on effectue successivement sur les colonnes i et i+1, i et i+2, ..., i et n les transformations:

$$\begin{pmatrix} c_1 & c_2 \\ \vdots & \vdots \end{pmatrix} \begin{pmatrix} p_2 & -\frac{c_2}{g_2} \\ q_2 & \frac{c_1}{g_2} \end{pmatrix}, \quad \begin{pmatrix} g_2 & c_3 \\ \vdots & \vdots \end{pmatrix} \begin{pmatrix} p_3 & \frac{c_3}{g_3} \\ q_3 & \frac{g_2}{g_3} \end{pmatrix}, \quad \ldots, \quad \begin{pmatrix} g_{n-i} & c_{n-i+1} \\ \vdots & \vdots \end{pmatrix} \begin{pmatrix} p_{n-i+1} & \frac{c_{n-i+1}}{g_{n-i+1}} \\ q_{n-i+1} & \frac{g_{n-i}}{g_{n-i+1}} \end{pmatrix}$$

On a donc remplacé c_1 par :

$$p_{n-i+1}\,(\ldots p_4\,(p_3\,(p_2\,c_1 + q_2\,c_2) + q_3\,c_3) + q_4\,c_4) + \ldots) + q_{n-i+1}\,c_{n-i+1}$$

Cela revient à effectuer l'algorithme d'Euclide généralisé successivement sur:

$$(c_1 , c_2)$$
$$(\text{pgcd}(c_1 , c_2) , c_3)$$
$$\dots$$
$$(\text{pgcd}(c_1 , \dots , c_{n-i+1}) , c_{n-i+1})$$

On a résolu l'équation Diophantienne associée aux (n-i+1) coefficients considérés. Or nous avons vu précédemment que l'algorithme de Bradley fournissait une solution minimale $(x_1 , x_2 , \dots , x_{n-i-1})$ à cette équation. Comment tirer parti de ce résultat ?

Algorithme de Bradley [BRA71]

Considérons maintenant les coefficients $c_1 , c_2 , \dots , c_{n-i+1}$ dans leur ensemble. On cherche une transformation de déterminant 1 éliminant les n-i derniers coefficients et dont les coefficients sont de taille raisonnable.
Soit $(x_1 , x_2 , \dots , x_{n-i-1})$ la solution de l'équation Diophantienne donnée par l'algorithme de Bradley, et g le pgcd des coefficients. Dans quelques cas particuliers, il existe une transformation évidente qui se construit en deux étapes:

Dans un premier temps:

(i) Si il existe x_k égale à 1 ou c_k égale à 0 , on effectue sur la sous-matrice formée des (n-i+1) dernières lignes et dernières colonnes la transformation τ_1

$$\begin{pmatrix} 1 & 0 & 0 & x_1 & 0 & 0 & 0 \\ 0 & \ddots & 0 & \vdots & 0 & 0 & 0 \\ 0 & 0 & \ddots & x_{k-1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_{k+1} & \ddots & 0 & 0 \\ 0 & 0 & 0 & \vdots & 0 & \ddots & 0 \\ 0 & 0 & 0 & x_{n-i+1} & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & I_{k-2} & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & I_{n-i-1-k} \end{pmatrix}$$

où les x_j sont en colonne k.

(ii) Si il existe x_k égal à -1, on applique la transformation τ_1, puis on multiplie la colonne i par -1.

(iii) Si il existe k tel que c_1 divise c_k , on effectue la transformation τ_2:

$$\begin{pmatrix} 1 & 0 & 0 & \alpha & 0 & 0 & 0 \\ 0 & \ddots & 0 & \vdots & 0 & 0 & 0 \\ 0 & 0 & \ddots & x_{k-1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_{k+1} & \ddots & 0 & 0 \\ 0 & 0 & 0 & \vdots & 0 & \ddots & 0 \\ 0 & 0 & 0 & x_{n-i+1} & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & I_{k-2} & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & I_{n-i-1-k} \end{pmatrix}$$

avec $\alpha = x_1 - (c_k / c_1)(1 - x_k)$.

(iv) Si il existe k tel que x_k soit nul, on pose $x_j' = (1 - d)\, x_j$, avec $d = c_k / g$,et on effectue la transformation τ_3:

$$\begin{pmatrix} 1 & 0 & 0 & x_1' & 0 & 0 & 0 \\ 0 & \ddots & 0 & \vdots & 0 & 0 & 0 \\ 0 & 0 & \ddots & x_{k-1}' & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_{k+1}' & \ddots & 0 & 0 \\ 0 & 0 & 0 & \vdots & 0 & \ddots & 0 \\ 0 & 0 & 0 & x_{n-i+1}' & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & I_{k-2} & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & I_{n-i-1-k} \end{pmatrix}$$

Dans les quatre cas précédents, les (n-i+1) derniers coefficients obtenus en ligne i sont:

$$g \; c_2 \; c_3 \; \ldots \; c_{k-1} \; c_1 \; c_{k-1} \; \ldots \; c_{n-i+1}$$

Une fois dans cette situation:

des opérations de réduction par rapport à la colonne i (à chaque colonne, on retire le quotient de g par son $i^{ème}$ coefficient fois la colonne i) permettent d'éliminer les (n-i) derniers coefficients de la ligne i.

Lorsque l'on ne se trouve dans aucun des quatre cas ci-dessus, on peut se ramener au cas (i) en faisant apparaître un zéro par une transformation entre les colonnes i et i+1 par exemple.

L'algorithme de Bradley repose sur ces différentes remarques.

Algorithme de Bradley:

début
pour i:=1 **à** n **faire**
étape 1
Appliquer l'algorithme de Bradley pour trouver (x_c , x_{c+1} , ... , x_{n-i-1}) tel que pgcd(a_{ii} , $a_{i,j+1}$, ... , a_{in}) = $\sum_{j=i}^{n} x_{j-i+1}a_{ij}$
étape 2
Si il existe k tel que x_k=1 ou a_{k+i-1}=0, appliquer la transformation τ_1
Sinon si il existe k tel que x_k= -1, appliquer la transformation τ_1 puis multiplier la colonne i par -1
Sinon si il existe k tel que a_i divise a_{k+i-1}, appliquer la transformation τ_2
Sinon si il existe k tel que x_k soit nul, appliquer la transformation τ_3

Sinon
début
calculer g, s, t, tels que g = pgcd(a_{ii} , $a_{i,i+1}$) = s.$a_{i,i}$ + t.$a_{i,i+1}$.
faire apparaître un zéro en position (i,i)
retourner à l'étape 1
fin
étape 3
Pour j:= i+1 **à** n **faire**
retirer à la colonne j , a_{ij}/a_{ii} fois la colonne i
fin

La probabilité d'avoir le pgcd des éléments d'une ligne égal à 1 étant trés importante pour une matrice aléatoire, il est très fréquent de se trouver dans le cas (iv). En pratique, cet algorithme est donc très intéressant. La complexité théorique du grossissement des coefficients reste cependant exponentielle.

V.3.2. Algorithme mineur par mineur [KAN79], [CHO82]

En 1979, Kannan et Bachem ont proposé un algorithme qui suit le schéma suivant :

on suppose que tous les mineurs principaux de A sont non singuliers.
pour i:=1 **à** n **faire**
mettre le mineur principal d'ordre i sous forme normale d'Hermite

A l'étape i, les coefficients $a_{1,i}$, $a_{2,i}$,..., $a_{i-1,i}$ sont éliminés respectivement grâce aux coefficients diagonaux a_{11} , a_{22} ,..., $a_{i-1,i-1}$.

Les opérations d'éliminations et de réductions se font par les mêmes transformations que dans le paragraphe b.1.

La complexité de l'algorithme en opérations arithmétiques est alors:

$$n^2/2T_{pgcd} + \sum_{k=1}^{n-1} (k-1)^2(k-2)T_{mul} + \sum_{i=n-k+2}^{n} i\,(4T_{mul} + 2T_{div})$$

Les réductions nécessitent donc de l'ordre de $O(n^4)$ multiplications en précision infinie.

Les éléments de la colonne i ne sont pas modifiés avant l'étape i et les (i-1) premiers coefficients diagonaux ont déjà été réduits. Les transformations font donc intervenir des coefficients plus petits que ceux obtenus dans les premiers algorithmes. La borne sur les coefficients intermédiaires est alors [KAN79]

$$B_{coef} \leq 2n^{10n^2} B^{6n^2}$$

On peut également proposer un algorithme pour une matrice initiale quelconque; on permute les colonnes, s'il y a lieu, au cours de l'algorithme pour rendre les mineurs principaux non singuliers.

Algorithme mineur par mineur
début
pour k :=1 **à** n **faire**
index := 1
tant que index ≠ 0
pour i:=1 **à** k-1 **faire**

{ Triangularisation }

éliminer a_{ik}

{ Réduction }

réduire $a_{i1}, \ldots, a_{i,i-1}$

réduire $a_{k1}, \ldots, a_{k,k-1}$
si $a_{kk} = 0$ **permuter** les colonnes k et k+index, index=index+1
sinon index := 0
fin tantque
fin

Remarque:

Lorsqu'une permutation a été effectuée à l'étape (k-1) entre les lignes (k-1) et k, la colonne k récupérée à l'étape k possède déjà (k-1) zéros. On ne fait donc jamais d'élimination superflue. Cependant, la borne sur la taille des coefficients donnée par Kannan et Bachem n'est à priori plus valable lorsque l'on effectue des permutations.

Stratégie de Chou et Collins [CHO82]

En 1982, Chou et Collins ont proposé une modification de cette algorithme qui permet de diminuer la borne sur les coefficients intermédiaires.
Dans l'algorithme précédent les coefficients sont réduits de haut en bas et de gauche à droite.

Exemple: pour une matrice 4x4 les éléments sont réduits dans l'ordre indiqué:

$$\begin{bmatrix} x & & & \\ 1 & x & & \\ 2 & 3 & x & \\ 4 & 5 & 6 & x \end{bmatrix}$$

Chou et Collins diminuent la taille des coefficients en modifiant l'ordre de réduction des éléments. A l'étape k, la colonne i, i<k, est réduite en utilisant uniquement les colonnes déjà réduites i+1, ..., k. Les coefficients sont éliminés de droite à gauche et de bas en haut.

Ce qui donne pour l'exemple précédent:

$$\begin{bmatrix} x & & & \\ 4 & x & & \\ 5 & 2 & x & \\ 6 & 3 & 1 & x \end{bmatrix}$$

Les réductions sont alors effectuées à la fin de chaque étape, après toutes les éliminations.

L'algorithme est alors:

début
pour k :=1 **à** n **faire**
index := 1
tant que index ≠ 0
pour i:=1 **à** k-1 **faire**

{ Triangularisation }

éliminer a_{ik}

pour i:=k-1 **à** 1 **faire**

pour j:=i+1 **à** n **faire**

{ Réduction }

réduire $a_{j,i}$

si $a_{kk} = 0$ **permuter** les colonnes k et k+index, index=index+1
sinon index := 0
fin tantque
fin

En supposant qu'il n'y ait pas de permutation à effectuer, la borne sur les coefficients intermédiaires est alors [CHO82] :

$$\boxed{B_{coef} \leq 2^{3n} (nB)^{4(n+1)}}$$

V.3.3. Algorithme modulo d [Ill85], [DOM87]

Les différents algorithmes que nous avons vu manipulent des coefficients qui généralement ne cessent de croître, jusqu'à l'étape n - 1. Or les coefficients de la matrice de départ sont bornés par B et la forme finale est une forme réduite dont les coefficients sont bornés par le déterminant.

L'algorithme qui suit a été proposé en 1985 par Illopoulos et repris par Domich, Kannan et Trotter en 1987. Il permet de conserver, tout au long de l'algorithme, des coefficients bornés par le déterminant. Il est basé sur le théorème suivant :

Théorème: soit M une matrice nxn dont les colonnes sont dans L(A) (réseau généré par la matrice A) et $A' = \begin{pmatrix} A & M \\ 0 & I \end{pmatrix}$. Soit H' la forme normale d'Hermite de A', alors on a $H' = \begin{pmatrix} H & 0 \\ 0 & I \end{pmatrix}$, où H est la forme normale d'Hermite de A.

Preuve:
Chaque colonne de M étant une combinaison linéaire des colonnes de A, A' peut être mise sous la forme $\begin{pmatrix} A & 0 \\ 0 & I \end{pmatrix}$ par une transformation unimodulaire.
Il est clair que la forme d'Hermite de cette matrice est $\begin{pmatrix} H & 0 \\ 0 & I \end{pmatrix}$.

Dans ce qui suit nous noterons d= | déterminant (A) |.

Corollaire:
Pour tout i dans 1...n, le vecteur $d.e_i$ est un élément du réseau L(A).
e_i étant le ième vecteur unité.

Preuve:
$A.(dA^{-1}) = dI$. Or dA^{-1} est à coefficients entiers, donc chaque colonne de dI peut s'écrire comme combinaison linéaire entière des colonnes de A.

L'algorithme proposé par Illopoulos, que l'on nommera algorithme modulo d, consiste à calculer la forme normale d'Hermite de la matrice [A dI], tout en utilisant les colonnes de dI pour travailler modulo d.

Cet algorithme nécessite le calcul du déterminant de A. Nous verrons par la suite que ce calcul est d'un coût négligeable par rapport à celui du calcul de la forme d'Hermite.

A l'étape k de l'algorithme la matrice est sous la forme :

$$\text{ligne k} \rightarrow \begin{pmatrix} * & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ * & * & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ * & * & * & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ + & + & + & x & x & x & x & x & x & d & 0 & 0 & 0 \\ x & x & x & x & x & x & x & x & x & 0 & d & 0 & 0 \\ x & x & x & x & x & x & x & x & x & \vdots & \ddots & \vdots & 0 \\ x & x & x & x & x & x & x & x & x & 0 & \cdots & 0 & d \end{pmatrix}$$

légende:

+ coefficients à réduire à l'étape k
* coefficients déjà traités
en italique : coefficients à éliminer à l'étape k

Algorithme modulo d:

début
pour k:=1 **à** n **faire**

{ Triangularisation }

pour j:=k+1 **à** n+k **faire**
pour i:=k+1 **à** n **faire**

$$\begin{pmatrix} a_{ik} \\ a_{ij} \end{pmatrix} := \begin{pmatrix} (p\, a_{ik} + q\, a_{ij}) \bmod d \\ (-\, a_{kj}\, a_{ik} + a_{kk}\, a_{ij}) \bmod d \end{pmatrix}$$

où $p\, a_{kk} + q\, a_{kj} = \text{pgcd}\,(a_{kk}\,,\, a_{kj}) > 0$

$a_{kk} := \text{pgcd}(a_{kk}\,,\, a_{kj})$
$a_{kj} := 0$

{ Réduction }

pour j:=1 **à** k-1 **faire**
quotient := div (a_{kj} , a_{kk})
$a_{kj} := a_{kj}$ - quotient.a_{kk}
pour i:=k+1 **à** n **faire**
a_{ij} := mod(a_{ij} - quotient.a_{ik} , d)

fin

Si a et b sont des entiers, les fonctions *mod* et *div* sont ainsi définies:

$$\text{si } a = b\,q + r \ , 0 \le r < b, \text{ alors } q = \text{div}(a\ ,\ b) \text{ et } r = \text{mod}(a\ ,\ b).$$

On obtient ainsi des nombres positifs, ce qui est en accord avec le choix du pgcd. Les coefficients diagonaux sont automatiquement positifs.

En 1987, Kannan, Bachem et Trotter remarquent que cet algorithme peut être amélioré en appliquant le corollaire à des sous matrices.

Corollaire: Si on pose $d_1 = d$ et $d_{i+1} = d_i / h_{ii}$ pour i=1,2,...,n alors pour $1 \le i \le n$, $d_i.e_i$ est un élément du réseau L(A), où les h_{ii} désignent les éléments diagonaux de la forme d'Hermite de A.

Preuve: pour i=1, c'est vrai d'après le corollaire précédent. Ce corollaire peut être appliqué à la sous matrice (n - 1) x (n - 1) de H obtenue en supprimant la première ligne et la première colonne.
On a alors $d_2 = (d_1 / h_{11}) .e_1 \in L(H) = L(A)$ et ainsi de suite.

L'algorithme modulo d peut donc être amélioré en ajoutant simplement, aprés l'étape de réduction, l'instruction:

$$\textbf{Si } k < n \textbf{ alors } d = d / a_{kk}$$

Remarquons que la complexité de l'algorithme n'est pas modifiée puisque pour de nombreux exemples la forme d'Hermite ne possède que des 1 sur la diagonale sauf en dernière position. Il apparaît que dans les exemples que nous traitons dans la partie expérimentation, les premiers termes diagonaux de la forme d'Hermite sont en général égaux à 1.
Par la suite, pour simplifier, nous considèrerons fréquemment le premier algorithme.

Complexité de l'algorithme:

Dans un premier temps, l'algorithme précédent nécessite le calcul du déterminant de A. L'algorithme deGauss avec développements p-adiques [DIX82] nous permet de résoudre ce problème avec la complexité :

$$\boxed{O(\ n^3\ (\log^2 n + \log^2 B)\ )}$$

L'algorithme modulo d calcule la forme d'Hermite d'une matrice de dimension n x 2n. A l'étape k, k éléments de la matrice $d.I_n$ sont à éliminer, soit n éléments à éliminer dans la matrice globale. Le nombre total

d'éléments à éliminer sur les n étapes est donc n^2, dont $\frac{n.(n-1)}{2}$ pour la matrice A. Le nombre de réductions ne change pas.

La complexité arithmétique est alors:

$$\boxed{\sum_{k=1}^{n-1} n(n-k)(4T_{mul}+2T_{div}) + n^2\, T_{pgcd} + \sum_{k=1}^{n-1} k(n-k)(T_{mul}+T_{div})}$$

Soit

$$\boxed{O(n^3) \text{ opérations arithmétiques en précision infinie}}$$

Les coefficients intermédiaires sont bornés par d, et d'après la borne de Hadamard :

$$\log d \leq n\,(\log \frac{n}{2} + \log B)$$

La complexité est donc finalement

$$\boxed{O(n^5\,(\log^2 n + \log^2 B)\,)}$$

Remarquons que le coût de l'algorithme de Gauss est négligeable dans la complexité totale.

En annexe 1, nous donnons un exemple qui permet de comparer l'algorithme mineur par mineur de Kannan et Bachem et l'algorithme modulo d. R. Bradford a étudié les différents algorithmes pour des matrices de faible dimension, et donne différents résultats expérimentaux dans sa thèse [BRA88].

V.4. Conclusion

Pour simplifier le travail de parallélisation de la seconde partie, nous ne nous préoccupons que des matrices carrées, non singulières. Tous les algorithmes présentés ici peuvent être adaptés au cas général.

L'algorithme qui donne la meilleure complexité est l'algorithme modulo d. Nous verrons par la suite qu'il est bien adapté à la parallélisation.

Remarquons que le calcul de la forme normale d'Hermite est un problème particulièrement complexe. Du fait qu'il n'existe pas de relation d'ordre dans les corps finis, nous ne connaissons pas de techniques modulaires (comme pour l'algorithme de Gauss par exemple) pour effectuer ce calcul.

CHAPITRE VI

FORME NORMALE D'HERMITE SUR LES POLYNOMES

Nous nous intéressons ici aux matrices à coefficients dans $\mathbb{Q}[x]$.

L' algorithme proposé par Illopoulos pour les matrices à coefficients entiers peut s'appliquer, moyennant quelques modifications, aux matrices à coefficients dans $\mathbb{Q}[x]$. Les degrés intermédiaires restent bornés par le degré du déterminant. Nous trouvons donc des résultats analogues à ceux obtenus dans le cas entier. Intuitivement, cela n'est pas surprenant; en effet le degré d'un polynôme et la taille d'un entier jouent un rôle similaire.

Cependant, si le degré reste borné, le problème du grossissement des coefficients intermédiaires des polynômes n'est pas résolu.

VI.1. Définition

Définition:
Soit $H=(h_{ij})_{i=1,\ldots,n;j=1,\ldots,n}$ une matrice d'ordre n à coefficients dans $\mathbb{Q}[x]$.
H est dite sous Forme Normale d'Hermite si et seulement si:

(i) $h_{ij} = 0$, pour tout $j > i$,

(ii) degré(h_{ij}) < degré(h_{ii}), pour tout $j < i$,

(iii) tous les éléments diagonaux sont unitaires (le degré du coefficient de tête est 1),

VI.2. Algorithmes

Il est clair que les algorithmes décrits pour le cas entier peuvent être appliqués dans le cas de domaines euclidiens quelconques, moyennant le choix d'une division euclidienne définie dans cet ensemble.

Nous nous intéresserons maintenant au cas de matrices ayant pour coefficients des polynômes à une variable, eux-mêmes à coefficients rationnels.

Dans ce cas, l'algorithme modulo d s'écrit:

Algorithme modulo d:

début
pour k:=1 **à** n **faire**

{ Triangularisation }

pour j:=k+1 **à** n+k **faire**

pour i:=k+1 **à** n **faire**

$$\begin{pmatrix} a_{ik} \\ a_{ij} \end{pmatrix} := \begin{pmatrix} \text{pmod}(p\, a_{ik} + q\, a_{ij} , d) \\ \text{pmod}(-\, a_{kj}\, a_{ik} + a_{kk}\, a_{ij} , d) \end{pmatrix}$$

où $p\, a_{kk} + q\, a_{kj} = \text{pgcd}\,(a_{kk} , a_{kj})$, unitaire

$a_{kk} := \text{pgcd}(a_{kk} , a_{kj})$
$a_{kj} := 0$

{ Réduction }

pour j:=1 **à** k-1 **faire**

quotient := pdiv (a_{kj} , a_{kk})
$a_{kj} := a_{kj}$ - quotient.a_{kk}

pour i:=k+1 **à** n **faire**

$a_{ij} := \text{pmod}(a_{ij} - \text{quotient}.a_{ik} , d)$

fin

VI.3. Complexité

Deux quantités doivent être prises en compte dans le calcul de la complexité: le degré et la taille des coefficients des polynômes.

Nous donnons tout d'abord quelques définitions et propriétés classiques.

Soit a une entier, on notera *taille(a)* le nombre de bits nécessaires à l'écriture binaire de a, soit au plus $\lfloor \log_2 (a) \rfloor + 1$.

La taille d'un rationnel sous forme canonique (numérateur et dénominateur premiers entre eux) est la somme de la taille de son numérateur et de la taille de son dénominateur.

Soient P_i , i=1 à k, k polynômes de $\mathbb{Q}[x]$. Nous noterons
taille(P_i) le maximum des tailles des coefficients de P_i.

Proposition 1: si les degrés des P_i sont bornés par D, alors on a:

$$\begin{cases} \text{taille}(P_1 + P_2 + \ldots + P_k) \leq \log(k) + \max(\text{taille}(P_1),\ldots,\text{taille}(P_k)) \\ \text{taille}(P_1 P_2 \ldots P_k) \leq k \log(D) + \text{taille}(P_1) + \ldots + \text{taille}(P_k) \end{cases}$$

Proposition 2: soit M une matrice nxn à coefficients dans $\mathbb{Q}[x]$ et MD un borne sur les degrés des coefficients de M, alors:

$$\begin{cases} \deg(\text{dét}(M)) \leq n \deg(M) \\ \text{taille}(\text{dét}(M)) \leq 3\ MD\ n^3\ \text{taille}(M) \end{cases}$$

Preuve:
dét(M) est la somme de n! termes, chacun produit de n facteurs de degré borné par deg(M). On en déduit la première borne.

Posons M' = M.d où d est le produit des dénominateurs des coefficients des polynômes de M. M' est à coefficients des polynômes de $\mathbb{Z}[x]$ et dét(M) = dét(M')/d. On a alors

$$\text{taille}(d) \leq MD\ n^2\ \text{taille}(M)$$

dét(M') est la somme de n! termes, chacun produit de n facteurs de taille bornée par n log(MD) + n taille(M). La somme est donc de taille bornée par : $\log(n!) + n \log(MD) + n\ \text{taille}(M')$
qui est borné par : $n \log(n) + n \log(MD) + n(n^2\ MD + 1)\ \text{taille}(M)$
de plus

$$\begin{aligned} \text{taille}(\text{dét}(M)) &\leq \text{taille}(\text{dét}(M')) + \text{taille}(d) \\ &\leq n\ (\log(n\ MD) + (MD\ n^2 + 1)\ \text{taille}(M) + n\ MD\ \text{taille}(M)) \end{aligned}$$

On en déduit la borne donnée.

Modèle Arithmétique:

Dans un premier temps, nous nous intéressons à évaluer le nombre d'opérations arithmétiques sur les polynômes. On pose donc l'axiome suivant :

Axiome: les coûts des opérations polynomiales sont des constantes.

Nous noterons:

T_{padd}	: le coût de l'addition de deux polynômes
T_{pmul}	: le coût de la la multiplication de deux polynômes
T_{pdiv}	: le coût de la division de deux polynômes
T_{ppgcd}	: le coût du calcul du pgcd de deux polynômes

VI.3.1. Nombre d'opérations arithmétiques

Algorithme mineur par mineur

Théorème:
le coût de l'algorithme mineur par mineur est:

$$\sum_{i=2}^{n}\sum_{j=1}^{i-1}\left(T_{ppgcd} + (n-j)(4\,T_{pmul} + 2\,T_{padd})\right) + \sum_{i=2}^{n}\sum_{j=2}^{i-1}\left(T_{pdiv} + \sum_{k=j}^{n}(n-j+1)(T_{pmul} + T_{padd})\right)$$

soit de l'ordre de $O(n^4)$ opérations arithmétiques sur des polynômes.

Algorithme modulo d

Le calcul du déterminant peut se faire en évaluant dét(A(0)), dét(A(1)), ..., dét(A(n.MD)) où MD est une borne sur les degrés des coefficients de A. On interpole alors la solution.
On peut donc calculer le déterminant en un temps polynomial.

La complexité arithmétique se déduit de celle obtenue dans le cas entier:

$$\sum_{k=1}^{n-1} n\,(n-k)(4\,T_{pmul} + 2\,T_{pdiv}) + n^2\,T_{ppgcd} + \sum_{k=1}^{n-1} k\,(n-k)(T_{pmul} + T_{pdiv})$$

La complexité de cet algorithme est en $O(n^3)$.

Nous allons maintenant examiner plus précisément le problème de la croissance des coefficients intermédiaires, qui joue un rôle primordial dans la complexité du problème.

VI.3.2. Grossissement des coefficients intermédiaires

Algorithme mineur par mineur

Notons:

- A^i la matrice obtenue après l'étape i de l'algorithme mineur par mineur, c'est à dire la matrice dont le mineur principal d'ordre i est sous forme normale d'Hermite (A^1 est la matrice en entrée).

- K^i la matrice unimodulaire telle que : $A\,K^i = A^i$ $\quad (K^1 = I)$.

- MD une borne sur les degrés des entrées.

Pour l'algorithme mineur par mineur, on a les résultats suivants [KRI85]: après chaque étape de l'algorithme

Théorème 1:

$$\deg(A^i) \leq (2n + 1)\, MD$$

$$\deg(K^i) \leq 2n\, MD$$

Théorème 2:

$$\text{taille}(A^i) \leq 8\, MD^2\, n^3\, \text{taille}(A^1)$$

$$\text{taille}(K^i) \leq 6\, MD^2\, n^3\, \text{taille}(A^1)$$

On a d'autre part une borne sur les degrés de la matrice à chaque instant de l'exécution de l'algorithme (aussi bien après l'obtention de la forme d'Hermite d'un mineur principal, qu'après l'élimination ou la réduction d'un certain coefficient). Soit A la matrice à un instant donné de l'algorithme, on a:

Théorème 3:

$$\deg(A) \leq 4n\,(2n + 1)\, MD$$

$$\deg(K) \leq (16\, n^3 + 8\, n^2)\, MD$$

Remarquons que, même si nous avons une borne correcte sur la taille des coefficients après chaque étape, entre deux étapes ils peuvent croître considérablement en raison des divisions effectuées pour le calcul du pgcd.

La meilleure complexité pour le pgcd de polynômes est celle fournie par G.Collins pour le pgcd de polynômes à coefficients entiers:

Proposition 3[COL67]:
Supposons que $P_1(x)$ et $P_2(x)$ soient deux éléments de $\mathbb{Z}[X]$ avec

$$\begin{cases} L = \max\{\text{taille}(P_1(x)), \text{taille}(P_2(x))\} \\ D = \max\{\deg(P_1(x)), \deg(P_2(x))\} \end{cases}$$

Alors il existe U(x), V(x) et G(x) dans $\mathbb{Z}[X]$ tels que

$$\begin{cases} G \text{ est le pgcd de } P_1 \text{ et } P_2 \\ G(x) = U(x)\, P_1(x) + V(x)\, P_2(x) \end{cases}$$

De plus

$$\boxed{\text{Max}(\text{taille}(U(x)), \text{taille}(V(x)), \text{taille}(G(x))) \leq D\,(8\, L + \log_2 D)}$$

Preuve: c'est la borne obtenue lorsque l'on calcule le pgcd par l'algorithme des sous-résultants de G.Collins [COL67].

Le résultat qui suit, nous permettra de donner une borne sur la taille des coefficients pour la division euclidienne de deux polynômes à coefficients rationnels. On en déduit une borne sur les coefficients après réduction, ou sur le calcul du pgcd de deux polynômes à coefficients rationnels.

Proposition 4:
Soient $P_1(x)$ et $P_2(x)$ deux éléments de $\mathbb{Q}[x]$ de degrés respectifs $d_1$et d_2, avec $d_1 \geq d_2$. Soit $R(x)$ et $Q(x)$ le reste et le quotient de la division euclidienne de $P_1(x)$ par $P_2(x)$, alors:

$$\text{Max} \begin{Bmatrix} \text{taille}(Q(x)) \\ \text{taille}(R(x)) \end{Bmatrix} \leq (d_1 - d_2 + 1)(2\ \text{taille}(P_2) + 1) + \text{taille}(P_1)$$

Preuve:
Soient $q_0, q_1, \ldots, q_k$ les monômes du quotients de P_1 par P_2, et $R_0, R_2, \ldots, R_k = R(x)$ les restes successifs obtenus lors de la division, où k est tel que R_k est le premier reste de degré inférieur à d_2.
On a $q_0 = (ct(P_1)/ct(P_2))\ x^{d_1 - d_2}$ et :

$$\begin{cases} P_1 = q_0\ P_2 + R_0 \\ R_{i-1} = q_i\ P_2 + R_i\ , \text{ pour } 1 \leq i \leq k \end{cases}$$

on a donc:

$$\text{taille}(q_0) \leq \text{taille}(P_1) + \text{taille}(P_2)$$
$$\text{taille}(q_i) \leq \text{taille}(R_{i-1}) + \text{taille}(P_2)$$

et par suite:

$$\text{taille}(R_0) \leq 2\ \text{taille}(P_2) + \text{taille}\ (P_1) + 1$$
$$\text{taille}(R_i) \leq 2\ \text{taille}(P_2) + \text{taille}\ (R_{i-1}) + 1\ , \text{pour } 1 \leq i \leq k$$

Par induction, on déduit alors:

$$\text{taille}(R_k) \leq 2\ (k+1)\ \text{taille}(P_2) + \text{taille}\ (P_1) + k + 1$$

A chaque étape de la division, le degré du reste diminue au moins de 1, il y a donc au plus $d_1 - d_2 + 1$ étapes, et $k \leq d_1 - d_2$. On en déduit la borne sur $R(x)$ et $Q(x)$.

Dans le calcul du pgcd de deux polynômes par l'algorithme d'Euclide, le nombre d'itérations est borné par le minimum d des degrés des deux polynômes.
La taille des coefficients du pgcd obtenu est donc bornée par $2^d L$, où $L = \max\{\text{taille}(P_1(x)), \text{taille}(P_2(x))\}$.

Il est clair que l'algorithme de Collins donne une bien meilleure complexité mais il est également plus difficile à implanter.

Algorithme modulo d

Pour l'algorithme modulo d, les degrés des polynômes sont bornés par les degrés du déterminant, donc par n deg(A^1). On gagne donc un facteur de n sur les degrés par rapport à l'algorithme mineur par mineur. Cependant, chaque réduction par rapport au déterminant entraîne une croissance des coefficients des polynômes qui amplifie encore le grossissement dû aux autres opérations.
Une borne sur la taille des coefficients du déterminant nous est donnée par la proposition 2. Comme nous l'avons vu ci-dessus, une division euclidienne est très coûteuse. Donc, même si cet algorithme permet de limiter la taille des degrés des coefficients, il est beaucoup moins performant que l'algorithme précédent en raison de la taille des rationnels à manipuler.

VI.3.3. Implantation

Les deux algorithmes vus précédemment ont été implanté en Macsyma. Nous avons traité les exemples suivant:

$$(E^1)\begin{bmatrix} -x+1 & -2 & x^2+1 & 3 & 7.x-1 \\ x-2 & -1 & 3.x-1 & x-4 & x-5 \\ x & 8 & 6.x-1 & -5 & x-2 \\ x-2 & -5 & 3 & x-4 & -7 \\ -3 & 7.x^2 & -5 & 7.x-8 & x-6 \end{bmatrix}$$

$$(E^2)\begin{bmatrix} -2.x+1 & -2 & x^2+1 & 7.x-1 \\ 3.x-1 & x-4 & x^2-5 & 8.x-5 \\ x^3 & 8 & 6.x-1 & 6.x \\ x-2 & -5.x & -7 & 3.x-1 \end{bmatrix}$$

$$(E^3)\begin{bmatrix} -2.x^3+1 & -2.x^3 & x^2+1 & 7.x-1 \\ 3.x-1 & x^4-4 & x^2-5 & 8.x^2-5 \\ x^3 & 8 & 6.x-1 & 6.x \\ x-2 & -5.x & -7 & 3.x^2-1 \end{bmatrix}$$

Les temps d'exécution, en secondes, obtenus pour ces trois exemples sont répertoriés dans le tableau suivant:

	E^1	E^2	E^3
mineur par mineur	54	41	98
modulo d	118	101	539

Dans tous les cas, l'algorithme mineur par mineur s'avère plus performant que l'algorithme modulo d. Nous remarquons avec le troisième exemple, que cette remarque est d'autant plus nette que les degrés des polynômes sont plus grands. Signalons que, dans l'exemple E^3, le temps de calcul du déterminant (par l'algorithme de Gauss classique) est de 21 secondes, donc très faible par rapport au calcul de la forme d'Hermite. Cependant, ce déterminant étant d'un degré plus élevé que dans les deux premiers exemples, le coût de toutes les réductions effectuées par rapport aux colonnes contenant le déterminant influe de façon conséquente sur la complexité.

La complexité en nombre de réductions de l'algorithme mineur par mineur est d'un facteur n plus grande (ordre supérieur) que pour l'algorithme modulo d; mais pour des matrices de petites dimensions, ce facteur influe guère. D'autre part le nombre d'éliminations est plus important dans l'algorithme modulo d ($n^2/2$ éliminations de plus). L'intérêt de l'algorithme mineur par mineur est donc d'autant plus net que les matrices sont de petites dimensions. Et il est clair que, compte tenu de la complexité du problème, on ne peut envisager de traiter de grandes matrices.

VI.4. Algorithme mineur par mineur avec cumul

VI.4.1. Principe de l'algorithme

Nous proposons un algorithme construit à partir de l'algorithme mineur par mineur de Kannan et Bachem. Il consiste à cumuler le plus d'opérations possible à chaque étape. Les éliminations et réductions sont effectuées par des transformations identiques et dans le même ordre dans les deux algorithmes. Les différences se situent au niveau de l'application des transformations sur la matrice de départ.
Un intérêt de cet algorithme est de "rassembler" le parallélisme sous-jacent à l'algorithme au niveau des produits de matrices.

Idée Générale :
Les transformations élémentaires sont calculées en suivant le schéma de l'algorithme mineur par mineur, mais les coefficients ne sont mis à jour que lorsque cela est nécessaire pour le calcul d'une nouvelle transformation.

Les opérations seront, autant que possible, effectuées, en priorité, entre les coefficients des matrices de transformation.

Dans la suite, nous représentons par $a_{k,l}^{(i,j)}$ le coefficient d'indice k,l de la matrice $A^{(i,j)}$ obtenue à la $i^{ième}$ étape de l'algorithme mineur par mineur, après l'élimination du $(j-1)^{ième}$ élément de la colonne i.

Soient $g = s.\, a_{j,j}^{(i,j)} + t.\, a_{j,i}^{(i,j)} = pgcd\left(a_{j,j}^{(i,j)}, a_{j,i}^{(i,j)}\right)$ et $q_k = \left\lfloor \frac{a_{i,k}^{(i,j)}}{a_{i,i}^{(i,j)}} \right\rfloor$, et soient les tâches élémentaires $E_j^{(i)}$ (élimination) et $R_j^{k(i)}$ (réduction) suivantes :

$$E_j^{(i)} = \begin{array}{c} \\ j \\ i \\ \\ \end{array} \begin{array}{c} \begin{array}{cccc} j & & i & \end{array} \\ \begin{bmatrix} 1 & & & 0 \\ & & & \\ & s & \frac{-a_{j,i}^{(i,j)}}{g} & \\ & & & \\ & t & \frac{a_{j,j}^{(i,j)}}{g} & \\ & & & \\ 0 & & & 1 \end{bmatrix} \end{array}$$

$$R_j^{k(i)} = \begin{array}{c} \\ j \\ \\ \end{array} \begin{array}{c} \begin{array}{c} k \end{array} \\ \begin{bmatrix} 1 & & \\ & \ddots & \\ -q & & 1 \end{bmatrix} \end{array}$$

L'algorithme est alors basé sur la remarque fondamentale suivante :

Il n'est indispensable de mettre à jour les coefficients d'indice:		
(j,j) et (j,i)	que lorsqu'on calcule $E_j^{(i)}$	pour $i \geq j$
(j,k), k<j,	que lorsqu'on calcule $R_j^{k(i)}$	

Soient i, k tels que $1 \leq i, k \leq n$. On distinguera deux niveaux de cumul:

- au niveau le plus bas : cumul des transformations sur le mineur principal d'ordre i durant l'étape i.

Lors du calcul de la forme d'Hermite du mineur d'ordre i, la ligne j<i n'est mise à jour que pour l'élimination de l'élément d'indice (j, i).
On notera TI la matrice des transformations effectuées pour la mise sous forme d'Hermite du mineur d'ordre i.

- au niveau le plus haut : cumul des transformations à effectuer sur les lignes d'indice supérieur ou égal à k, et qui correspondent aux étapes 1 à k-1 de l'algorithme mineur par mineur.

Les lignes d'indice supérieur à k ne sont pas modifiées jusqu'à l'étape k. La ligne k est mise à jour à l'étape k pour le calcul des réductions sur la ligne k.
On notera T la matrice des transformations effectuées depuis le début jusqu'à un instant donné de l'exécution de l'algorithme.

VI.4.2. Algorithme

L'algorithme décrit ci-dessous cumule les opérations comme indiqué précédemment.
Les principales étapes sont :

- Calcul de la nouvelle élimination, et mise à jour de la matrice TI qui cumule les éliminations élémentaires
- Calcul de la nouvelle réduction, et mise à jour de la matrice RI qui cumule les réductions élémentaires
- Traitement de la ligne
- Mise à jour de la matrice de cumul des transformations T

Algorithme

convention : $T^{(i)}$ mineur principal d'ordre i de la matrice T.

début
$T := Id_n$;

Pour i = 2 **à** n **faire**

$TI := Id_n$;

Pour j = 1 **à** (i-1) **faire**

/*Les éléments de la ligne j sont mis à jour sauf l'élément */
/* $a_{j,j}$ -noté x ici- déjà modifié à l'étape (i-1) */
Si (j > 1) **alors** $[a_{j,1},\ldots,a_{j,j-1},x, 0,\ldots,0,a_{j,i}] := [a_{j,1},\ldots,a_{j,i}].TI^{(i)}$;

/* Calcul de la transformation $E_j^{(i)}$ */
(s,t,g) := bezout ($a_{j,j}$, $a_{j,i}$);

$$TI := TI.\ \begin{matrix} \\ j \\ i \\ \\ \end{matrix} \begin{matrix} \begin{matrix} j & & i \end{matrix} \\ \begin{bmatrix} 1 & & 0 \\ & s & \frac{-a_{j,i}^{(i,j)}}{g} \\ & t & \frac{a_{j,j}^{(i,j)}}{g} \\ 0 & & 1 \end{bmatrix} \end{matrix}$$

$a_{j,j} := g$;
$a_{j,i} := 0$;
/* Calcul des réductions sur la ligne j */
$RI := Id_n$;

Pour k = 1 **à** (j-1) **faire**

$$q := \left\lfloor \frac{a_{j,k}}{a_{j,j}} \right\rfloor ; \quad RI := RI. \begin{array}{c} \\ \\ j \\ \\ \\ \end{array} \begin{array}{c} k \\ \left[\begin{array}{ccccc} 1 & & & & \\ & \ddots & & & \\ & & \ddots & & \\ & -q & & \ddots & \\ & & & & 1 \end{array}\right] \end{array}$$

/* RI cumule les */
/* réductions de */
/* la ligne j */

$a_{j,k} := a_{j,k} - q.a_{j,j}$;

$TI := TI.RI$; /* TI cumule les opérations effectuées */ /* à l'étape i sur les lignes 1 à i-1 */

$T := T.TI$; /* T cumule les transformations */ /* effectuées. En fin d'algorithme */ /* T=U i.e. A.T=H */

/* Traitement de la ligne i */
/* La ligne qui n'avait pas encore été modifiée rattrape son retard */
$[a_{i,1}, \ldots, a_{i,i}] := [a_{i,1}, \ldots, a_{i,i}] . (T)^{(i)}$;
$RI := Id_n$;
/* Sur la ligne i, on n'effectue que des réductions */
Pour k = 1 **à** (i-1) **faire**

$$q := \left\lfloor \frac{a_{i,k}}{a_{i,i}} \right\rfloor ;$$

$$RI := RI . \begin{array}{c} \\ \\ i \\ \\ \\ \end{array} \begin{array}{c} k \\ \left[\begin{array}{ccccc} 1 & & & & \\ & \ddots & & & \\ & & \ddots & & \\ & -q & & \ddots & \\ & & & & 1 \end{array}\right] \end{array} ;$$

$a_{i,k} := a_{i,k} - q.a_{i,i}$;
Si (i < n) **alors** T := T . RI ;
Fin

Des simplifications pour l'implantation :
Les produits de matrice effectués lors de l'algorithme ne nécessitent souvent pas une mise à jour de tous les coefficients :

- Le produit des réductions à l'étape j de l'étape i est de la forme

$$\begin{array}{c} \\ \\ \\ \\ j \\ \\ \\ \end{array} \begin{array}{c} \\ \\ \\ \\ \left[\begin{array}{ccccc} 1 & & & & \\ & \ddots & & & \\ & & \ddots & & \\ -q_1 & \cdots & -q_{j-1} & \ddots & \\ & & & & 1 \end{array}\right] \end{array}$$

$q_1, q_2, \ldots, q_{j-1}$ sont les (j-1) quotients obtenus, correspondant chacun à une réduction.
Il suffit donc de stocker $q_1, q_2, \ldots, q_j$.

- Les opérations de matrices, à l'étape i, ne modifient que les mineurs principaux d'ordre i.
- L'opération TI := $TI.E_j^{(i)}$ ne modifie que deux colonnes de TI.

VI.4.3. Complexité

a. Résultats expérimentaux

Expérimentalement, l'algorithme ci-dessus est plus rapide que l'algorithme mineur par mineur, du moins dès que la taille du problème traité est suffisamment importante.
L'algorithme avec cumul nécessite davantage de contrôle et ne devient par conséquent intéressant qu'au delà d'un certain seuil. Cependant la complexité du calcul est suffisamment importante pour que ce seuil reste faible : degré > 2 pour des matrices d'ordre 4 (coefficients des polynômes supérieur à 1000).

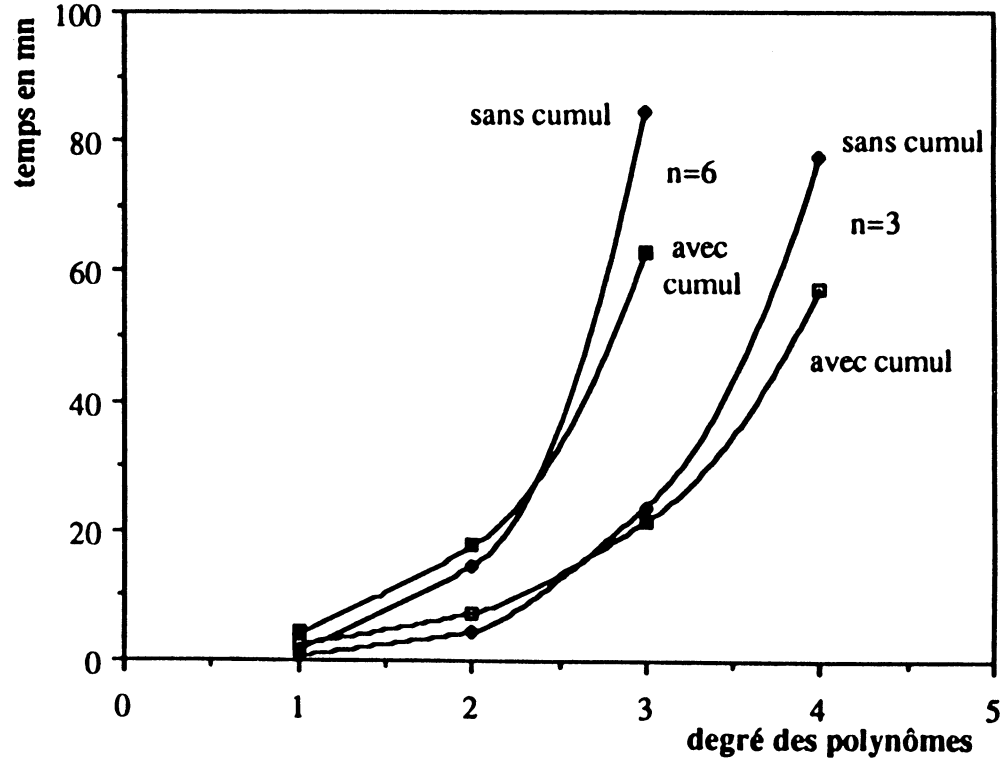


Figure VI.1 : algorithme mineur par mineur avec ou sans cumul sur une matrice d'ordre 4

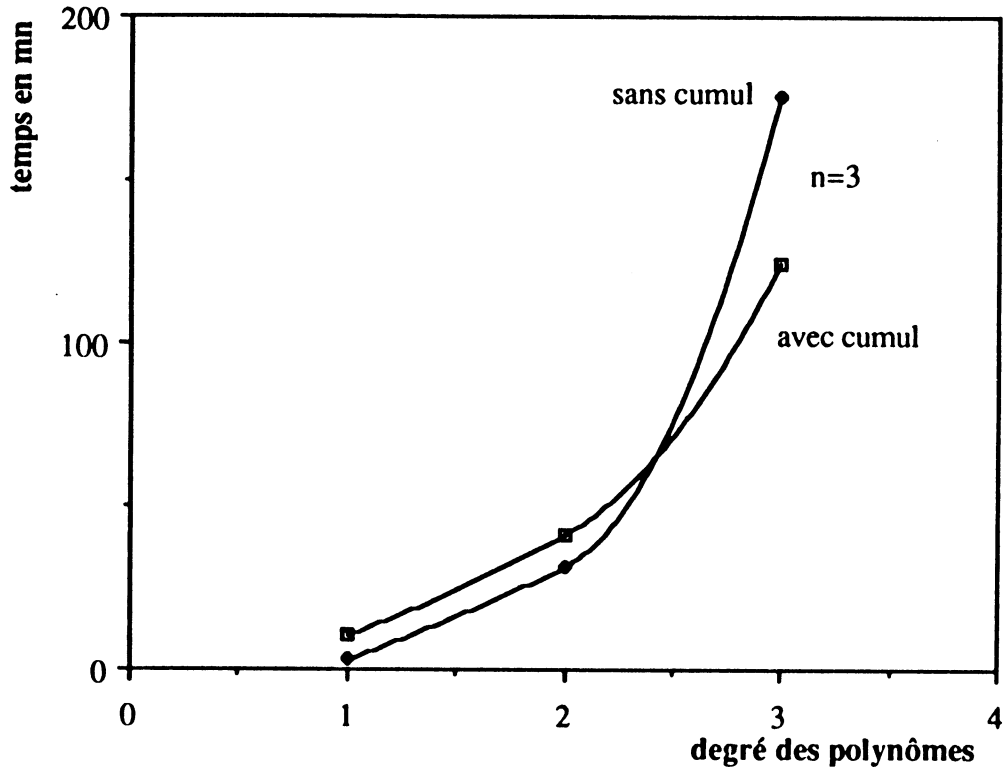


Figure VI.2 : algorithme mineur par mineur avec ou sans cumul sur une matrice d'ordre 5

Les résultats ci-dessus ont été obtenus sur Macsyma pour des matrices d'ordre 4 et 5 de polynômes denses dont les degrés variaient entre 1 et 4. n est le nombre de digits nécessaires à l'écriture décimale des coefficients (entiers) des polynômes de la matrice initiale. Les deux courbes se croisent. C'est un phénomène que nous avons retrouvé sur les autres exemples traités (matrices quelconques dont nous perturbions les degrés et les coefficients).

b. Etude théorique :

A l'étape i :

pour l'élimination sur la ligne j, le nombre d'opérations en précision infinie (on négligera les additions) correspond à

si $j < i$:

- l'application de TI sur la ligne j : $j^2\, T_{pmul}$
- calcul d'un pgcd : T_{ppgcd}
- mise à jour de TI : $4\, i\, T_{pmul}$
- calcul de RI, mise à jour des élements de la ligne j, calcul de TI = TI.RI : $(j-1)\,(T_{pdiv} + 2T_{pmul})$

si $j = i$:

- calcul de T = T.TI : $i^3\, T_{pmul}$
- l'application de T sur la ligne i : $i^2\, T_{pmul}$

- calcul de RI, mise à jour des élements de la ligne i, calcul de T = T.RI : $(i-1)\,(T_{pdiv} + 2T_{pmul})$

On en déduit le coût total de l'algorithme:

$$\sum_{i=1}^{n} \left(\sum_{j=2}^{i} (j^2\, T_{pmul} + (j-1)\,(T_{pdiv} + 2T_{pmul})) + (i-2)\, T_{ppgcd} + (4i(i-2) + i^3)\, T_{pmul}\right)$$

On remarque que l'algorithme avec cumul est du même ordre de complexité que l'algorithme sans cumul. Pourquoi est-il expérimentalement intéressant?
Rappelons tout d'abord que les bornes sur les coefficients des matrices intermédiaires sont :

$$\deg(A^i) \leq (2\,n + 1)\,MD$$

$$\deg(K^i) \leq 2\,n\,MD$$

$$\text{taille}(A^i) \leq 8\,MD^2\,n^3\,\text{taille}(A^1)$$

$$\text{taille}(K^i) \leq 6\,MD^2\,n^3\,\text{taille}(A^1)$$

Dans l'algorithme avec cumul, nous travaillons sur les coefficients des matrices de transformation alors que dans l'algorithme sans cumul nous calculons les coefficients de la matrice A^i au fur et à mesure.
La taille des coefficients des matrices de transformations est d'un facteur 3/4 fois plus petite que la taille des coefficients des matrices A^i. Dans l'algorithme avec cumul, nous travaillons donc sur des coefficients plus petits le plus longtemps possible, ce qui explique les résultats obtenus.

La complexité est cependant très difficile à évaluer en tenant compte de la croissance des coefficients intermédiaires et les deux algorithmes sont par conséquent complexes à comparer.

VI.4.4. Validité de l'algorithme et adéquation au parallélisme

Il est certain que l'algorithme proposé nécessiterait une étude plus précise, à la fois théorique et expérimentale, pour justifier des résultats obtenus. Cependant, le calcul de la forme d'Hermite de matrices à coefficients polynomiaux est un problème très complexe et une méthode qui donne de bons résultats expérimentaux est par conséquent très intéressante.

Mais l'avantage le plus marquant du cumul des opérations séquentielles est de rassembler le parallélisme inhérent de l'algorithme mineur par mineur autour des opérations de produits matrice-vecteur ou matrice-matrice, deux opérations qui se prêtent particulièrement bien à la parallélisation.

VI.5. Applications

VI.5.1. Forme normale de Smith et forme normale de Jordan [MAR64], [NEW72], [OZE87].

Nous voyons ici comment calculer la forme de Smith sachant calculer la forme d'Hermite et comment calculer la forme de Jordan d'une matrice à coefficients dans $\mathbb{Q}$, sachant calculer la forme de Smith d'une matrice à coefficients dans $\mathbb{Q}[x]$.
Kannan et Bachem ont donné un algorithme polynomial pour calculer la forme normale de Smith, utilisant des calculs de forme normale d'Hermite. Cet algorithme fait intervenir à la fois des opérations de colonnes pour le calcul de forme d'Hermite de certains mineurs et des opérations de lignes pour éliminer les coefficients sous-diagonaux. Nous donnons ci-dessous l'algorithme, pour plus de précision le lecteur peut se référer à [KAN79].

Algorithme de Kannan et Bachem [KAN79]

```
procedure FNS(n, A)
Début
pour i = 1...(n-1) faire

    Mise_sous_forme_Smith := Faux
    tant que non(Mise_sous_forme_Smith) faire

        { élimination des éléments de la ième colonne }
        hligne(i, A)

        { mise sous forme d'Hermite du mineur inférieur d'ordre i }
        hermite(i, A)

        si (A_{i+1, i+1} est le seul élément non nul en colonne i+1) alors

            si   (∃ j, k (i+1 ≤ j ≤ k, i+1 ≤ k ≤ n) t.q.
                                        A_{j,k} ne divise pas A_{i+1, i+1})
                 alors ajouter colonne k à la colonne i+1 dans A

            sinon Mise_sous_forme_Smith := Vrai

Fin
```

où Hermite(i, A) calcule la forme normale d'Hermite de A privée de ses i premières colonnes et i premières lignes, et hligne(i, A) fait apparaître des zéros sur la ième colonne de A, sous la diagonale, par des opérations unimodulaires sur les lignes de A.

De la forme de Smith de (xI_n - A) à la forme de Jordan de A

Etant donné une matrice A à coefficients dans $\mathbb{Q}$, nous allons montrer qu'à partir de la forme de Smith de xI_n - A, il est possible de calculer une forme normale, appelée forme normale de Jordan de A.

Définition:
Soient A et B deux matrices à coefficients dans $\mathbb{Q}$. On dit que *A est semblable à B* si il existe une matrice inversible U à coefficients dans $\mathbb{Q}$ telle que $A = U^{-1}BU$.

Théorème:
Deux matrices A et B de $M_n(\mathbb{Q})$ sont semblables sur $\mathbb{Q}$ si et seulement si leurs *matrices caractéristiques* xI_n - A et xI_n - B sont équivalentes sur $\mathbb{Q}[x]$, c'est à dire, ont même facteurs invariants.

Les facteurs invariants de la matrice caractéristique de A sont appelés *invariants de similarité de A.*

Définition:
Une *matrice compagnon* est une matrice carrée de la forme suivante:

$$C = \begin{pmatrix} 0 & 1 & 0 & & \cdots & & 0 \\ 0 & 0 & 1 & 0 & & \cdots & 0 \\ \vdots & \vdots & & \ddots & \ddots & \ddots & \vdots \\ & & & & & & 0 \\ 0 & 0 & & \cdots & & 0 & 1 \\ a_0 & a_1 & & \cdots & & a_{n-2} & a_{n-1} \end{pmatrix}$$

On associe à C le polynôme $p(x) = x^n - a_{n-1}x^{n-1} - \ldots - a_0 = \text{dét}(x.I - C)$.

Réciproquement , à tout polynôme unitaire p de degré n, on associe une unique matrice compagnon d'ordre n.

Soit A de $M_n(\mathbb{Q})$ et $P_A(x) = \text{dét}(xI_n - A)$ son polynôme caractéristique de racines $\lambda_1, \lambda_2, \ldots, \lambda_n$ appartenant à C.

Notons $J_r(\lambda_i)$ le bloc d'ordre r, appelé bloc de Jordan:

$$J_r(\lambda_i) = \begin{pmatrix} \lambda_i & 1 & 0 & \cdots & 0 \\ 0 & \lambda_i & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \lambda_i & 1 \\ 0 & \cdots & \cdots & 0 & \lambda_i \end{pmatrix}$$

Théorème:

(i) A est semblable à une matrice bloc-diagonale

$$J = \mathrm{diag}(J_{r_1}(\lambda_1), \ldots, J_{r_2}(\lambda_1), \ldots, J_{r_p}(\lambda_i), \ldots, J_{r_q}(\lambda_k))$$

$$\text{avec } \sum_{p=1}^{q} r_p = n.$$

(ii) J est unique à une permutation près des blocs $J_r(\lambda_i)$.

J est appelée *forme de Jordan* de A.

Corollaire: soit $p = p_1 \times p_2^2 \times \ldots \times p_k^k$ la décomposition sans carrés de p (les p_i étant premiers entre eux et sans facteurs carrés), et soient $\lambda_{i,1}, \lambda_{i,2}, \ldots, \lambda_{i,d_i}$ les d_i racines distinctes de p_i.
Alors la forme de Jordan de la matrice compagnon associée à P est:

$$J = \mathrm{diag}_{\substack{i=1 \text{ à } k \\ j=1 \text{ à } d_i}} (J_i(\lambda_{i,j}))$$

Théorème(forme normale de Frobenius)[MAR64]: soit A de $M_n(\mathbb{Q})$, alors A est semblable sur $\mathbb{Q}$ à la somme directe des matrices compagnons de ses invariants de similarité différents de 1.

Théorème: la forme de Jordan de A est la somme directe des formes de Jordan des matrices compagnons de la forme de Frobenius de A, donc des matrices compagnons associées aux facteurs invariants de similarité de A différents de 1.

On peut donc, connaissant la forme de Smith de $(xI_n - A)$, et par suite ses facteurs invariants, calculer la forme de Jordan de A. On peut se rapporter à la thèse de Patrick Ozello sur le calcul de la forme de Frobenius et de la forme de Jordan [OZE87].

VI.5.2. Application à l'automatique

Un des problèmes classiques rencontré en théorie du contrôle réside en la résolution d'un système d'équations différentielles linéaires à coefficients constants, fonction du temps. Le système admet une entrée scalaire u(t) et une sortie scalaire y(t).
Il s'agit alors d'étudier le comportement du système dans des circonstances données.
Le système est de la forme:

$$\begin{cases} \dot{x}(t) = A\,x(t) + B\,u(t) \\ y(t) = C\,x(t) \end{cases}$$

Il lie un vecteur entrée u(t) de K^m, à un vecteur sortie y(t) de K^p, par un vecteur d'état x(t) de K^n (K corps donné), où A est dans K^{nxn}, B dans K^{nxm}, C dans K^{pxn}.

Un système dynamique linéaire est alors défini par le triplet (A, B, C).

Soient respectivement Y = Y(s) et U = U(s) les transformées de Laplace de y et u.
Elles sont liées par la relation:

$$Y(s) = T(s)\,U(s)$$

où T(s) est appelée matrice de transfert du système et est donnée par

$$T(s) = C\,(\,s\,I - A\,)^{-1}\,B.$$

T(s) est une matrice dont les coefficients sont des fractions rationnelles en s dont le degré du numérateur est inférieur au degré du dénominateur.
A chaque système est associé une unique fonction de transfert. Par contre, à une matrice de transfert T est associée toute une classe de systèmes, appelée réalisation de T, chacun ayant T comme matrice de transfert.
La matrice T définit le transfert entrée-sortie du système mais cache le comportement interne (état).

Différents problèmes peuvent alors se poser:

- étant donnée une fonction de transfert T, peut-on construire un système (A, B, C) dont T soit la matrice de transfert ?

- parmi tous les systèmes possibles (s'il en existe), peut-on en choisir un ayant une représentation plus agréable ?

Une première étape, dans la résolution de ces problèmes, consiste à se ramener à une représentation canonique de la matrice T.

Posons $\psi = \psi(s)$ le polynôme unitaire (coefficient de tête égal à 1) qui est le produit des dénominateurs des coefficients de T. La matrice ψ T est une matrice polynomiale.
Soit H la forme normale d'Hermite de $\psi(s)\ T(s) = H(s)\ U(s)$, où U(s) est une matrice unimodulaire. $\frac{1}{\psi(s)} H(s)$ est une matrice équivalente à droite à T(s), et H constitue une représentation canonique de T.
Cette forme réduite permet de donner une représentation plus agréable de la matrice de transfert, également plus simple à manipuler.

La forme de Smith est une autre représentation de T, mais plus complexe à obtenir.
On peut se référer à [HEY70] pour des renseignements plus précis sur la théorie des systèmes.

Conclusion

La forme normale d'Hermite et la forme de Smith jouent un rôle important en théorie des groupes algébriques [NEW72], en théorie des systèmes [HEY70] et en programmation en nombres entiers [GAR72].

ANNEXE

COMPARAISON DES ALGORITHMES DE CALCUL DE LA FORME NORMALE D'HERMITE

L'exemple qui suit est extrait d'une session Macsyma. "hmineur" et "hermod" sont deux procédures que nous avons programmé et qui permettent de calculer la forme normale d'Hermite d'une matrice respectivement par l'algorithme mineur par mineur et par l'algorithme modulo d.
Les paramètres en entrée sont : la matrice initiale, sa dimension.

```
(c15) m;

                [   75      42      45      84   ]
                [                                ]
                [  326     182     195     364   ]
                [                                ]
                [ - 3267   729     781     1458  ]
                [                                ]
                [  1190   - 2742  - 2695  - 5482 ]

(c16) hmineur(m,4);

                % Traitement du mineur principal d'ordre 2 %

                   [    3        0       45      84   ]
                   [                                  ]
                   [    8       14      195     364   ]
                   [                                  ]
                   [  22896   - 63963   781     1458  ]
                   [                                  ]
                   [ - 30628   85210   - 2695  - 5482 ]

                % Traitement du mineur principal d'ordre 3 %

                  [     3          0          0        84   ]
                  [                                        ]
                  [     0          1          0       364   ]
                  [                                        ]
                  [     0          0          1       1458  ]
                  [                                        ]
                  [ 202038052  - 15527785  - 3400  - 5482 ]
```

```
                              % Resultat %

                              [ 3  0  0  0 ]
                              [            ]
                              [ 0  1  0  0 ]
(d16)                         [            ]
                              [ 0  0  1  0 ]
                              [            ]
                              [ 0  1  0  2 ]
```

```
(c17) hermmod(m,4);

                    % Apres elimination sur la premiere ligne %

                    [ 3  0  0  0  0  0  0  0 ]
                    [                        ]
                    [ 2  4  3  2  2  6  0  0 ]
                    [                        ]
                    [ 0  3  1  0  0  0  6  0 ]
                    [                        ]
                    [ 2  2  5  0  2  0  0  6 ]

                    % Apres elimination sur la seconde ligne %

                    [ 3  0  0  0  0  0  0  0 ]
                    [                        ]
                    [ 2  1  0  0  0  0  0  0 ]
                    [                        ]
                    [ 0  2  1  2  2  0  6  0 ]
                    [                        ]
                    [ 2  3  2  0  2  0  0  6 ]

                    % Apres reduction sur la seconde ligne %

                    [ 3  0  0  0  0  0  0  0 ]
                    [                        ]
                    [ 0  1  0  0  0  0  0  0 ]
                    [                        ]
                    [ 2  2  1  2  2  0  6  0 ]
                    [                        ]
                    [ 2  3  2  0  2  0  0  6 ]

                   % Apres elimination sur la troisieme ligne %

                    [ 3  0  0  0  0  0  0  0 ]
                    [                        ]
                    [ 0  1  0  0  0  0  0  0 ]
```

```
[                       ]
[ 2  2  1  0  0  0  0  0 ]
[                       ]
[ 2  3  2  2  4  0  0  6 ]
```

```
% Apres reduction sur la troisieme ligne %
```

```
[ 3  0  0  0  0  0  0  0 ]
[                       ]
[ 0  1  0  0  0  0  0  0 ]
[                       ]
[ 0  0  1  0  0  0  0  0 ]
[                       ]
[ 4  5  2  2  4  0  0  6 ]
```

```
% Apres reduction sur la quatrieme ligne %
```

```
[ 3  0  0  0  0  0  0  0 ]
[                       ]
[ 0  1  0  0  0  0  0  0 ]
[                       ]
[ 0  0  1  0  0  0  0  0 ]
[                       ]
[ 0  1  0  2  0  0  0  0 ]
```

```
% Resultat %
```

(d17)

```
[ 3  0  0  0 ]
[            ]
[ 0  1  0  0 ]
[            ]
[ 0  0  1  0 ]
[            ]
[ 0  1  0  2 ]
```

Remarquons que dans cet exemple l'algorithme modulo d s'avère nettement plus interessant que l'algorithme mineur par mineur. Nous avons en effet choisi une matrice dont le déterminant est beaucoup plus petit que les coefficienst. Notons que, sur de nombreux autres exemples la difference de taille entre les coefficients intermédiaires obtenus dans les deux algorithmes est beaucoup moins nette et que l'algorithme modulo d nécessite un nombre d'éliminations plus important, mais beaucoup moins de réductions.

Deuxième Partie

PARALLELISATION DU CALCUL DE LA FORME NORMALE D'HERMITE

SOMMAIRE DE LA DEUXIEME PARTIE

CHAPITRE I RESULTATS THEORIQUES

I.1. Introduction....81
I.2. Modèle théorique - La classe $\mathcal{NC}$....81
I.3. Complexité théorique du calcul de la forme d'Hermite....83
I.4. Conclusion....88

CHAPITRE II ALGORITHMES PARALLELES

II.1. Introduction....89
II.2. Ordonnancements pour machine à mémoire partagée....92
II.2.1. Tâches et systèmes de tâches....92
II.2.2. Ordonnancement pour l'algorithme modulo d....93
II.3. Machine à mémoire distribuée....102
II.3.1. Le modèle....102
II.3.2. Matrices à coefficients entiers....103
II.3.2.1. Répartition par colonnes....104
II.3.2.1.1.Description....104
II.3.2.1.2.Algorithme HC....106
II.3.2.2. Répartition par lignes....112
II.3.2.2.1.Description....112
II.3.2.2.2.Algorithme HL....113
II.3.2.3. Complexité des deux algorithmes....116
II.4. Algorithme parallèle pour le calcul de la forme normale d'Hermite d'une matrice à coefficients polynomiaux....121
II.4.1. Précédence des tâches....121
II.4.2. Algorithme parallèle....122
II.4.2.1. Description....122
II.4.2.2. Algorithme MPL....122
II.4.4.3. Etude de la complexité....124

CHAPITRE III. IMPLANTATION

III.1. Cas entier 126
III.1.1. Programmation 126
III.1.1.1. Outils arithmétiques 126
III.1.1.2. Primitives de communication 127
III.1.1.3. Gestion mémoire 128
III.1.1.4. Initialisation de la matrice 129
III.1.2. Résultats expérimentaux 129
III.1.2.1. Temps d'exécution 129
III.1.2.2. Efficacité (classique et Gustafson) 133
III.1.2.3. Mémoire 137
III.2. Cas polynomial 139
III.2.1. Programmation 139
III.2.1.1. Outils arithmétiques 139
III.2.1.2. Primitives de communication 139
III.2.1.3. Gestion mémoire 140
III.2.1.4. Initialisation de la matrice 140
III.2.2. Résultats expérimentaux 141
III.2.2.1. Dans $\mathbb{Q}[x]$ 141
III.2.2.2. Dans $\mathbb{Z}/p\mathbb{Z}$ 142

CONCLUSION

BIBLIOGRAPHIE

CHAPITRE I

RESULTATS THEORIQUES

I.1. Introduction

Le parallélisme en calcul formel est une discipline jeune. Les résultats existants sont essentiellement des résultats théoriques basés sur les modèles de complexité de N.Pippenger [PIP79], et S.A.Cook [COO83].
A.Borodin, J.Von Zur Gathen, J.Hopcroft [BOR82] ont donné des résultats de complexité pour des problèmes d'algèbre linéaire (système linéaire, déterminant, polynôme caractéristique, ...).
E.Kaltofen, M.S.Krishnamoorthy et D.Saunders ont établi en 1987 [KAL87] un résultat de complexité pour la forme d'Hermite; nous le présenterons dans ce chapitre.

Notons que ces résultats théoriques sont pour la plupart établis en s'autorisant, à priori, un nombre infini de processeurs, dépendant polynomialement de la taille du problème à résoudre: la motivation est alors de réduire le plus possible le temps total de résolution. Par exemple, Eberly [EBE85] propose, pour le calcul du rang d'une matrice bande, une famille de réseaux arithmétiques de taille $n^{13} \log^4 n$ pour résoudre le problème en un temps $O(\log n \log m + \log n \log \log n)$ où m est la largeur de la bande).

Des études plus expérimentales ont été réalisées; citons notamment les travaux de S.Watt [WAT88] et C.Ponder [PON88].

I.2. Modèle théorique - La classe $\mathcal{NC}$

Le développement récent des machines parallèles a motivé la création d'un nouvel axe de recherche sur la complexité: la théorie du calcul parallèle synchrone, dans laquelle le nombre de processeurs est borné par un paramètre H(n) (au niveau matériel), de même que l'espace mémoire est borné par un paramètre S(n) en théorie de complexité séquentielle classique. H(n) est en général un polynôme de la variable n (classe $\mathcal{NC}$).
De nombreux modèles ont été proposés pour le calcul parallèle.

La première classe de modèle correspond au modèle à mémoire partagée (PRAM) dans lequel un grand nombre de processeurs communiquent via une mémoire à accès aléatoire. Beaucoup d'algorithmes ont été mis au point pour ce modèle. Cependant, pour des études de complexité précises, ce modèle n'est pas très satisfaisant car trop de détails sont encore à clarifier:

- la façon dont sont résolus les conflits de lecture et d'écriture

(contraintes CRCW, CREW ou EREW par exemple, pour spécifier si les lectures (respectivement les écritures) sont concurrentes ou exclusives).

- les opérations de base permises sur chaque processeur.

Des modèles plus fins, tiennent compte de ces facteurs : dans ce chapitre, nous nous reportons au modèle plus précis, proposé par Borodin [BOR77].

Il est essentiel de limiter la complexité physique d'un circuit pour résoudre un problème donné de façon à ce que le temps de traversée du circuit et son nombre de portes (logiques) suffisent pour évaluer la complexité algorithmique du problème considéré. On considère donc une famille de circuits *uniforme*, c'est à dire qui peuvent être décrits "facilement" en fonction de la taille du problème traité [RUZ81].

Nous nous restreignons ici au cas de la log-uniformité [COO83].

Définition: Une famille de circuits (booléens) est dite "uniforme" si une machine de Turing peut générer une description du circuit à n entrées, sur un espace O(log(n)) .

Définition : Une machine parallèle est une famille uniforme de circuits booléens acycliques $(B_n)_{n\in \mathbb{N}}$ tels que B_n a n entrées [COO87].

Notations : H(n) désigne le nombre de portes de B_n au niveau matériel, et T(n) désigne le temps de calcul parallèle, ou la profondeur du circuit, c'est à dire la longueur du plus long chemin (orienté) dans le circuit.

Ce type de modèle se justifie pratiquement du fait que la plupart des machines existantes sont construites à partir de circuits booléens. D'une part, à un algorithme parallèle donné on peut associer un crcuit booléen le réalisant. D'autre part, on sait réaliser matériellement, à n fixé, un circuit booléen physique dont on connaît la description logique.

N.B. Dans la suite, on désignera par le terme *processeur* tout circuit ayant un nombre fini de portes, indépendant de la taille du problème traité.

Le problème de base du parallélisme est alors de définir les problèmes qui peuvent être résolus plus rapidement avec plusieurs processeurs plutôt qu'avec un seul.

Nicolas Pippenger [PIP79] formalise cette question en définissant la classe $\mathcal{NC}$ des problèmes qui peuvent être résolus très rapidement.

Soit P un problème et P_n une instance de ce problème de taille n (c'est à dire que P_n a n entrées).

Définition : P appartient à $\mathcal{NC}$ si et seulement si pour tout n il existe une famille uniforme de circuits booléens B_n ayant n entrées qui résout le problème P_n en un temps poly-log et qui a un nombre polynomial de processeurs. C'est à dire :

temps $T(n) = (\log n)^{O(1)}$ sur une machine parallèle avec un nombre "praticable" de processeurs $H(n) = n^{O(1)}$

Nous rappelons ci-dessous les définitions des classes $\mathcal{NC}^k$ définies par S.A.Cook [COO83] qui partitionnent la classe $\mathcal{NC}$.

Définition: la classe $\mathcal{NC}^k$ est constituée de toutes les fonctions f, calculables par une famille de circuits uniforme de taille polynomiale et de profondeur $O(\log^k n)$.

Autrement dit, $\mathcal{NC}^k$ est la classe de toutes les fonctions qu'il est possible de calculer avec un circuit ayant un nombre polynomial de cellules, et une profondeur en $O(\log^k n)$.
La classe $\mathcal{NC}$ correspond alors à l'union de toutes les classes $\mathcal{NC}^k$:

$$\mathcal{NC} = \bigcup_{k=0}^{\infty} \mathcal{NC}^k$$

Il est important de classifier les problèmes, et donc de définir une relation d'ordre entre les problèmes, liée à leur complexité.

Définition: on dit qu'une fonction f est $\mathcal{NC}^1$-réductible à une fonction g ($f \leq_{\mathcal{NC}^1} g$) s'il existe une famille de circuits uniforme qui calcule f en un temps logarithmique, et qui utilise des nœuds *oracles* pour calculer g.
Un nœud *oracle* pour g est un nœud ayant r entrées $(y_1,\ldots,y_r)$ et s sorties $(z_1,\ldots,z_s)$ et qui calcule la fonction $(z_1,\ldots,z_s) = g(y_1,\ldots,y_r)$. La profondeur d'un tel nœud est assimilé à : $\lceil \text{Log } (r) \rceil$.

D'une manière évidente, $\leq_{\mathcal{NC}^1}$ est réflexive et transitive.

I.3. Complexité théorique du calcul de la forme d'Hermite

Dans cette partie, nous présentons un résultat de Kaltofen, Krishnamoorthy et Saunders [KAL87] sur la complexité du calcul parallèle de la forme normale d'Hermite d'une matrice à coefficients dans K[x], où K est un corps. Nous montrons que ce problème est $\mathcal{NC}^1$-réductible au problème de la résolution d'un système linéaire dans K. Nous noterons HERMITE le problème du calcul de la forme normale d'Hermite et SYSTEME LINEAIRE le problème de résolution d'un système d'équations linéaires (calcul d'une solution si elle existe, sinon indication qu'il n'y en a pas).

Nous noterons K_n l'ensemble des matrices carrées d'ordre n à coefficients dans K[x].

Soit A dans K_n, on notera $a_{i,j}(x)$ la composante d'indice (i, j) de A, et $a_{i,j,k}$ le coefficient de x^k dans $a_{i,j}(x)$.
Etant donnés:

- une matrice régulière A dans K_n dont les coefficients sont de degrés bornés par d,
- un vecteur $(d_1, d_2, \ldots, d_n)$ d'entiers positifs,

On considère le problème $\mathcal{P}$ suivant:

$\mathcal{P}(A, (d_1, d_2, \ldots, d_n))$:

Trouver P, G $\in K_n$, s'ils existent, tels que

$$A\,P = G \qquad (\mathcal{F})$$

avec
$$\begin{cases} G \text{ triangulaire inférieure} \\ \deg(g_{i,i}) = d_i\ ,\ g_{i,j,d_i} = 1 \text{ et } \deg(g_{ij}) < d_i \\ \deg(p_{i,j}) \le n\,d + \max_{1 \le i \le n}(d_i) \end{cases}$$

Sinon, indiquer qu'il n'y a pas de solution.

Proposition:

(1) le problème $\mathcal{P}(A, (d_1, d_2, \ldots, d_n))$ a au moins une solution, si et seulement si chaque d_i est supérieur ou égal au degré du $i^{ème}$ coefficient diagonal de la forme normale d'Hermite de A

(2) si chaque d_i est exactement le degré du $i^{ième}$ coefficient de la forme normale d'Hermite de A, alors $\mathcal{P}(A, (d_1, d_2, \ldots, d_n))$ admet une unique solution, P est unimodulaire, et G est alors la forme normale d'Hermite de A.

Preuve: cf. [KAL87].

Remarquons que le degré des coefficients de la forme d'Hermite de A est borné par le degré du déterminant de A donc par e = n d.

Lemme 1:
pour tout i, le problème:

$$\mathcal{P}(A, (d_1 = e, \ldots, d_{i-1} = e, d_i = \deg(h_{ii}), d_{i+1} = e, \ldots, d_n = e))$$

admet une solution.

Preuve:
Soit H la forme d'Hermite de A et soit U la matrice unimodulaire telle que AU = H. Soit D la matrice diagonale définie par:

$$D = \mathrm{diag}(x^{\text{e-deg}(h_{11})}, \ldots, x^{\text{e-deg}(h_{i-1,i-1})}, 1, x^{\text{e-deg}(h_{i+1,i+1})}, \ldots, x^{\text{e-deg}(h_{nn})})$$

Il est clair que P = UD et G = HD est une solution au système ($\mathfrak{P}$). Montrons que c'est également une solution du problème, c'est à dire que la borne sur les coefficients de P est valide.
Puisque A P = G et dét(A) P = adj(A) G, les degrés des coefficients de P sont bornés par les degrés des coefficients de adj(A) G, donc par :

$$(n - 1)\, d + e$$ CQFD.

Soit A une matrice nxn à coefficients dans K[x] dont les composantes sont des polynômes de degré borné par d et de coefficients de taille bornée par L (représentable sur L bits). A est alors représentable en binaire sur $(n^2.d.L)$ bits, et $(n^2.d.L)$ est donc la taille de A.

Lemme 2: pour $d_i \leq nd$ le sytème linéaire ($\mathfrak{P}$) est constitué de $O(n^3\, d)$ équations en $O(n^3\, d)$ inconnues. Ses entrées sont de taille L.

Preuve:
On a d'une part:
$\deg(p_{i,j}) \leq n\, d + \max_{1 \leq i \leq n}(d_i)$, et $d_i \leq nd$, donc P contient au plus $(n^2\,(2nd + 1))$ inconnues.
D'autre part, G possède au plus $(n^2/2)$ composantes non nulles, le coefficient de tête du terme diagonal est connu et tous les degrés des termes sous-diagonaux sont bornés par e. D'où, G possède au plus $((n^2/2)\, nd)$ inconnues.
Finalement, le système présente $O(n^3 d)$ inconnues.

A chaque coefficient d'un polynôme d'une composante de G on peut faire correspondre une équation linéaire dans K. On a donc à résoudre un vrai système linéaire en $O(n^3 d)$ inconnues.
Le théorème qui suit est basé sur les remarques (1) et (2) de la proposition. Pour construire une $\mathcal{NC}^1$-réduction du problème HERMITE au problème SYSTEME LINEAIRE nous allons considérer une famille de n(e + 1) circuits, capable de résoudre le problème
$\mathfrak{P}(A, (d_1 = e, \ldots, d_{i-1} = e, d_i = j, d_{i+1} = e, \ldots, d_n = e))$ $(i = 1, \ldots, n;\ j = 0, \ldots, e)$ et qui indiquent si le probléme admet une solution. Les d_i minimums pour lesquels nous obtenons un résultat positif nous indiquent le degré du $i^{\text{ième}}$ coefficient diagonal de la forme normale d'Hermite.

Théorème 1: Le problème HERMITE sur K[x] est $\mathcal{NC}^1$-réductible au problème SYSTEME LINEAIRE sur K.

Preuve : Soit $C_{n^3.d}$ un circuit ayant n entrées $d_1, d_2, \ldots, d_n$ et une sortie, qui résout le problème $\mathcal{P}(A, (d_1, d_2, \ldots, d_n))$: il renvoie $+\infty$ s'il n'y a pas de solution et d_i sinon, où la valeur de i sera précisée par la suite.

Soit $\mathcal{M}_{e+1}$ un circuit ayant (e+1) entrées et une sortie qui calcule le minimum de ses entrées.

Considérons le circuit de la figure I.1, dans lequel on a injecté les valeurs précisées sur le schéma pour n (e + 1) circuits de type $C_{n^3.d}$.
Il est clair que le circuit $C_{n^3.d}$ du bas récupère en entrée un vecteur $(d_1,d_2,\ldots,d_n)$, où d_i, $1 \leq i \leq n$, est le degré de la $i^{ième}$ composante de la forme d'Hermite de A. Ce circuit peut alors résoudre le problème $\mathcal{P}$ pour ces valeurs, et donc calculer la forme d'Hermite de A.

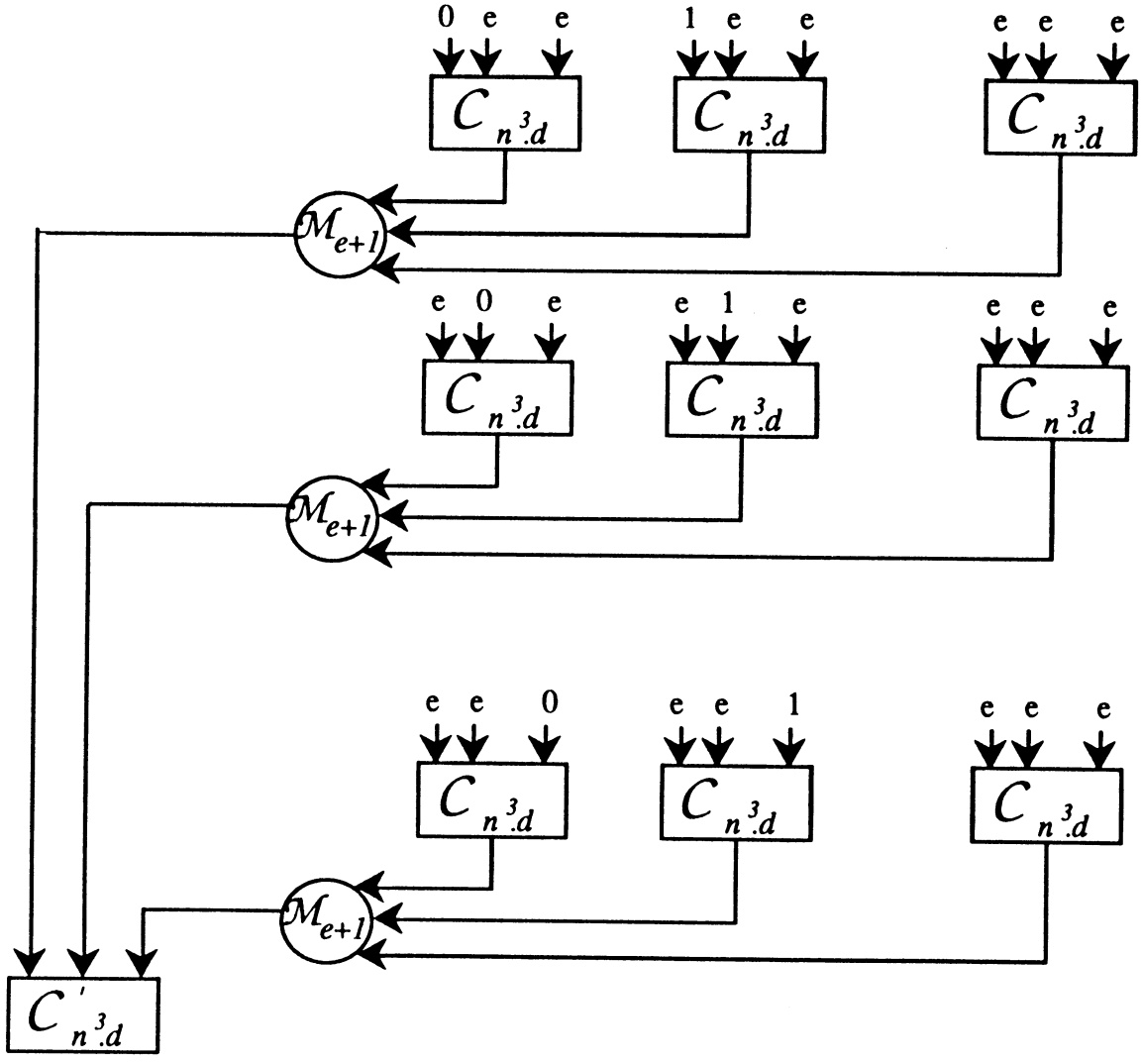


Figure I.1 : $\mathcal{NC}^1$-réduction de la forme d'Hermite à la résolution d'un système linéaire

Toutes les cellules C_{n^3d} résolvent en parallèle un système linéaire de taille n^3d. Soit $C_{SL}(n^3d)$ le coût de la résolution de ce système.
La profondeur due à la porte oracle associée est -cf définition de la porte oracle- d'après le lemme 2 : O(Log(taille(A))).
La profondeur de chacune des n.(e+1) cellules ainsi construites est donc :

$$O(Log(taille(A)))$$

Chaque cellule de ce type retourne d_i si le système admet une solution pour $d_i=j$, et $+\infty$ sinon.

Les sorties d'une même ligne (cf figure I.1) alimentent les (e+1) entrées de la cellule $\mathcal{M}_{e+1}$ associée à la ligne. Ce circuit calcule alors le minimum de ses entrées. D'après la proposition précédente, ce minimum existe et est fini.
La profondeur d'un circuit $\mathcal{M}_{e+1}$ est (calcul du minimum de (e+1) nombres): $O(Log\ (e+1)) \leq O(Log(taille(A)))$

Le dernier processeur C'_{n^3d} résout avec les d_i convenables un système linéaire de taille n^3d, qui délivre en sortie la forme normale d'Hermite de A (d'après la proposition).

La profondeur totale du circuit est donc finalement :

$$\boxed{O(Log(taille(A)))}$$

On en déduit donc que l'on a construit un circuit qui résout le problème HERMITE avec un nombre polynomial de cellules, et qui a même profondeur que le circuit résolvant le problème SYSTEME LINEAIRE.
HERMITE est donc $\mathcal{NC}^1$ réductible au problème SYSTEME LINEAIRE.

Cas particulier important: $\mathbf{K \equiv \mathbb{Q}}$

Dans le cas de $\mathbb{Q}[x]$, la profondeur d'un circuit résolvant un système linéaire de taille n^3d est [BOR82]:

$$C_{SL}(n) = O(Log^2\ n)$$

Le problème HERMITE est $\mathcal{NC}^1$-réductible au problème de la résolution d'un système linéaire, problème de classe $\mathcal{NC}^2$ dans $\mathbb{Q}[x]$, on en déduit que le calcul de la forme normale d'Hermite est au plus $\mathcal{NC}^2$ ($\mathcal{NC}^1$ ou $\mathcal{NC}^2$).

I.4. Conclusion

Cette étude théorique nous permet de mesurer le degré de parallélisme inhérent d'un problème et donc de savoir s'il est utile ou non d'envisager une approche parallèle.

Le calcul de la forme d'Hermite d'une matrice est dans la classe $\mathcal{NC}$ et possède donc un degré de parallélisme important.

Différentes études théoriques ont permis de classifier certains problèmes algébriques comme, par exemple, le calcul du déterminant, l'inversion d'une matrice et le calcul du polynôme caractéristique [BOR82] (ces problèmes étant, pour une matrice de rang n, à coefficients entiers sur n bits, dans la classe $\mathcal{NC}^2$).

Cette étude théorique montre qu'il est intéressant d'aborder une parallélisation effective du problème. Dans la suite de cette seconde partie, nous envisageons deux niveaux d'études, correspondant à deux modèles parallèles différents : moins bien définis formellement que le modèle de Cook, ils présentent l'intérêt d'être proches des architectures parallèles existantes.

Nous considérons, dans un premier temps, un modèle PRAM dans lequel les processeurs communiquent via une mémoire partagée. Les lectures et les écritures sont exclusives (modèle EREW). Nous négligeons les temps de lecture et d'écriture en mémoire.

Dans un deuxième temps nous nous plaçons dans le cadre d'une machine à mémoire distribuée. Nous considérerons un réseau de processeurs qui s'échangent des messages par des liens physiques. Les communications fonctionnnent selon le principe du rendez-vous : ici, on ne peut plus négliger ces temps de communication, qui sont bien sûr linéaires en fonction de la taille des données échangées lorsque un processeur communique à l'un de ses voisins immédiats. Ce modèle est justifié en pratique par une implantation sur un hypercube (FPS T40).

CHAPITRE II

ALGORITHMES PARALLELES

II.1. Introduction

La notion de tâches parallèles est contemporaine du modèle séquentiel. Les premiers ordinateurs parallèles datent des années 60 (Illiac IV par exemple, ou la machine SIMD Pilot ACE [WIL55] [DAV90]); ils connaissent cependant un grand essor depuis le début des années 70.
Un ordinateur parallèle est formé de plusieurs processeurs pouvant exécuter simultanément plusieurs tâches, une tâche étant une unité indivisible de traitement .
Flynn a proposé en 1966 [FLY66] une classification des architectures parallèles. Dans cette classification, basée sur la distinction flots d'instructions, flots de données, différents types de machines sont distingués.

Les machines multiprocesseurs appartiennent à la classe MIMD (Multiple Intructions Multiple Data). La particularité de cette classe de machines est que chaque processeur possède sa propre unité de contrôle.

Chaque processeur effectue des opérations sur ses données, va chercher ou remplacer des données en mémoire, communique avec d'autres tâches.

Les premières machines de ce type sont constituées d'un certain nombre de processeurs ayant accès à une mémoire globale (on multiplie les organes de traitement): ce sont les machines à mémoire partagée.

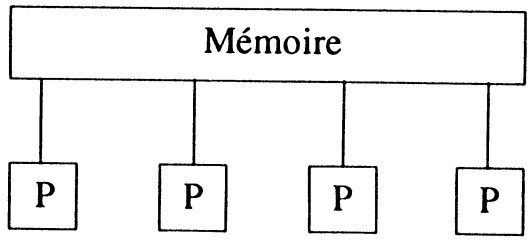


Figure II.1 : Machine MIMD à mémoire partagée

L'accès à une même mémoire limite le nombre de tâches que les processeurs peuvent effectuer simultanément et peut engendrer des problèmes de conflits mémoire. Plus le nombre de processeurs est important, plus ce phénomène est amplifié et les performances tendent à diminuer.
Pour diminuer ce problème on peut associer à chaque processeur une mémoire locale qui lui octroie une certaine indépendance (Cray 2).

La seconde génération de machines MIMD est constituée de processeurs possédant chacun une mémoire locale assez importante, la mémoire globale est supprimée, ce sont les machines à mémoire distribuée. Des langages de spécification permettent d'assurer la cohérence des communications.

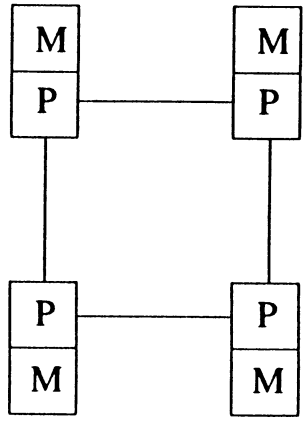


Figure II.2 : Machine MIMD à mémoire distribuée

Les processeurs sont organisés autour d'un réseau d'interconnexions afin qu'ils puissent communiquer.
Une tâche est susceptible d'envoyer un message à n'importe quelle autre tâche à n'importe quel moment; on dit que les envois sont asynchrones.
Le réseau d'interconnexion joue donc un rôle important, l'idéal étant, à priori, que chaque processeur soit cablé à tous les autres (réseau "crossbar"). Cependant, le nombre de connexions nécessaires augmentant très vite avec le nombre de processeurs (de l'ordre du carré du nombre de processeurs), la fabrication d'un tel réseau pose rapidement des problèmes pratiques.
Les architectures de type anneau, hypercube (Intel, FPS), grille (réseaux de Transputers), présentent un bon compromis nombre de connexions-temps d'accès entre deux processeurs.

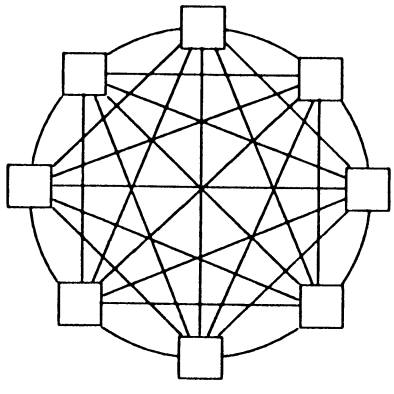


Figure II.3 : Réseau "crossbar"

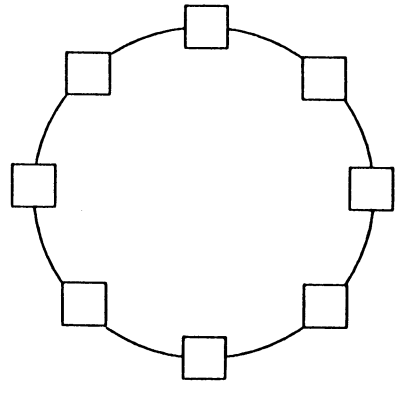


Figure II.4 : Anneau

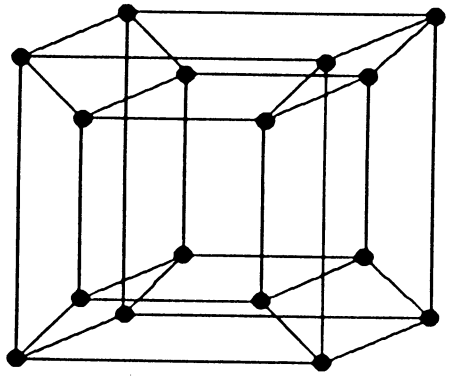


Figure II.5 : Hypercube

En résumé, les deux caractéristiques principales qui différencient les différentes machines MIMD sont:

- l'organisation de la mémoire :
  - mémoire partagée: une mémoire commune à tous les processeurs
  - mémoire distribuée: chaque processeur a sa propre mémoire.
- leur réseau de connexion (topologie). Dans les architectures modernes reconfigurables, ce réseau peut être modifié dynamiquement en cours d'algorithme (Supernode).

La seconde classe de machines parallèles est la classe des machines SIMD (Single Instruction Multiple Data). Ces machines sont par essence synchrones, elles possèdent un seul organe de contrôle qui communique à tous les processeurs une même instruction. L'organe de contrôle donne la cadence.
Dans cette catégorie, on inclue également les machines dites cellulaires possédant un très grand nombre de cellules dont la puissance de calcul est assez faible. Chaque cellule effectue une tâche élémentaire. Ce sont des machines dites à parallélisme massif, leur puissance vient de ce qu'une instruction peut être répétée des milliers de fois.
La Connection Machine, par exemple, qui possède 2^{16} processeurs est capable d'effectuer 65536 tâches élémentaires identiques mais sur des données différentes en parallèle.

Un dernier facteur qu'il faut prendre en compte dans la classification de ces machines est le temps de communication entre deux processeurs voisins. Outre l'influence de l'architecture du réseau de connexion, le temps de communication est lié au temps de transfert d'un processeur à son voisin. Ce temps dépend, en général, du temps d'initialisation de la connexion et du temps de transfert d'une unité élémentaire. Ces valeurs peuvent varier considérablement entre les différentes machines. Il existe également des machines avec "routeur", sans initialisation des liens de communication.

Nous donnerons, dans ce chapitre, quelques ordonnancements possibles pour le calcul de la forme normale d'Hermite dans le cadre d'une machine à

mémoire partagée. Nous présenterons ensuite des implantations parallèles pour une machine à mémoire distribuée de type hypercube.

II.2. Ordonnancements sur une machine à mémoire partagée

De nombreux travaux ont été effectués pour les problèmes d'algèbre linéaire numérique par [SAM78], [DAO89], [COR87],... Nous donnerons ici, à titre indicatif et bien qu'ils ne soient pas démontrés optimaux, quelques ordonnancements possibles, inspirés des algorithmes de Sameh et Kuck et des algorithmes gloutons pour l'algorithme modulo d. Ces algorithmes numériques sont destinés à des calculs de factorisation orthogonale de matrice (Householder, Givens). Or l'algorithme modulo d présente des similarités avec l'algorithme de Givens au niveau de la formulation générale de l'algorithme. La particularité du calcul de la forme d'Hermite, qui manipule de "vrais" entiers en précision infinie, apporte en outre une contrainte fondamentale, qu'on ne peut ignorer pour dégager un algorithme parallèle efficace.

Les algorithmes de calcul formel, qui présentent un grossissement des coefficients intermédiaires important, sont difficilement décomposables en tâches de coûts comparables.

L'algorithme modulo d, dans le cas entier, présente un avantage considérable: toutes les opérations étant effectuées modulo le déterminant, on peut raisonnablement considérer que les opérations de base du type $\mathrm{mod}(\alpha\, a + \beta\, b, d)$ ou $\mathrm{mod}(b - \alpha\, a, d)$ sont du même ordre de complexité. On peut alors se baser sur des hypothèses du même type que dans le cas des algorithmes d'algèbre linéaire.
Cependant, ce genre d'étude semble plus difficilement réalisable pour les autres algorithmes que nous avons vu, la taille des coefficients peut varier énormément et par suite les coûts des opérations ne restent pas constants.

II.2.1 Tâches et systèmes de tâches

Une des approches classiques pour paralléliser un algorithme consiste à dégager de l'algorithme séquentiel des sous-tâches indépendantes et de répartir ces sous-tâches entre les différents processeurs.

Rappelons tout d'abord les notions de tâche et de système de tâches présentées par S.P.Kumar [KUM82], E.G.Coffman et P.J.Denning [COF73], M.Cosnard et Y.Robert [COR87]. Cette notion permet de formaliser l'étude de la parallélisation d'un algorithme.

Une *tâche* est une unité indivisible de traitement caractérisée par ses entrées, ses sorties et son temps d'exécution. On peut donc la représenter par un

triplet (E_T, S_T, t_T) pour désigner respectivement l'ensemble des entrées, l'ensemble des sorties et le temps d'exécution de la tâche T.
Cette notion est très parlante qualitativement, même si elle n'est pas vraiment définie formellement.

On définit une *relation de précédence*: T << T' signifie que la tâche T doit être terminée avant que la tâche T' ne commence.

Définition (R.E.Lord, J.S.Kowalik, S.P.Kumar [LKK83]):
Un ensemble de tâches $J=\{T_1, T_2, \ldots, T_n\}$ associé à une relation d'ordre partiel << constitue *un système de tâches* et est noté C=(J,<<).

Définition:
un système de tâches $C = (\{T_1, T_2, \ldots, T_n\}, <<)$ est un *sytème de précédence* si << est une relation de précédence et si, pour tout i et k, une et une seule des conditions suivantes est vérifiée:

1) $T_i << T_k$
2) $T_k << T_i$
3) T_i et T_k sont indépendantes (c'est à dire qu'elles ne modifient aucune variable commune)

Définition:
Le graphe de précédence (ou graphe des tâches) associé à un système de précédence $C = (\{T_1, T_2, \ldots, T_n\}, <<)$ est défini de la manière suivante:

1)L'ensemble des sommets du graphe est l'ensemble des tâches de C

2)T_i et T_k sont reliées par une arête si et seulement si T_i et T_k sont consécutives ($T_i << T_k$, et il n'existe aucune autre tâche T_j telle que $T_i << T_j << T_k$).

II.2.2 Ordonnancements pour l'algorithme modulo d

Après avoir rappelé l'algorithme séquentiel, nous en extrairons des tâches élémentaires correspondant à une certaine granularité. Les contraintes liés à l'algorithme seront alors dégagées. Un graphe de précedence sera donné et des ordonnancements seront proposés.

Algorithmes séquentiels

Nous rappelons le schéma de l'algorithme modulo d présenté dans la première partie de cette thèse. Nous verrons ensuite que la stratégie d'élimination proposée n'est pas unique.

Algorithme modulo d:

début
pour k:=1 **à** n **faire**

{ Triangularisation }

pour j:=k+1 **à** n+k **faire**
pour i:=k+1 **à** n **faire**

$$\begin{pmatrix} a_{ik} \\ a_{ij} \end{pmatrix} := \begin{pmatrix} (p\ a_{ik} + q\ a_{ij}) \bmod d \\ (-\ a_{kj}\ a_{ik} + a_{kk}\ a_{ij}) \bmod d \end{pmatrix}$$

où $p\ a_{kk} + q\ a_{kj} = pgcd\ (a_{kk}\ ,\ a_{kj}) > 0$

$a_{kk} := pgcd(a_{kk}\ ,\ a_{kj})$
$a_{kj} := 0$

{ Réduction }

pour j:=1 **à** k-1 **faire**
quotient := div $(a_{kj}\ ,\ a_{kk})$
$a_{kj} := a_{kj}$ - quotient.a_{kk}
pour i:=k+1 **à** n **faire**
$a_{ij} :=$ mod$(a_{ij}$ - quotient.$a_{ik}\ ,\ d)$

fin

Dans cet algorithme, les éliminations sont toujours effectuées par rapport à l'élément diagonal. En réalité, pour éliminer un élément, plusieurs transformations sont possibles et, par conséquent, plusieurs algorithmes séquentiels sont envisageables. L'ordre d'éliminations des éléments peut également être modifié, la complexité théorique restent la même.

Tâches élémentaires

Y. Saad a montré [SAA86] que compte tenu du modèle de communication la parallélisation ne devenait intéressante qu'à partir d'une granularité en O(n). Nous nous appuyerons donc sur cette remarque pour le choix des tâches élémentaires considérées.

Nous considérerons deux tâches élémentaires dont se compose l'algorithme modulo d, correspondant respectivement à l'élimination d'un coefficient et à la réduction d'un élément.

N'ayant pas a priori définit le processus d'éliminations, l'élimination d'un élément sera définit par une fonction de trois paramètres. Notons $\tau(h,j,k)$ la

tâche consistant à éliminer le coefficient $a_{k,j}$ à l'aide de la colonne h. L'algorithme correspondant à cette tâche élémentaire est :

τ (h,j,k) ==
(s,t,g) := bezout (a_{kh} , a_{kj})
pour i:=k **à** n **faire**

$$\begin{pmatrix} a_{ih} \\ a_{ij} \end{pmatrix} := \begin{pmatrix} \mathrm{mod}(s*a_{ih} + t*a_{ij} , d) \\ \mathrm{mod}\left(\frac{-a_{kj}*a_{ih} + a_{kh}*a_{ij}}{g} , d\right) \end{pmatrix}$$

La tâche élémentaire R(j,k) permet de réduire le j^{ième} élément de la ligne k.

L'algorithme est le suivant:
R(j,k) ==
q := div(a_{kj}, a_{kk})
pour i:=k **à** n **faire**
a_{ij} := mod(a_{ij} - q*a_{ik} , d)

Reprenons le schéma de l'algorithme séquentiel dans lequel on change l'ordre d'élimination (on élimine de droite à gauche, l'algorithme reste valide et la complexité n'est pas modifiée) pour une présentation plus claire de la suite:

algorithme modulo d:
pour k:=1 **à** n **faire**
pour j:=n+k **à** k+1 **faire**
τ (k,j,k)
pour j:=1 **à** k-1 **faire**
R(j,k)

Posons $E_k^j = \tau$ (h,j,k) et R_k^z = R(z,k).

On a donc un ensemble de tâches :

$$J=\{E_k^j , j>k\} \cup \{R_k^z , z<k\}$$

et en reprenant les notations on a :

E ($E_k^j = \tau$ **(h,j,k)**) = {A(i,h)/ k≤i≤n} U {A(i,j)/ k≤i≤n}
E (R_k^z = **R(z,k)**) = {A(i,z)/ k≤i≤n} U {A(i,k)/ k≤i≤n}
S ($E_k^j = \tau$ **(h,j,k)**) = {A(i,h),A(i,j)/ k≤i≤n}
S (R_k^z = **R(z,k)**) = {A(i,z)/ k≤i≤n}

Signalons tout d'abord que τ **(h,j,k)** modifie les colonnes h et j. Il y a donc, pour k fixé, au plus n/2 transformations τ **(h,j,k)** disjointes.

Nous nous intéressons ici à un parallélisme à gros grain et considérons les tâches élémentaires:

E_k^j (j>k) : élimination sur la ligne k de l'élément en colonne j (correspond à une tâche τ **(h,j,k)**, h n'étant pas encore défini).

R_k^z (z<k) : réduction sur la ligne k de l'élément en colonne z.

Une tâche élémentaire consiste donc en une opération sur une ou deux colonnes.

Les tâches étant définies, une étude sur les contraintes de précédence imposées par l'algorithme doit être effectuée.

contrainte 1: la réduction des éléments à gauche de la diagonale sur la ligne k ne peut être effectuée qu'une fois le $k^{ième}$ élément diagonal de la forme d'Hermite connu, donc une fois les E_k^j effectués. D'où :

$$\boxed{E_k^j << R_k^z \text{ , pour tout } j>k \text{ et } z<k.}$$

contrainte 2: un élément annulé (ou réduit) précédemment ne doit pas être détruit par la suite, d'où :

$$\boxed{E_k^j << E_{k+1}^j \quad \text{et} \quad R_k^z << R_{k+1}^z}$$

contrainte 3: les opérations modulo d lors du traitement de la $k^{ième}$ ligne, correspondent à des transformations par rapport aux colonnes n+k+1, ..., 2n (à condition qu'elles n'aient pas encore été modifiées). Ce qui, a priori, implique:

$$\begin{cases} E_k^j << E_{k+i}^{n+k+i} \\ R_k^z << E_{k+i}^{n+k+i} \end{cases} \quad \text{pour } i = 1...(n-k)$$

On a le résultat général suivant:

Théorème:
Si on suppose que la tâche τ **(h,j,k)** pour j > k fait intervenir une colonne h et une colonne j possédant k-1 zéros en lignes 1 à k-1, alors les opérations peuvent être effectuées modulo d, quelque soit le déroulement de l'algorithme.

Preuve:
Rappelons que la transformation qui permet de faire les réductions modulo d correspond à une transformation par rapport à une colonne n+k , mais ne modifiant pas la colonne n+k.

Si $E_{k+j}^{n+k+j} = \tau$ **(h,j,k)**, cela sous-entend que $E_{k+j}^{h} << E_{k+j}^{n+k+j}$. La transformation effectuée fait donc intervenir une colonne h dont on a déjà annulé les éléments en ligne 1 à k+j-1; elle ne peut donc affecter une tâche E_k^t qui lui succède (si $E_k^t = \tau$ **(s,t,k)** alors s≠h). Donc, quel que soit l'ordre d'exécution des deux tâches, le résultat est le même.

Par conséquent, on peut effectuer la tâche E_k^t comme si la tâche E_{k+j}^{n+k+j} n'avait pas été effectuée et faire les opérations modulo d. De même pour les tâches R_k^t .
La contrainte 3 ne sera donc pas prise en compte.

Remarque: R_k^1, R_k^2, ..., et R_k^{k-1} peuvent être effectuées en parallèle.

Graphe de précédence et ordonnancements

Nous essayons ici de donner des ordonnancements dans le cadre d'une architecture à p processeurs à mémoire partagée. Nous nous appuyerons sur les résultats de M. Cosnard et M. Daoudi pour la méthode de Givens [DAO89]: l'algorithme de Givens présente en effet une analogie avec l'algorithme mod d au niveau de la précédence des tâches: une rotation de Givens modifie deux lignes.

Nous supposerons qu'un processeur est capable de:

- lire une colonne en mémoire
- effectuer une transformation τ ou **R**
- d'écrire en mémoire les colonnes modifiées

Deux colonnes sont modifiées par une transformation de type τ : deux processeurs ne pourront donc travailler simultanément sur une même colonne. La matrice contient 2n colonnes, le nombre de processeurs maximum utilisables est donc n pour les éliminations.

On supposera de plus que le temps d'une transformation est constant, indépendant de la taille des colonnes (en réalité il varie au cours de l'agorithme, la dimension des colonnes variant de n à 1). On mesurera donc le temps en nombre de transformations.

Le premier algorithme suit une stratégie d'élimination proposée par Sameh et Kuck pour l'algorithme de Givens.

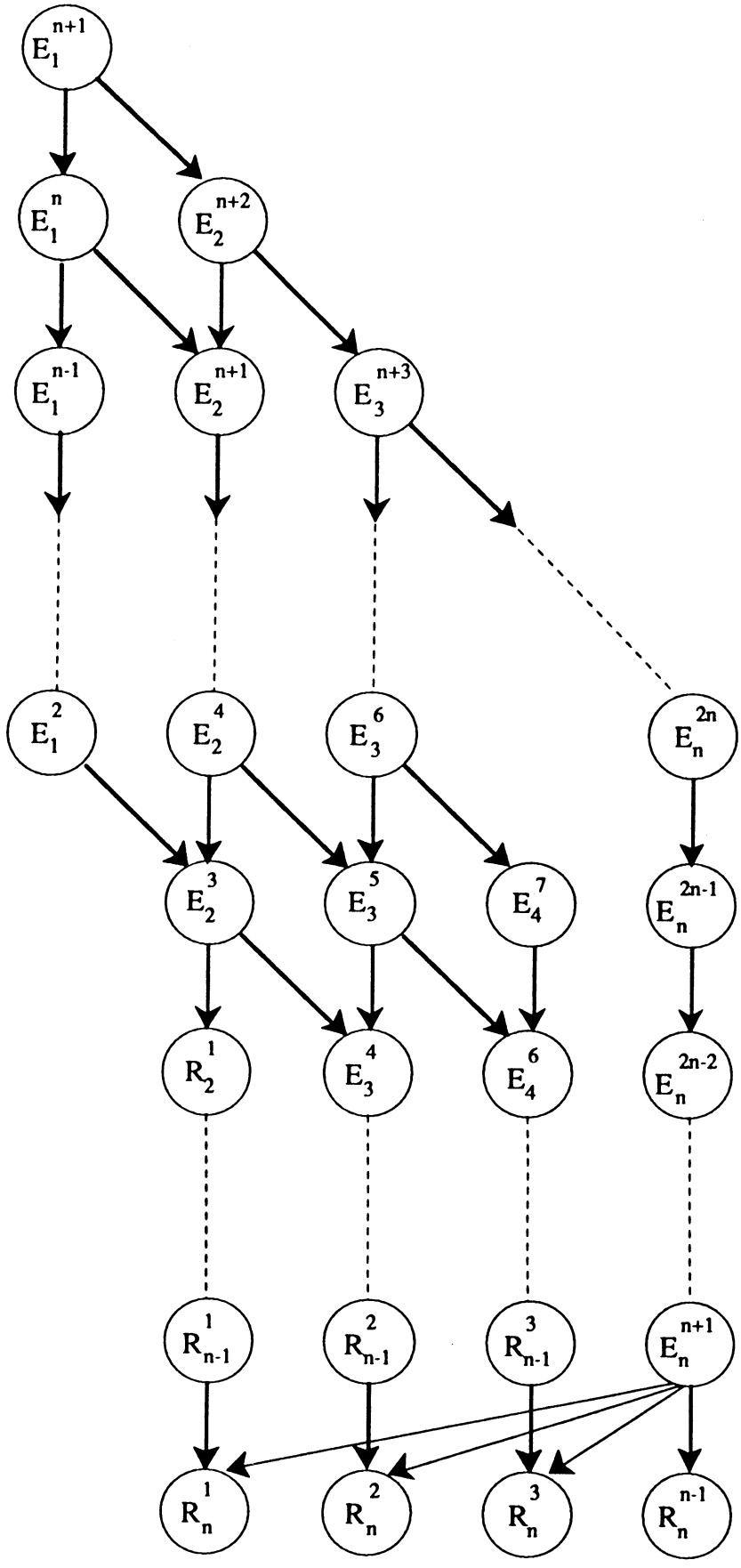


Figure II.6 : Graphe des Tâches

Pour éliminer l'élément en position (k,j) on effectue $\tau(j-1,j,k)$. L'élément en position (k,j) peut être éliminé dés que l'élément en position (k-1,j-1) a été annulé.

On pose $E_k^j = \tau(j-1,j,k)$, d'où : $E_k^j << E_k^{j-1}$.

Et on obtient un système parallèle de tâche C=(J,<<), où :

$$<< \equiv [(E_k^j, E_{k+1}^j) \cup (E_k^j, E_k^{j-1}) \cup (R_k^z, R_{k+1}^z) \cup (E_k^j, R_k^z) \; / 1 \le k \le n-1,\ k < j \le n+k,\ 1 \le z < k]$$

Toutes les contraintes de précédence imposées par l'algorithme sont ainsi définies.

On remarque que, pour c donné, $1-n \le c \le n-1$, les tâches :

$$E_a^b \text{ avec } \begin{cases} 1 \le a \le n \\ a+1 \le b \le a+n \\ b-2a = c \end{cases} \text{ sont indépendantes.}$$

On obtient donc le graphe des tâches représenté sur la figure de la page précédente. Les flêches représentent la relation de précédence.

Posons $\delta = n \log_q B + n/2 \log_q n$, où B est une borne sur la valeur absolue des coefficients, et q la base dans laquelle on travaille.

Le coût arithmétique des tâches est (pour j>k et l<k) alors de l'ordre de (cf Partie I) :

$$\boxed{\begin{array}{c} C(E_k^j) = ((n-k)(4T_{mul}(\delta) + 2T_{div}(\delta)) + T_{pgcd}(\delta)) \\ C(R_k^z) = (n-k+1)\,(T_{mul}(\delta) + T_{div}(\delta)) \end{array}}$$

Le coût du plus long chemin est :

$$\sum_{j=n+1}^{2} C(E_1^j) + \sum_{k=2}^{n} C(E_k^{k+1}) + C(R_n^{n-1})$$

$$= \sum_{j=2}^{n+1} ((n-1)(4\,T_{mul}(\delta) + 2\,T_{div}(\delta)) + \log(\delta) T_{mul}(\delta))$$

$$+ \sum_{j=2}^{n} ((n-j)(4\,T_{mul}(\delta) + 2\,T_{div}(\delta)) + \log(\delta) T_{mul}(\delta))$$

$$+ T_{mul}(\delta) + T_{div}(\delta)$$

Finalement la profondeur de l'algorithme est:

$$((n-1)(3n/2 -1) + 1)\, T_{mul}(\delta) + (2n -1)T_{pgcd}(\delta) + (2(n-1)(3n/2-1)+1)T_{div}(\delta)$$

Elle correspond au chemin E_1^{n+1} , E_1^{n},..., E_1^{2}, E_2^{3}, E_3^{4}, ..., E_n^{n+1}, R_n^{n-1} .
En suivant une stratégie inspirée de celle de Sameh et Kuck, on obtient pour n=10 le schéma suivant:

*	10	9	8	7	6	5	4	3	2	1									
12	*	11	10	9	8	7	6	5	4	3	2								
13	13	*	12	11	10	9	8	7	6	5	4	3							
14	14	14	*	13	12	11	10	9	8	7	6	5	4						
15	15	15	15	*	14	13	12	11	10	9	8	7	6	5					
16	16	16	16	16	*	15	14	13	12	11	10	9	8	7	6				
17	17	17	17	17	17	*	16	15	14	13	12	11	10	9	8	7			
18	18	18	18	18	18	18	*	17	16	15	14	13	12	11	10	9	8		
19	19	19	19	19	19	19	19	*	18	17	16	15	14	13	12	11	10	9	
20	20	20	20	20	20	20	20	20	*	19	18	17	16	15	14	13	12	11	10

Figure II.7 : Stratégie de Sameh et Kuck (n = 10)
Les nombres correspondent aux étapes successives où les éléments sont modifiés.

Théorème 1: le nombre total de pas pour calculer la forme d'Hermite d'une matrice nxn par l'algorithme modulo d et en suivant la stratégie de Sameh et Kuck sur n processeurs est 2n.

Preuve:
Montrons dans un premier temps que, sans restriction sur le nombre de processeurs, le nombre de pas nécessaire est 2n.

• Quand l'élimination progresse de droite à gauche, dans la première ligne, le nombre de coefficients annulés augmente de 1 tous les pas.
Le nombre de pas nécessaire pour annuler les coefficients de la première ligne, jusqu'en position (1, 2), est donc égal à n.
Dés que le coefficient (i, j), j>i, a été éliminé, on peut annuler le coefficient (i+1, j+1). Donc, si le coefficient (i, i+1) est éliminé à l'étape k, le coefficient (i+1, i+2) sera éliminé à l'étape k+1. Le coefficient (2, 3) sera éliminé à l'étape n+1 et, par induction, le coefficient (n, n+1) sera éliminé à l'étape (2n -1).

• La réduction du coefficient (2, 1) peut être effectuée au pas n+2 (élimination terminée sur la seconde ligne).

Dés que les éliminations sont terminées sur une ligne, toutes les éliminations sur cette ligne peuvent être effectuées simultanément.
Les éliminations de la dernière ligne se terminant à l'étape (2n -1), il faut 2n étapes pour terminer l'algorithme.

Montrons que n processeurs suffisent.

Lemme: le nombre maximum d'opérations simultanées est égal à n et ce nombre est atteint au pas n (il correspond à l'annulation des éléments en positions (1,2), (2,4), (3,6), ..., (n,2n)).

Preuve:
Soit R_k le nombre de réductions effectuées à l'étape k et E_k le nombre d'éliminations effectuées à cette même étape.

On a $\begin{cases} R_k = 0 \text{ si } k < n+2 \\ R_k = k - n - 1 \text{ si } k \geq n+2 \end{cases}$ et $\begin{cases} E_k = k \text{ si } k \leq n \\ E_k = 2n - k \text{ si } k > n \end{cases}$

D'où

$$R_k + E_k = \begin{cases} k \text{ si } k \leq n \\ 2n - k \text{ si } k = n+1 \\ n - 1 \text{ si } k > n+1 \end{cases}$$

On en déduit que le nombre maximum de tâches effectuées simultanément est n, à l'étape n. CQFD.
Ce nombre correspond au diamètre du graphe.
L'exécution des E_k^j précède celle des R_k^z. Essayons une stratégie où l'élimination de la ligne k est effectuée en priorité. On s'intéressera donc ici à une stratégie de type glouton: à chaque pas on annule le maximum d'éléments possible par ordre croissant des lignes, et de droite à gauche.

*	4	3	2	2	2	1	1	1	1	1								
7	*	6	5	4	4	3	3	3	2	2	2							
9	9	*	8	7	6	6	5	5	4	4	3	3						
11	11	11	*	10	9	8	7	7	6	6	5	5	4					
13	13	13	13	*	12	11	10	9	8	8	7	7	6	5				
15	15	15	15	15	*	14	13	12	11	10	9	9	8	7	6			
17	17	17	17	17	17	*	16	15	14	13	12	11	10	9	8	7		
18	18	18	18	18	18	18	*	17	16	15	14	13	12	11	10	9	8	
19	19	19	19	19	19	19	19	*	18	17	16	15	14	13	12	11	10	9
20	20	20	20	20	20	20	20	20	*	19	18	17	16	15	14	13	12	11	10

Figure II.8 : Un algorithme glouton

Nous n'avons pas réussi à calculer le nombre d'étapes nécessaires à l'exécution de l'algorithme, mais nous avons simulé l'algorithme pour différentes valeurs de n et nous avons obtenu les résultats suivants:

dimension de la matrice	nb. tâches maximum	pas	nb. de pas total
190	231	189	380
150	179	150	300
100	120	99	200

Pour n = 150, l'algorithme s'effectue en 2n pas mais le nombre de tâches maximum est supérieur à n. Il s'avère donc que la première stratégie est meilleure que l'algorithme glouton.

II.3. Machine à mémoire distribuée

II.3.1. Le modèle

♦ Modèle d'architecture

Nous nous intéressons à une machine à mémoire distribuée, dans laquelle les processeurs sont connectés de voisin à voisin suivant une topologie hypercube. Ils communiquent par envoi de messages.
Le protocole de communications est basé sur le rendez-vous entre deux processeurs.
On suppose que communications et calculs ne se font pas en parallèle.

♦ Modèle de complexité

Coût des communications: le coût des communications est une fonction linéaire de la longueur des messages. Le temps nécessaire pour communiquer un message de longueur nL est [SAA85]

$$\beta + n\,\tau$$

où β est un temps d'initialisation du canal, et τ le temps de transfert d'un message de taille L.

Asymptotiquement, β pourra être négligé devant $n\,\tau$. Cependant, il est important de tenir compte du rapport β/τ, spécifique à chaque machine, pour définir la granularité des messages envoyés [TOU89].

♦ Modèle d'algorithmes

Le but est de trouver, pour un algorithme donné, un graphe des tâches qui minimise les calculs, les communications et les temps de synchronisation (latence). On se placera ici sous certaines hypothèses appropriées aux architectures sur lesquelles les communications se font par échanges de messages.

On considérera un architecture à p processeurs, et pour simplifier nous supposons que p divise n. On suppose de plus que les processeurs sont numérotés de 0 à p-1. On définit tout d'abord une fonction **alloc**

$$\textbf{alloc} : \{ 1, 2, \ldots, n \} \longrightarrow \{ 0, 1, \ldots, p-1 \}$$

qui associe à une colonne (ou une ligne) i un processeur alloc(i). on supposera qu'à chaque processeur sont alloués n/p colonnes (ou lignes). Différents schémas de répartitions ont été étudiés pour des algorithmes d'algèbre linéaire avec le même type de topologie. Nous avons choisi ici une répartition équitable des données entre les processeurs.

Pour réduire les coûts de communications et de gestion mémoire nous reprenons les hypothèses formulées par Gerasoulis et Nelken [GER88]:
1/ Principe de localité : un processeur ne peut modifier que les données qui sont stockées dans ce processeur.
2/ Hypothèse de complétude : le programme de chacun des processeurs est de la forme:

```
début
    pour k=1 à n faire
        Echanges de données (ligne ou colonne)
        Calculs: Elimination / Réduction
fin
```

Ces hypothèses imposent certaines restrictions sur l'ordonnancement des tâches.

II.3.2. Matrices à coefficients entiers

Nous allons maintenant étudier la parallélisation de la forme normale d'Hermite sur une architecture de type hypercube, à $p=2^n$ processeurs.
L' hypercube est un ordinateur parallèle dont les processeurs, et les connections de voisins à voisins reliant ces derniers, forment un n-cube binaire.

Définition: un n-cube binaire est un graphe composé de 2^n sommets numérotés de 0 à 2^n-1 tel que deux sommets sont reliés entre eux si et seulement si leur représentation binaire diffère exactement d'un bit.

Une des caractéristiques d'un n-cube est que l'on peut lui inscrire de nombreuses autres topologies, notamment la structure d'anneau qui peut être définie à partir des codes Gray [SAA85], ou la structure d'arbre. Ces propriétés facilitent l'organisation et l'échange de données entre processeurs.

On supposera par la suite que les processeurs sont numérotés P_i, pour i = 0 à p-1.

Nous considérerons toujours l'algorithme modulo d et nous en présenterons deux parallélisations; l'une correspondant à une répartition par colonnes de la matrice (algorithme HC), l'autre à une répartition par lignes (algorithme HL).

II.3.2.1. Répartition par colonnes

II.3.2.1.1 Description

La matrice est répartie une fois pour toute dans les processeurs, en début d'algorithme. Plaçons nous dans un premier temps sous l'hypothése d'une répartition équilibrée de la matrice par colonnes.

Séparons la tâche $E_k^j = \tau(h,j,k)$ en deux sous tâches:

Soit e_k^j, la tâche consistant à transformer la colonne j pour faire apparaître un zéro en ligne k.

Soit f_k^j , la tâche consistant à transformer la colonne h pour faire apparaître le pgcd de a_{kh} et a_{kj} en ligne k.
Par application du principe de localité, toutes les tâches de l'ensemble $\{e_1^j, e_2^j, e_3^j, \ldots, e_n^j\}$ sont effectuées dans le processeur alloc(j), toutes les tâches de l'ensemble $\{f_1^j, f_2^j, f_3^j, \ldots, f_n^j\}$ sont effectuées dans le processeur alloc(h).

♦ Le principe de localité réduit le temps de calcul :

Supposons que l'on introduise un zéro en colonne j grâce à la colonne h, en ligne k. Cette transformation peut être soit effectuée dans le processeur alloc(j), ou dans le processeur alloc(h), soit répartie entre les deux processeurs.
Si on l'effectue dans le processeur alloc(j) alors :
alloc(h) envoie sa colonne à alloc(j). alloc(j) effectue E_k^j, puis renvoie la colonne h modifiée à alloc(h).

Si on applique le principe de localité:

alloc(h) et alloc(j) s'échangent leur colonnes. En parallèle:
- alloc(j) applique l'algorithme d'Euclide étendu sur les coefficients de tête, puis effectue e_k^j
- alloc(h) applique l'algorithme d'Euclide étendu sur les coefficients de tête, puis effectue f_k^j

Les tâches e_k^j et f_k^j étant du même ordre de complexité, la tâche E_k^j, dans la seconde méthode, a été décomposée en deux sous-tâches de même taille, effectuées en parallèle (seuls les coefficients de bezout sont calculés dans les deux processeurs). La seconde solution s'avère donc moins coûteuse en temps de calcul.

♦ Le principe de localité réduit le temps de communication :
Considérons la tâche R_k^h. Cette tâche peut être effectuée soit dans le processeur alloc(k), soit dans le processeur alloc(h).

Dans le premier cas, alloc(h) envoie la colonne h à alloc(k), celui-ci effectue la réduction et renvoie la colonne h modifiée à alloc(h).

Dans l'autre cas, alloc(k) envoie la colonne k à alloc(h), et celui-ci effectue la réduction. Il y a donc deux fois moins de communications dans le cas où l'on applique le principe de localité.

♦ Le principe de complétude réduit les temps de communication, ou l'occupation mémoire :
Dés qu'un processeur reçoit une colonne, il effectue toutes les transformations faisant intervenir cette colonne. Ceci évite d'avoir à communiquer deux fois la même colonne, ou de stocker une colonne pour une opération ultérieure.

Considérons l'exemple suivant:

processeurs	P_0	P_1	P_2	P_3
colonnes	1, 5, 9, 13	2, 6, 10, 14	3, 7, 11,15	4, 8, 12, 16
étape 1	E_1^{13}	E_1^{14}	E_1^{15}	E_1^{16}
étape 2	E_1^{9}	E_1^{10}	E_1^{11}	E_1^{12}
étape 3	E_1^{5}	E_1^{6}	E_1^{7}	E_1^{8}
étape 4	f_1^2	e_1^2	f_1^4	e_1^4

Aprés l'étape 3, les processeurs P_0 et P_1 s'échangent les colonnes 1 et 2. Le processeur P_0 peut alors

- soit effectuer f_1^2

- soit effectuer E_2^{13}, puis effectuer f_1^2. Mais il faut alors garder la colonne reçue en mémoire.

II.3.2.1.2 Algorithme HC

La façon la plus naturelle de paralléliser l'algorithme mod d est de subdiviser la matrice A en p blocs de n/p colonnes, et de construire pour chaque bloc une sous matrice de (A D) formée des n/p colonnes du bloc et des n/p colonnes correspondantes de D. Une sous-matrice est allouée à chaque processeur, de telle façon que chaque colonne soit stockée dans un processeur et un seul.

Plusieurs méthodes de décomposition de la matrice sont étudiées pour la matrice de Gauss dans [SAA86]. Nous choisissons ici une décomposition qui permet d'équilibrer le travail des processeurs. On affecte au processeur P_i les colonnes $i + 1 + k\,p$, pour $k = 0$ à $((2n/p)-1)$ (figure II.9).

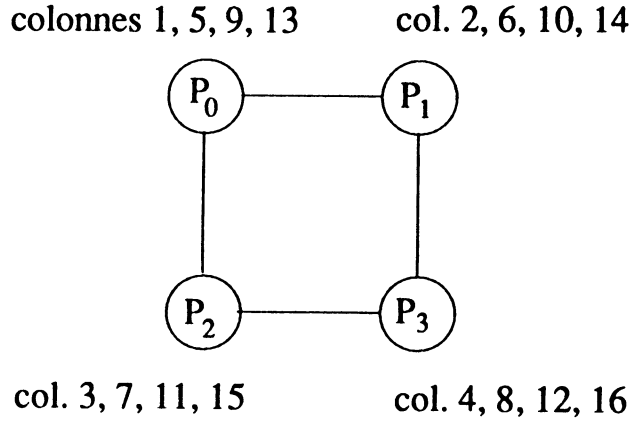


Figure II.9 : Répartition équilibrée des colonnes sur 8 processeurs, n=4.

Le principe de l'algorithme HC consiste à effectuer les éliminations et les réductions ligne par ligne. L'ordre d'élimination des éléments ou l'ordre de réduction pouvant être choisi arbitrairement, il sera ici fonction de la répartition des colonnes. La seule contrainte est que les réductions sur la ligne k ne se font qu'une fois les éliminations effectuées sur cette même ligne. Les éliminations se font, de gauche à droite et par rapport à la première colonne où les éliminations ne sont pas terminées, dans chacune des sous-matrices.
L'algorithme s'effectue en n étapes, chacune composée de deux phases d'élimination et de deux phases de réduction.

A l'étape k, l'élimination évolue en deux phases:

• Dans une première phase locale, chaque processeur utilise ses propres colonnes pour éliminer sur la ligne k.
A la fin de la première phase, dans chaque processeur il reste un élément à annuler, sauf dans le processeur qui contient la colonne k.

• Dans la deuxième phase, les processeurs communiquent un vecteur pour annuler les (p-1) éléments restants (un sur chaque processeur).
Sur l'hypercube, on peut éliminer les p-1 éléments restants en $\log_2 p$ communications.

Une fois les éliminations terminées, tous les processeurs doivent récupérer la colonne k pour effectuer les réductions. La réduction des éléments de la ligne k s'effectue donc elle aussi en deux phases:
• Dans une première phase de communication, le processeur qui possède la colonne k (appelé processeur racine) la diffuse à tous les autres.
• Dans une deuxième phase, locale, les processeurs effectuent les réductions en utilisant la colonne reçue.

Nous noterons $(\tilde{a}_{ij})_{1\le i\le n\ ,\ 1\le j\le 2n/p}$ les coefficients de la sous-matrice du processeur dans lequel on se place.

On définit les opérations suivantes:

♦ élimination dans la sous matrice :

$\tau(c,j,k)$ ==
(s,t,g):=bezout($\tilde{a}_{kc}$,$\tilde{a}_{kj}$)
pour h:=k à n faire

$$\begin{pmatrix}\tilde{a}_{hc}\\ \tilde{a}_{hj}\end{pmatrix} := \begin{pmatrix} \mathrm{mod}(s.\tilde{a}_{hc} + t.\tilde{a}_{hj}\ ,\ d) \\ \mathrm{mod}((-\tilde{a}_{kj}.\tilde{a}_{hc} + \tilde{a}_{kc}.\tilde{a}_{hj})/g\ ,\ d)\end{pmatrix}$$

♦ Fin de l'étape d'élimination :
On fait apparaître un pgcd:
$f_r(c,k)$ ==
(s,t,g):=bezout($\tilde{a}_{kc}$,r_k)
pour h:=k à n faire
$\tilde{a}_{hc}$:= mod(s.$\tilde{a}_{hc}$ + t.r_h, d)
On fait apparaître un zéro:
$e_r(c,k)$ ==
(s,t,g):=bezout(r_k,$\tilde{a}_{kj}$)
pour h:=k à n faire
$\tilde{a}_{hj}$:= mod((- $\tilde{a}_{kj}$.r_h + r_k.$\tilde{a}_{hj}$)/g , d)

♦ Réduction
Dans le processeur qui possède la colonne k:
R(c,j,k,q) ==
pour h:=k à n faire
$\tilde{a}_{hc}$:= mod($\tilde{a}_{hc}$ - q.$\tilde{a}_{hj}$, d)
Dans un processeur qui ne possède pas la colonne k:
$R_r(j,k,q)$ ==
pour h:=k à n faire
$\tilde{a}_{hj}$:= mod($\tilde{a}_{hj}$ - q.r_h , d)

La variable c indique la colonne à partir de laquelle se fait l'élimination dans le processeur P_i. Toutes les colonnes qui la précède, dans ce processeur, correspondent à des colonnes de la matrice globale sur lesquelles toutes les éliminations ont été effectuées.

Algorithme HC:

PAR pour tout processeur P_i **faire**

SEQ

c:=1
pour k=1 à n **faire**

{ élimination dans les colonnes > k de A }

pour j= (c+1) **à** (n/p+c) **faire**
τ(**c,j,k**)

{ fin de l'élimination sur la ligne k }

cf détails p. 110

{ diffusion de la colonne k }

si alloc(k)=i **alors**
diffuser la colonne k (colonne c de P_i)
sinon
recevoir colonne k dans le vecteur r

{ réduction des colonnes < k }

pour j=1 **à** (c-1) **faire**
si alloc(k)=i **alors**
q:= div($\tilde{a}_{kj}$,$\tilde{a}_{kc}$)
R(c,j,k,q)
sinon
q:= div($\tilde{a}_{kj}$,r_k)
R_r(j,k,q)

si alloc(k)=i **alors** c:=c+1

Le schéma qui suit décrit l'état de la sous matrice A_i dans le processeur P_i durant l'exécution de l'algorithme.

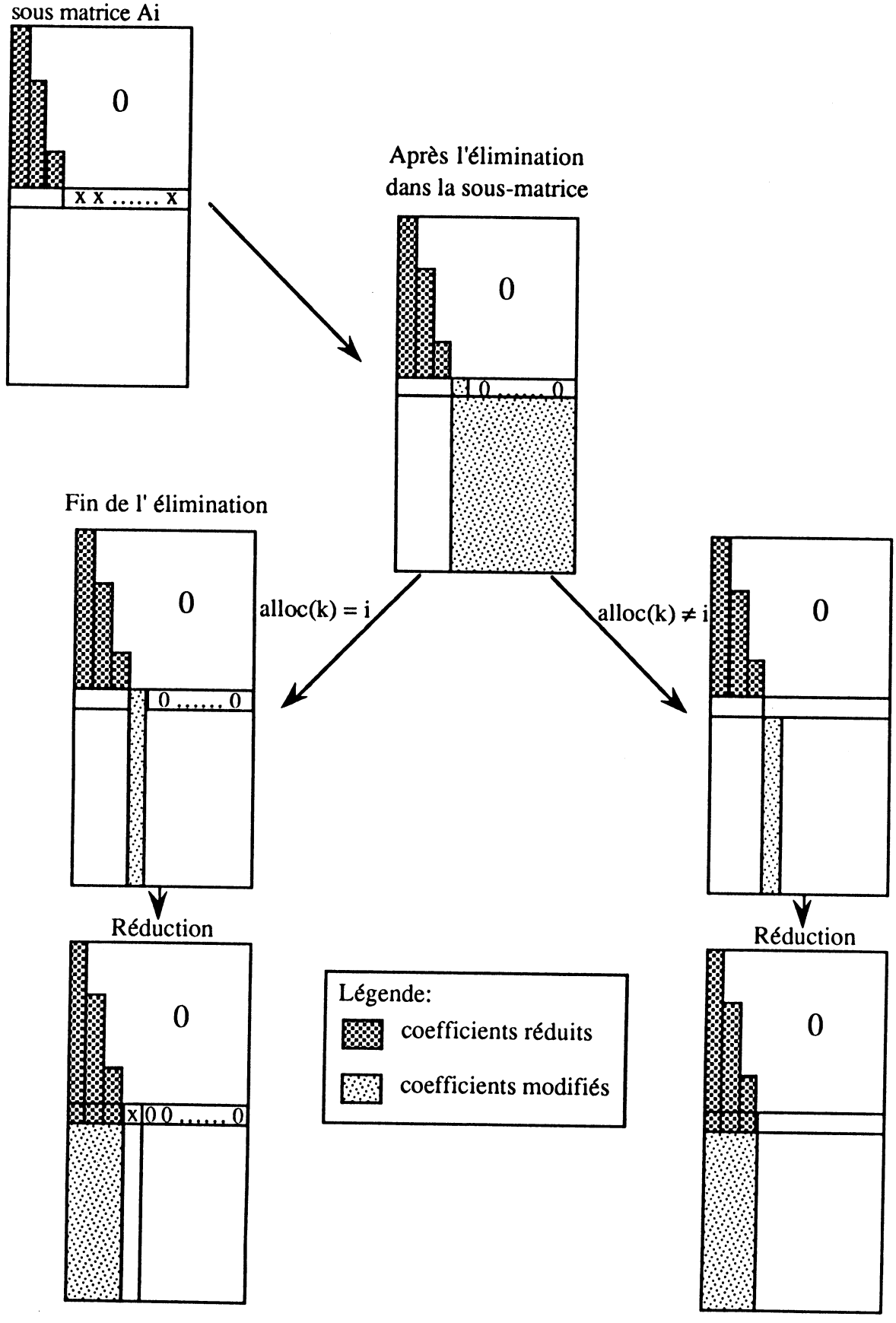


Figure II.10 : Etape k de l'algorithme HC dans le processeur P_i

Remarque:
La deuxième version de l'algorithme modulo d proposée par Domich, Kannan et Trotter [DOM87] peut être parallélisée de la même façon. En effet, aprés chaque élimination, tous les processeurs récupèrent la colonne k et en particulier le $k^{ième}$ coefficient diagonal. L'instruction à ajouter serait donc:

si k<n **alors si** alloc(k) = i **alors** d := div(d,$\tilde{a}_{kc}$)
sinon d := div(d , r_l)

A l'ètape k, le processeur $P_{alloc(k)}$ est dit processeur "racine" (sa colonne c est la colonne k de A) en raison du rôle qu'il joue lors de la fin de l'élimination et lors de la réduction.

Plusieurs méthodes peuvent être utilisées pour la diffusion d'un message dans l'hypercube. Nous employons ici le schéma classique de remontée dans un hypercube, vers le processeur racine, sur un arbre de hauteur $\log_2 p$ inscrit dans l'hypercube. De nombreux types d'arbres peuvent être utilisés. La diffusion d'un message de taille n peut cependant se faire en un temps minimum en utilisant $\log_2 p$ arbres et en découpant le message en plusieurs paquets qui sont chacun transmis sur un des arbres ("broadcast rotatif" proposé par Jhonsson et Ho [JHH89]).

Fin de l'élimination et processus de diffusion :

On définit la fonction premier_bit_a_1(i) qui renvoie le premier bit non nul du processeur i en partant des poids faibles, et on pose: premier_bit_a_1(0)=dim.
Notons P_r, le processeur racine et P_v la position de P_i dans un cube dont on aurait permuté les numéros de façon à ce que le processeur P_r devienne le processeur zéro.

$$P_v = \text{mod}((P_i - P_r) , p)$$

{ fin de l'élimination}
pour j = 0 **à** dim - 1
  si (premier_bit_a_1(P_v) > j) **alors**
    envoyer colonne c au processeur voisin
    recevoir dans r une colonne du processeur voisin
    effectuer $\mathbf{f_r}$ **(c,k)**

  sinon si (premier_bit_a_1(P_v) = j) **alors**
    recevoir dans r une colonne du processeur voisin
    envoyer colonne c au processeur voisin
    effectuer $\mathbf{e_r}$ **(c,k)**

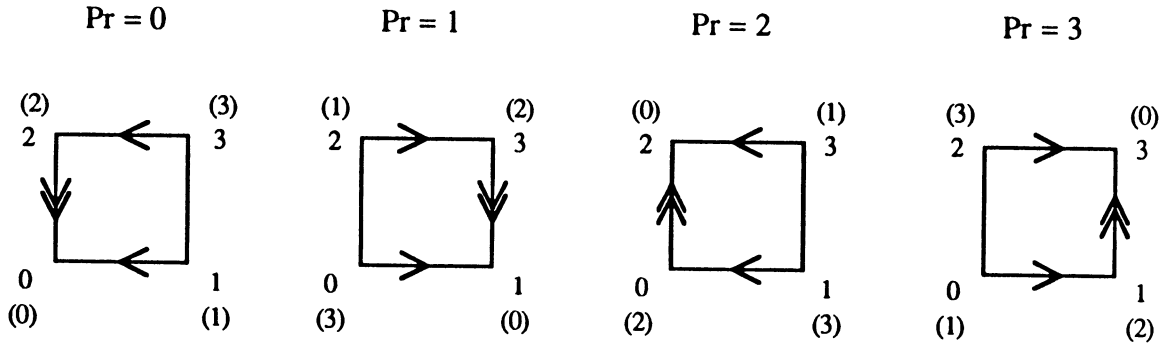
Nous donnons ci-dessous le schéma d'élimination sur 4 processeurs pour n = 8.

Le nombre r est en position (i,k) si a_{ki} est éliminé en utilisant les colonnes i et r, à la $k^{ième}$ étape de l'algorithme. On suppose que la stratégie de répartition est la suivante: le processeur proc possède les colonnes proc + 1, proc + p + 1, proc + 2p + 1, ..., proc + (2n/p - 1) p - 1. A l'étape k, le processeur racine est Pr = mod (k-1, p).

colonne	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
processeur	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
k = 1	*	1	1	3	1	2	3	4	1							
k = 2		*	4	2	2	2	3	4	5	2						
k = 3			*	3	3	5	3	4	5	6	3					
k = 4				*	6	4	4	4	5	6	7	4				
k = 5					*	5	5	7	5	6	7	8	5			
k = 6						*	8	6	6	6	7	8	9	6		
k = 7							*	7	7	9	7	8	9	10	7	
k = 8								*	10	8	8	8	9	10	11	12

Figure II.11 : schéma d'élimination pour p = 4 et n = 8.

Le processus de communication est alors donné par la figure ci-dessous, les communications se font en fait dans les deux sens, mais pour ne pas surcharger le dessin, nous n'avons indiqué que la seconde partie de la communication. Nous avons noté entre parenthèses la position virtuelle du processeur.



Première étape de communication

Deuxième étape de communication

Figure II.12 : schéma des communications p = 4.

Le processus de diffusion se fait en suivant le schéma de communication inverse (descente dans l'arbre en partant du processeur racine):

```
{ processus de diffusion }
pour j = dim-1 à 0
    si (premier_bit_a_1(Pv) > j) alors
        envoyer colonne c au processeur voisin

    sinon si (premier_bit_a_1(Pv) = j) alors
        recevoir dans r une colonne du processeur voisin
```

II.3.2.2. Répartition par lignes

II.3.2.2.1 Description

Notons tout d'abord que, à l'étape k de l'algorithme séquentiel, les transformations à effectuer pour éliminer et réduire les éléments sur la ligne k dépendent uniquement de la ligne k.

Le deuxième algorithme proposé, noté HL, consiste à subdiviser la matrice A en p blocs de n/p lignes. A chaque étape, le processeur qui possède la ligne k l'envoie à tous les autres processeurs. Chaque processeur peut alors calculer les transformations à effectuer sur les coefficients qu'il possède en mémoire. L'algorithme s'effectue donc en n étapes qui peuvent se décomposer en deux phases: une phase de communication et une phase de calcul, interne à chaque processeur.

• Durant la phase de communication, la ligne k est diffusée à partir du processeur racine dans tous les autres processeurs par le même processus que dans l'algorithme HC.

• La phase de calcul se décompose en une phase d'élimination et une phase de réduction.

Pour chaque élimination, chaque processeur:

- construit la transformation correspondante (à l'aide des coefficients de la ligne k dans le processeur racine, et à l'aide des coefficients de la ligne reçue pour les autres),

- l'applique sur les lignes de la sous matrice (ainsi que sur la ligne reçue pour les processeurs non racine, le dernier coefficient obtenu étant nécessaire pour effectuer les réductions).

De même pour chaque réduction, chaque processeur:

- construit la transformation correspondante (à l'aide des coefficients de la ligne k dans le processeur racine, et à l'aide des coefficients de la ligne reçue pour les autres),

- l'applique sur les lignes de la sous matrice (non compris la ligne reçue pour les processeurs non racine).

II.3.2.2.2. Algorithme HL

Nous noterons $(\tilde{a}_{ij})_{1\le i\le n/p\ ,\ 1\le j\le 2n}$ les coefficients de la sous-matrice du processeur dans lequel on se place.
On définit les opérations suivantes:

- τ (L,k,j):

  $(s,t,g):=\text{bezout}(\tilde{a}_{Lk},\tilde{a}_{Lj})$

  pour i:=L à n/p faire

  $$\begin{pmatrix}\tilde{a}_{ik}\\ \tilde{a}_{ij}\end{pmatrix}:=\begin{pmatrix}\text{mod}(s.\tilde{a}_{ik} + t.\tilde{a}_{ij}\ ,\ d)\\ \text{mod}((-\tilde{a}_{Lj}.\tilde{a}_{ik} + \tilde{a}_{Lk}.\tilde{a}_{ij})/g\ ,\ d)\end{pmatrix}$$

- τ_r (L,k,j):

  $(s,t,g):=\text{bezout}(r_k,r_j)$

  pour i=L à n/p faire

  $$\begin{pmatrix}\tilde{a}_{ik}\\ \tilde{a}_{ij}\end{pmatrix}:=\begin{pmatrix}\text{mod}(s.\tilde{a}_{ik} + t.\tilde{a}_{ij}\ ,\ d)\\ \text{mod}((-r_j.\tilde{a}_{ik} + r_k.\tilde{a}_{ij})/g\ ,\ d)\end{pmatrix}$$

  $r_k := g$

- R(L,j,k,q):

  pour i=L à n/p faire

  $\tilde{a}_{ij} := \text{mod}(\tilde{a}_{ij} - q.\tilde{a}_{ik}\ ,\ d)$

On construit alors l'algorithme HL:

Algorithme HL:

```
PAR pour tout processeur Pi faire
    SEQ
        L:=1
        pour k=1 à n faire

            { diffusion de la ligne k de A }
                si alloc(k)=i alors
                    diffuser la ligne k (ligne L de Pi)
                sinon
                    recevoir ligne k dans le vecteur r

            { élimination sur la ligne k }
                pour j=k+1 à (n+k) faire
                    si alloc(k)=i  alors   τ (L,k,j)
                                   sinon   τ r(L,k,j)

            { réduction des colonnes < k }
                pour j=1 à (k-1) faire
                    si alloc(k)=i alors
                        q:= div(ãLj,ãLk)
                    sinon
                        q:= div(rj,rk)
                    R(L,j,k,q)
            si alloc(k)=i alors L:=L+1
```

La ligne L représente à chaque étape la première ligne sur laquelle n'ont pas encore été effectuées les éliminations et les réductions dans la sous-matrice.

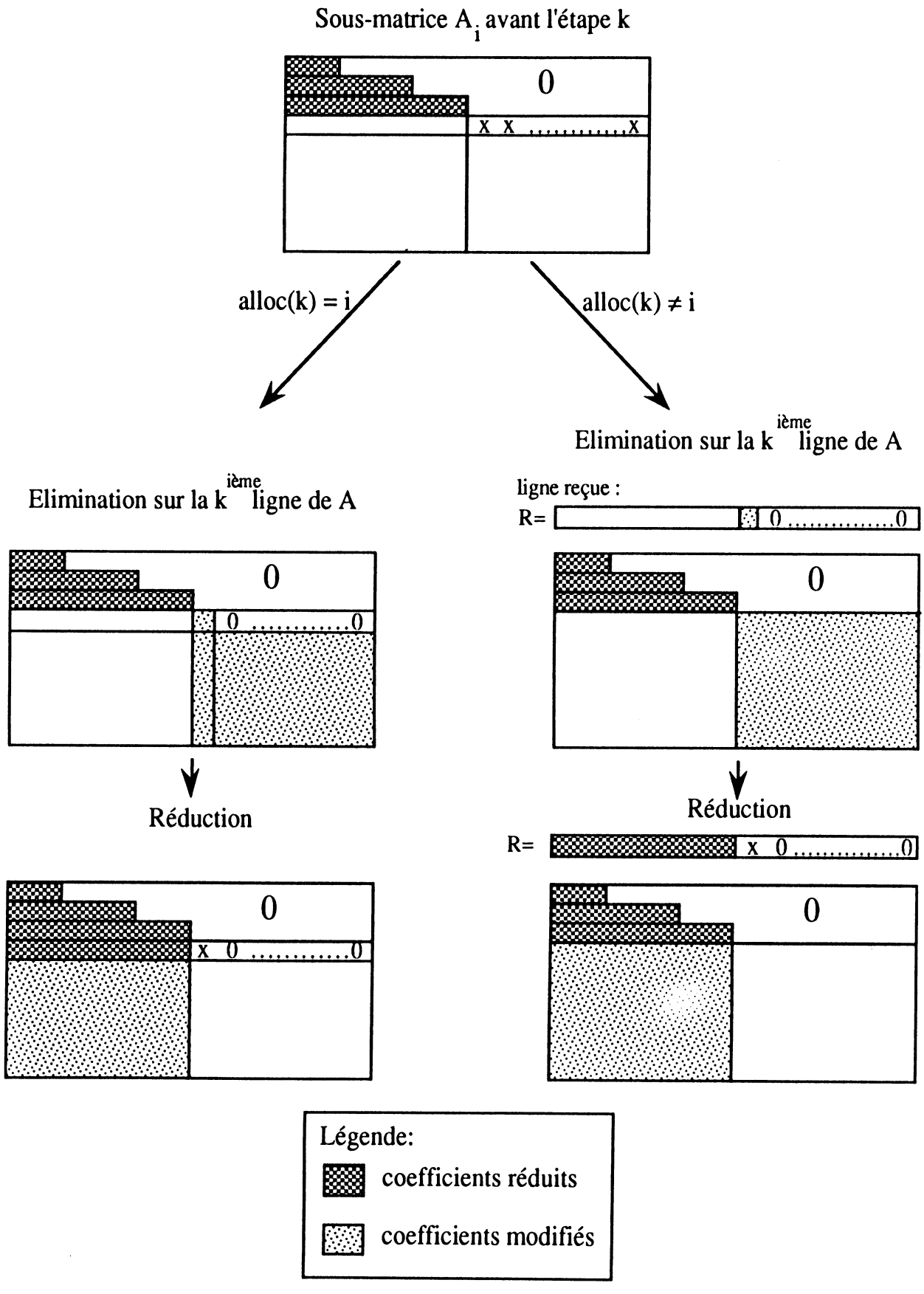


Figure II.13 : Etape k de l'algorithme HL dans le processeur Pi

II.3.2.3. Complexité des deux algorithmes

Pour les deux algorithmes, il est clair que asymptotiquement les coûts arithmétiques sont identiques et qu'ils correspondent au coût séquentiel divisé par le nombre de processeurs. Nous examinons cependant plus en détail les différentes étapes des deux algorithmes.

Coût arithmétique:

• Algorithme HC

A l'étape k, le coût d'une élimination se décompose en:

- le coût du calcul de la transformation à effectuer:

$$T_{pgcd}(\delta) + 2\,T_{div}(\delta)$$

- le coût de son application sur les deux colonnes concernées :

$$(n - k)\,(4T_{mul}(\delta) + 2\,T_{div}(\delta))$$

Cette opération étant effectuée $(n/p + \log_2 p)$ fois au plus, le coût total des éliminations à l'étape k est borné par:

$$(n/p + \log_2 p)\,(\,(n - k)\,(4T_{mul}(\delta) + 2\,T_{div}(\delta)) + 2\,T_{div}(\delta) + T_{pgcd}(\delta)\,)$$

Le coût des réductions est:

$$(\lceil k/p \rceil)\,((n-k+1)(T_{mul}(\delta) + T_{div}(\delta)) + T_{div}(\delta))$$

k variant de 1 à n, on en déduit le coût arithmétique total:

$$\sim (n/p + \log_2 p)(\,n^2\,(2\,T_{mul}(\delta) + T_{div}(\delta)) + 2\,nT_{div}(\delta) + nT_{pgcd}(\delta)\,) + n^3(T_{mul}(\delta) + T_{div}(\delta))\,/6p$$

Coût arithmétique total:

$$\boxed{\sim n^3\,(13T_{mul}(\delta) + 7T_{div}(\delta))/6p}$$

• Algorithme HL

A l'étape k, le coût d'une élimination correspond au coût du calcul de la transformation correspondante et de son application sur la sous matrice:

$$(n/p - \lfloor k/p \rfloor)\,(4T_{mul}(\delta) + 2\,T_{div}(\delta)) + T_{pgcd}(\delta) + 2\,T_{div}(\delta)$$

Le coût total de l'élimination à l'étape k est donc:

$$n\,(\,(n/p - \lfloor k/p \rfloor)\,(4T_{mul}(\delta) + 2\,T_{div}(\delta)) + 2\,T_{div}(\delta) + T_{pgcd}(\delta)\,)$$

Le coût des réductions à cette même étape est:

$$(k-1)(n/p - \lfloor k/p \rfloor)(T_{mul}(\delta) + T_{div}(\delta))$$

k variant de 1 à n, on en déduit le coût arithmétique total:

$$\sim n^2(n(2\,T_{mul}(\delta) + T_{div}(\delta))/p + 2\,T_{div}(\delta) + T_{pgcd}(\delta) + n(T_{mul}(\delta) + T_{div}(\delta))/6p)$$

Coût arithmétique total:

$$\boxed{\sim n^3(13T_{mul}(\delta) + 7T_{div}(\delta))/6p}$$

La seule différence intervenant dans les deux complexités apparaît dans le nombre de pgcd à calculer. On calcule p fois plus de pgcd dans l'algorithme HL que dans l'algorithme HC.

Dans l'algorithme HC, à l'étape k, chaque processeur calcule d'abord n/p pgcd. On utilise ensuite le principe "diviser pour règner":
on calcule en parallèle (n/2p) pgcd, puis (n/4p) pgcd, ..., puis 1 pgcd.

Dans l'algorithme HL, à l'étape k, les n pgcd sont calculés en séquentiel dans chaque processeur.

Cependant, les coûts arithmétiques sont asymptotiquement égaux.

Coût des communications

Dans les deux algorithmes, on suppose que l'on envoie uniquement les coefficients que l'on sait non nuls.
Pour le coût de l'envoi d'une colonne ou d'une ligne, nous détaillerons les processus de communication dans la partie implantation (chapitre III).

• Algorithme HC

A l'étape k, il y a deux processus de communication:

Le coût des communications lors du processus de fin d'élimination correspond au coût de la remontée vers un processeur racine (envoi de colonnes possédant k-1 premiers coefficients nuls):

$$(2\,\beta + (n-k+1)\,\delta\,\tau)\,\log_2 p$$

La diffusion de la colonne k pour les réductions, correspond au schéma inverse et est de même coût en temps de communication.

En sommant sur le nombre d'étapes, on obtient le coût total des communications:

$$\boxed{n(4\ \beta + n\ \delta\ \tau\ )\ \log_2 p}$$

• Algorithme HL

A l'étape k, il y a un seul processus de communication qui correspond à l'envoi de la ligne k (n + k coefficients non nuls) à tous les processeurs, soit:

$$(2\ \beta + (n + k)\ \delta\ \tau\ )\ \log_2 p$$

Le coût total est donc:

$$\boxed{n\ (2\ \beta + 3\ n\ \delta\ \tau\ /2)\ \log_2 p}$$

Conclusion:
Le deuxième algorithme nécessite donc plus de temps de communications effectif (non compris les temps de latence).

Synchronisations :

Nous devons, afin de comparer ces deux algorithmes, tenir compte des problèmes de synchronisations des processeurs. Nous verrons qu'ils jouent un rôle important lors de l'exécution des algorithmes; les temps de latence peuvent influencer considérablement sur les coûts.

Les deux figures suivantes montrent les différentes synchronisations entre processeurs lors du déroulement des deux algorithmes.

Il apparaît que les opérations s'enchaînent plus rapidement lorsque la matrice est répartie par lignes (HL: figure II.15) que lorsqu'elle est répartie par colonnes (HC: figure II.14). En effet, l'algorithme par lignes nécessite beaucoup moins de synchronisation entre processeurs (moitié moins que dans l'algorithme par colonnes). Ce phénomène est prédominant dans les mesures expérimentales et par suite l'algorithme HL s'avère meilleur.

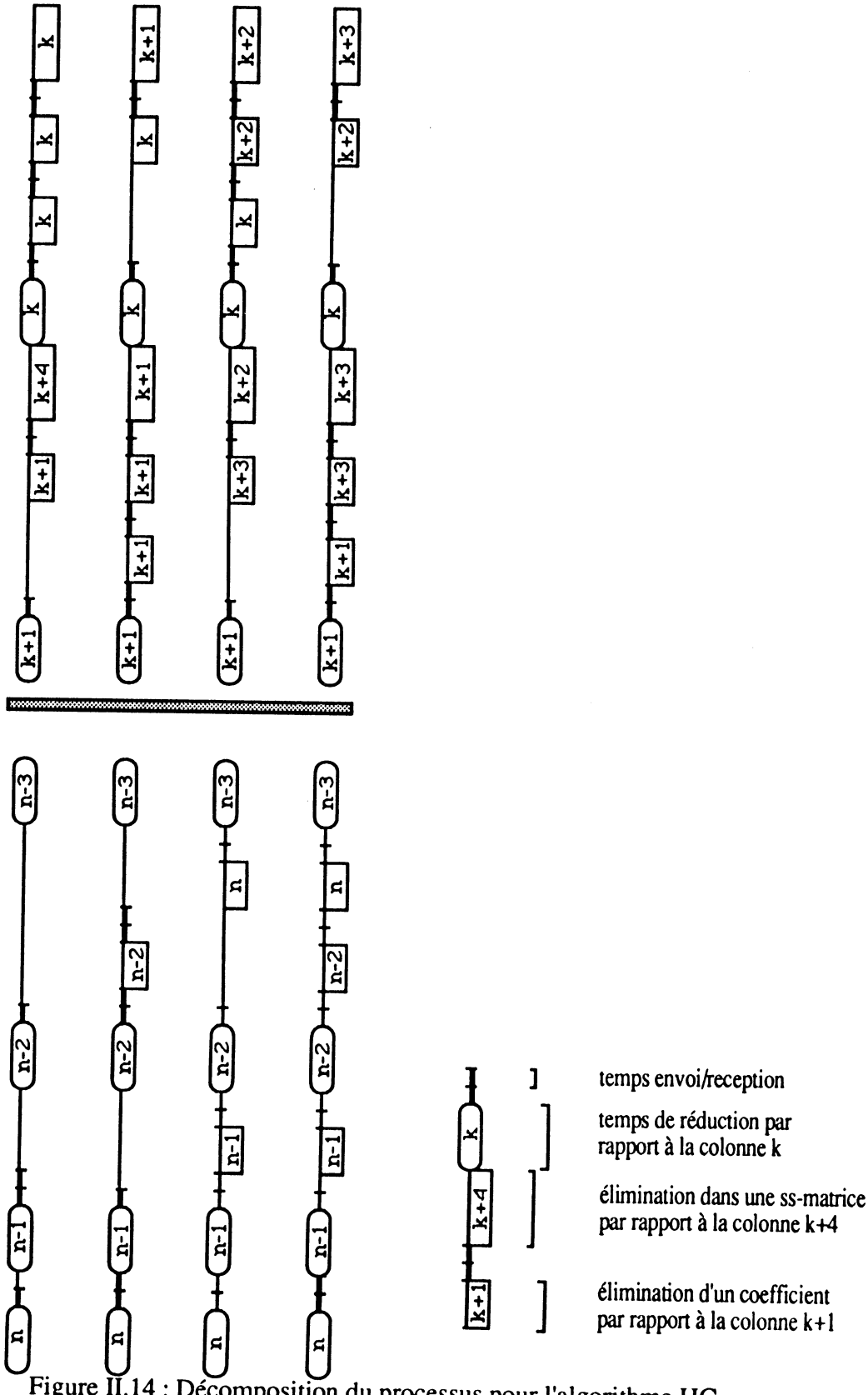


Figure II.14 : Décomposition du processus pour l'algorithme HC

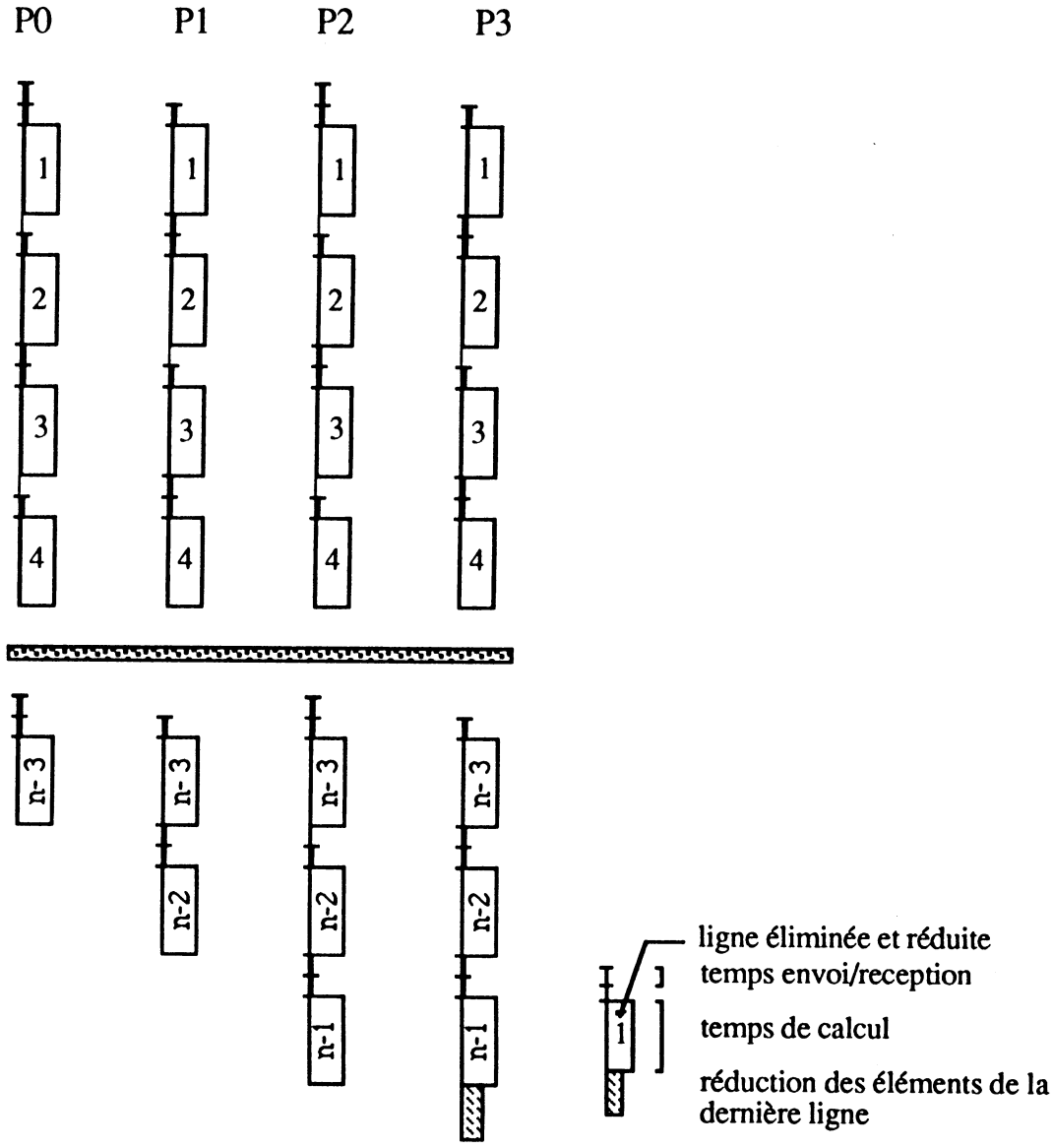


Figure II.15 : Décomposition du processus pour l'algorithme HL

II.4. Algorithme parallèle pour le calcul de la forme normale d'Hermite d'une matrice à coefficients polynomiaux

Comme nous l'avons précisé dans la partie théorique, le problème du cas polynomial est que les algorithmes séquentiels que nous connaissons présentent une croissance des coefficients intermédiaires très importante. Il n'est, en particulier, pas possible de considérer que le coût des éliminations reste du même ordre durant l'exécution de l'algorithme. Il est donc primordial de définir un modèle théorique qui tienne compte de cette contrainte.

II.4.1. Précédence des tâches

Nous avons vu, dans la première partie, que l'algorithme mineur par mineur présentait une meilleur complexité que l'algorithme modulo d car la croissance des coefficients intermédiaires est moindre. Cet algorithme est donc le mieux adapté au problème. Bien que, comme nous l'avons vu dans le cas entier, cet algorithme ne présente pas à priori de tâches indépendantes (élimination) de haut niveau, nous nous attachons ici à en étudier la parallélisation.

Pour conserver les bornes théoriques sur la taille des coefficients, nous devons considérer certaines restrictions sur l'ordre d'exécution des taches.

Les contraintes sont alors:

♦ l'élimination, de l'élément (i, j) ne peut être effectuée qu'après qu'aient été réalisées:

- les éliminations des éléments (1, j), (2, j), ..., (i - 1, j)
- toutes les éliminations de la colonne i
- les réductions sur les lignes i + 1 à j

♦ la (j - k - 1) ième réduction de l'élément (i, j) ne peut être effectuée qu'une fois les éliminations terminées sur la colonne i.

Notons $E^{(i,j)}$ la tâche qui consiste au calcul des coefficients de la transformation d'élimination et à l'application de celle-ci.

On en déduit que :

$$E^{(i,j)}_{i < j < n-1} << E^{(t,\, j+k)}_{1 \le t-j < k < n-j}$$

Par suite, les éliminations sur deux colonnes distinctes ne peuvent être effectuées simultanément. Une répartition par colonne ne peut donc être envisagée car à un instant donné, seule une élimination pourrait être effectuée et un seul processeur (celui possédant cette colonne) pourrait travailler.

II.4.2. Algorithme parallèle

II.4.2.1. Description

Nous allons considérer une répartition par lignes et paralléliser la procédure d'élimination elle-même.
On suppose que la matrice est répartie dans les processeurs, chacun d'eux possédant n/p lignes de la matrice initiale. Une fois encore, on considère la répartition la plus équilibrée.

L'algorithme se décompose en n étapes, l'étape k correspondant à l'obtention de la forme normale d'Hermite du mineur d'ordre k+1.

L'étape k se décompose elle-même en k sous-étapes. La sous-étape t de l'étape k se décompose en:

- une phase de diffusion de la ligne t à partir du processeur qui la possède
- une phase de calcul comprenant:
  - le calcul de la transformation à effectuer pour l'élimination et son application sur les lignes de la sous-matrice.
  - pour chaque élément à réduire sur la ligne t, le calcul de la transformation à effectuer et son application sur la sous-matrice.

II.4.2.2. Algorithme MPL

Nous allons construire un algorithme parallèle, noté MPL, qui calcule la forme normale d'Hermite d'une matrice à coefficients polynomiaux et qui correspond à une répartition par ligne de la matrice.

Nous noterons $(\tilde{a}_{ij})_{1\le i\le n/p\ ,\ 1\le j\le n}$ les coefficients de la sous-matrice du processeur dans lequel on se place.
r est un vecteur de dimension n dans lequel sont receptionnées les lignes envoyées par un processeur voisin. Il sert de mémoire tampon pour les communications.
On définit les opérations suivantes:

- élimination de l'élément (t, k) dans le processeur qui possède la ligne t

E(linit, k, t)==
(u,v,g):=bezout($\tilde{a}_{linit,t}$, $\tilde{a}_{linit,k}$)
pour j:=(linit+1) à n/p faire

$$\begin{pmatrix} \tilde{a}_{jt} \\ \tilde{a}_{jk} \end{pmatrix} := \begin{pmatrix} u.\tilde{a}_{jt} + v.\tilde{a}_{jk} \\ (-\tilde{a}_{linit,k}.\tilde{a}_{jt} + \tilde{a}_{linit,t}.\tilde{a}_{jk})/\, g \end{pmatrix}$$

$\tilde{a}_{linit,t} := g$
$\tilde{a}_{linit,k} := 0$

♦ élimination de l'élément (t, k) dans un processeur qui ne possède pas la ligne t :

E_r (linit, k, t)==
(u,v,g):=bezout(r_l , r_k)
pour j:= linit à n/p faire

$$\begin{pmatrix} \tilde{a}_{jt} \\ \tilde{a}_{jk} \end{pmatrix} := \begin{pmatrix} u.\tilde{a}_{jt} + v.\tilde{a}_{jk} \\ (- r_k.\tilde{a}_{jt} + r_t.\tilde{a}_{jk})/ g \end{pmatrix}$$

r_t : = g; r_k := 0

♦ Réduction des éléments en position (t, 1), (t, 2), ..., (t, t-1) par rapport à l'élément en position (t, t)
dans le processeur qui possède la ligne t:

R(linit, t):
pour c=1 à t-1 faire
quot := pdiv($\tilde{a}_{linit,c}$, $\tilde{a}_{linit,t}$)
pour j=linit à n/p faire
$\tilde{a}_{jc}$:= $\tilde{a}_{jc}$ - quot.$\tilde{a}_{jt}$

dans un processeur qui ne possède pas la ligne t:

R_r(linit, t):
pour c=1 à t-1 faire
quot := pdiv(r_c ,r_t)
pour j=linit à n/p faire
$\tilde{a}_{jc}$:= $\tilde{a}_{jc}$ - quot.$\tilde{a}_{jt}$

Algorithme MPL:

```
PAR pour tout processeur Pi faire
    SEQ
        pour k=1 à n faire
            linit := 1
            index := 1
            tant que index ≠ 0 faire
                pour t=1 à k faire
                    si alloc(t) = 1 alors
                        diffuser ã_linit,1, ã_linit,2, ..., ã_linit,t, ã_linit,k
                        E(linit, k, t)
                        R(linit, k, t)
                    sinon
                        recevoir r1, r2, ..., rt, rk
                        Er (linit, k, t)
                        Rr (linit, k, t)
                fpour
                si alloc(t) = 1 alors
```

```
                    si (ã_linit,t = 0) alors
                        permuter(k, k+index)
                        index := index + 1
                    sinon index := 0
            sinon   si (r_k = 0) alors
                        permuter(k, k+index)
                        index := index + 1
                    sinon index := 0
                fsi
            ftantque
        fpour
```

II.4.4.3. Etude de la complexité

Coût arithmétique:

On note D une borne sur les degrés des polynômes et δ une borne sur la taille de leurs coefficients.

A l'étape t de l'étape k, le coût d'une élimination correspond au coût du calcul de la transformation correspondante et de son application sur la sous matrice:

$$(n/p - \lfloor t/p \rfloor)\,(4T_{pmul}(D,\delta) + 2\,T_{pdiv}(D,\delta)) + T_{ppgcd}(D,\delta) + 2\,T_{pdiv}(D,\delta)$$

Le coût des réductions à cette même étape est:

$$(t-1)(n/p - \lfloor t/p \rfloor)(T_{pmul}(D,\delta) + T_{pdiv}(D,\delta))$$

Le coût total de l'élimination à l'étape k est donc:

$$(\,k(n/p) - p\lfloor k/p \rfloor(\lfloor k/p \rfloor+1)/2)\,(4T_{pmul}(D,\delta) + 2\,T_{pdiv}(D,\delta)) + k(2\,T_{pdiv}(D,\delta) + T_{ppgcd}(D,\delta)\,)$$

Et le coût total de la réduction à cette même étape est:

$$((k(k-1)/2)(n/p) - (p^2(4\lfloor k/p \rfloor+5) - 9p)\lfloor k/p \rfloor(\lfloor k/p \rfloor+1)/12) \cdot (T_{pmul}(D,\delta) + T_{pdiv}(D,\delta))$$

k variant de 1 à n, on en déduit le coût arithmétique total:

Coût arithmétique total:

$$\boxed{\sim (n^4/12p + n^3(1/p + 1/4p^2))(T_{mul}(D,\delta) + T_{div}(D,\delta))}$$

Coût des communications

Le coût de l'envoi d'une colonne ou d'une ligne de polynômes est détaillé dans la partie implantation lors de la description des processus de communication (chapitre III). On supposera que l'on envoie les groupes de polynômes en un seul bloc contigu: cette méthode garantit un temps minimal de communication.

Soit D une borne sur les degrés des polynomes, et δ une borne sur la taille des coefficients.

Le coût des communications lors du processus de diffusion au pas t de l'étape k correspond au coût le l'envoi en $\log_2 p$ étapes de (t+1) coefficients polynomiaux:

$$(2\beta + (t+1) D \delta \tau) \log_2 p$$

En sommant sur le nombre d'étapes, on obtient le coût total des communications:

$$\sum_{k=1}^{n} \sum_{t=1}^{k} (2\beta + (t+1) D \delta \tau) \log_2 p$$

Soit:

$$\boxed{n((n+1)\beta + (n^2/6 + 3(n+1)/4) D \delta \tau) \log_2 p}$$

CHAPITRE III

III. IMPLANTATION

Les algorithmes ont été implantés sur le FPS T40 à 32 processeurs: le réseau de connexion est de type hypercube. Les algorithmes sont écrits en langage C avec quelques primitives de communication par rendez-vous fournies par FPS. Ces communications ayant un coût linéaire avec une forte constante, il est fondamental de concaténer les messages avant envoi pour minimiser les temps de communication.

Le même code exécutable est chargé sur tous les processeurs. Les programmes sont paramétrés par une variable *pos* indiquant le numéro du processeur. Chaque processeur connaît le numéro qui lui est associé et peut ainsi sélectionner le code qui le concerne.

Le développement parallèle a été construit autour du système PAC, auquel nous avons participé pour l'écriture des primitives d'arithmétique polynomiale et d'arithmétique dans les corps finis.

III.1. Cas entier

III.1.1. Programmation

III.1.1.1. Outils arithmétiques

La bibliothèque PAC fournit toutes les primitives de base pour les opérations en précision infinie [ROC89]. Les fonctions sont programmées en C et en assembleur du transputer T414 d'Inmos.

Les structures de données choisies sont adaptées au parallélisme. Un grand entier, par exemple, est stocké dans un bloc de mots consécutifs.

Notons que, pour les deux algorithmes, le calcul du déterminant est effectué par l'algorithme de Gauss avec développements p-adiques. Une parallélisation de cet algorithme a été proposée par G. Villard [VIL88], il est implanté sur le FPS T40. Le programme est chargé indépendamment du programme de calcul de la forme d'Hermite, le déterminant est récupéré dans un fichier. Notons que les temps de calcul du déterminant obtenus sont très faibles par rapport aux temps de calcul de la forme d'Hermite. Nous n'en avons pas tenu compte dans la suite des résultats, une étude expérimentale est effectuée dans la thèse de G. Villard.

III.1.1.2. Primitives de communication

La primitive de base utilisée , fournie par FPS et particulièrement adaptée à la connectique hypercube, est:

oto_x (chemin, dim, b_env, b_rec, &buff_env, &buff_rec, taille)

Elle permet de communiquer une donnée d'un processeur à un voisin immédiat.
Ses paramètres sont:

chemin : définit les canaux de communications,
dim, b_env, b_rec : indiquent la dimension de communication, et le dim-ième bit du processeur qui envoie et de celui qui reçoit
&buff_env, &buff_rec : sont les adresses des variables (envoi et réception)
taille : est la taille du message en nombre d'octets.

A l'aide de cette première primitive, nous avons défini deux autres fonctions, indispensables pour notre problème:

- ota_hyp(chemin, dim_hyp, nb_proc, proc_rac, &buff_env, &buff_rec, taille)

Elle permet d'envoyer une donnée d'un processeur à tous les autres en utilisant la configuration hypercube. Les nouveaux paramètres sont:

dim_hyp : dimension de l'hypercube
nb_proc : nombre de processeurs
proc_rac : processeur racine qui doit communiquer sa donnée à tous les autres.

- oto_VZ(chemin, dim, b_env, b_rec, &vect_env, &vect_rec, dim_vect)

Elle permet de communiquer un vecteur de longs entiers entre deux processeurs voisins directs. Ses nouveaux paramètres sont:

&vect_env, &vect_rec : sont les adresses des vecteurs
dim_vect : est la dimension de ces vecteurs.

Le modèle de complexité pour l'envoi d'un entier doit tenir compte du coût de l'envoi de sa taille et du coût de l'allocation d'un bloc de cette taille par le processeur recepteur. Notons $T_{alloc}(t)$ le temps d'allocation d'un bloc de

taille t et $T_{déplace}(t)$ le coût du déplacement en mémoire d'un entier de taille t. Le coût du transfert d'un entier de taille t est alors:

$$2\beta + t\,\tau + T_{alloc}(t)$$

La fonction oto_VX peut être programmée de plusieurs façons.

1°) On envoie les composantes du vecteur de dimension n une à une. Le coût de l'envoi est alors :

$$2\,n\,\beta + n\,t\,\tau + n\,T_{alloc}(t),$$

2°) On alloue une place de taille nt dans laquelle sont concaténés les n entiers. L'envoi se résume alors à la diffusion d'un seul bloc, et le coût total est:

$$T_{alloc}(n\,t) + n\,T_{deplace}(t) + 2\,\beta + n\,t\,\tau.$$

Or, sur le FPS T40, nous avons $T_{alloc}(nt) \leq n\,T_{alloc}(t)$ et de plus $T_{déplace}(t) \leq 2\,\beta$ si $t < 1400$.

Il est encore possible d'optimiser le coût en cherchant une valeur de N_0 qui, pour une taille t de coefficients et une dimension n données, minimise la fonction:

$$\psi(N) = (N+1)\beta + n\,t\,\tau + N\,T_{alloc}\left(\frac{n\,t}{N}\right)$$

$$\psi(N_0) = \min_{N \text{ entier}} \psi(N)$$

Le transfert se ferait alors en N_0 envois de $\frac{n\,t}{N_0}$ mots consécutifs.

Les tailles des coefficients et les dimensions des vecteurs variant énormément au cours de l'exécution d'un algorithme, c'est la deuxième méthode que nous avons choisie pour l'implantation.

III.1.1.3. Gestion mémoire

Nous utilisons les primitives alloc et free. Les variables cpt_place, cpt_alloue et cpt_lib permettent de connaître dynamiquement la place occupée, la place allouée et la place liberée au cours de l'exécution d'un algorithme. Ils sont en particulier d'une grande utilité pour vérifier la validité des programmes et gérer les libérations.

III.1.1.4. Initialisation de la matrice

Il s'agit ici de générer des matrices d'entiers non singulières de différentes dimensions. Nous avons utilisé un générateur de nombres pseudo-aléatoires uniforme [KNU66]. Les coefficients des matrices sont donnés par l'itération:

$$\begin{cases} u=774755\ a=62605\ \ b=113218009\ \ r=536870912 \\ \text{Pour } i=1\ldots n,\ j=1\ldots n+1 \\ \qquad u=(ua+b) \bmod r \quad A(i,j) = u \bmod B \end{cases}$$

où B est la borne sur les données.

III.1.2. Résultats expérimentaux

La mémoire disponible sur chaque noeud est de un mégaoctet. Nous sommes donc limités dans la taille des exemples traités. Dans les algorithmes parallèles, la matrice est répartie entre les processeurs. L'explosion des coefficients est telle que la plupart des exemples ne peuvent être traités sur un processeur. Ils ont alors uniquement été testés sur plusieurs processeurs.

III.1.2.1. Temps d'exécution

Dans les exemples qui suivent, la borne sur les coefficients en entrée est B=100.

Quel est l'algorithme le plus performant ?

En accord avec l'étude de complexité théorique précédente, l'algorithme HL, bien que du même ordre, s'avère plus performant (d'un facteur constant, dû à l'influence des coûts de synchronisation décrits précédemment).

En considérant que l'algorithme HL a un coût arithmétique supérieur (aspect qui n'apparaît cependant pas dans les complexités asymptotiques) ainsi qu'un coût en communication supérieur (en omettant les temps de latence) à l'algorithme HC, il est clair que les deux étapes de synchronisation ralentissent considérablement ce dernier.

C'est un aspect des communications qui apparaît clairement lors de l'expérimentation.

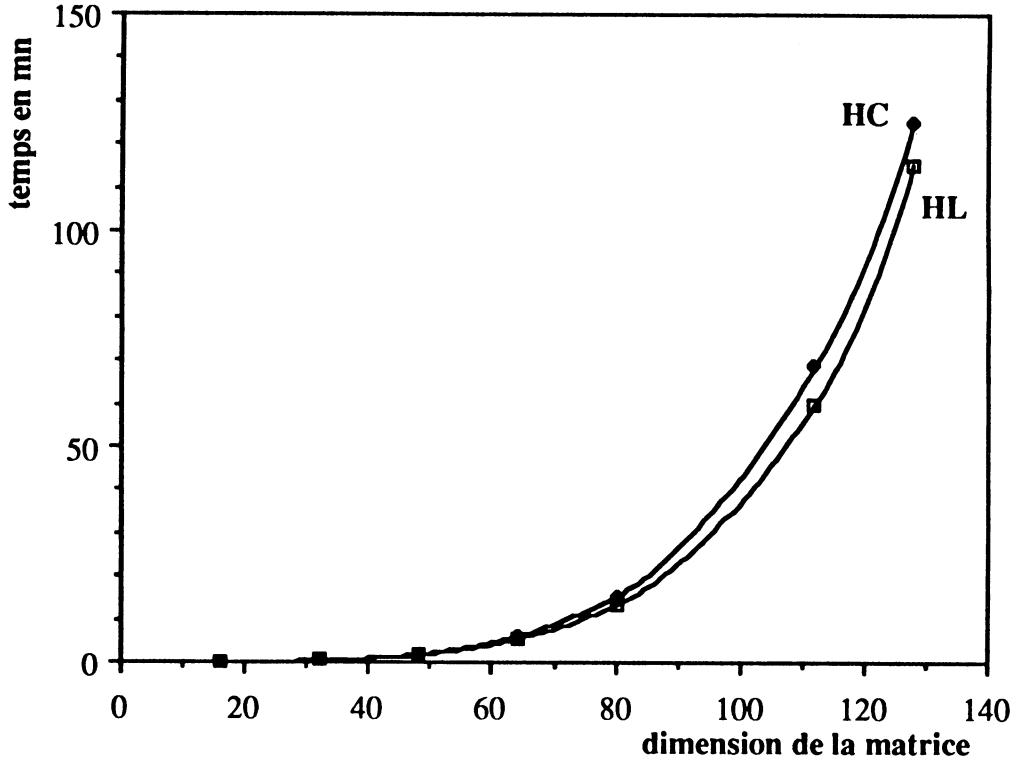


Figure III.1 : Temps d'exécution de HL et HC sur 16 processeurs

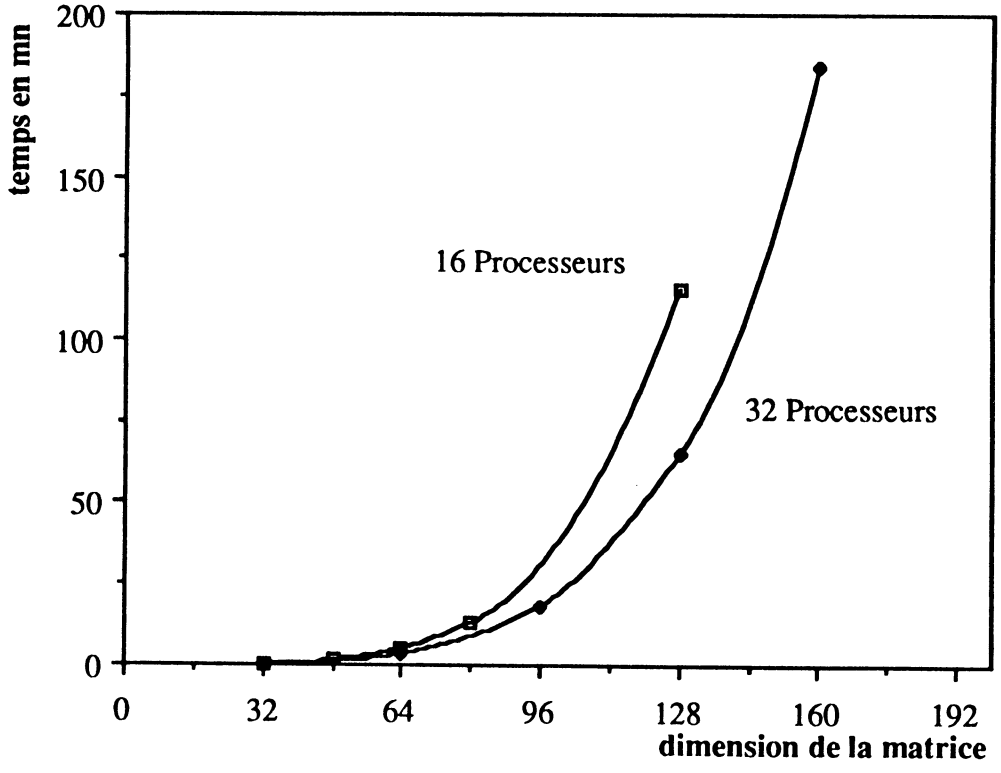


Figure III.2 : Temps d'exécution de HL sur 16 et 32 processeurs

Exemples traités :

Nous avons réussi à calculer la forme d'Hermite d'une matrice 160x160 sur 32 processeurs en 3 heures.

Le traitement d'une matrice 128x128 a pris 2 heures sur 16 processeurs au lieu de 1 heure 10 minutes sur 32.

Il est clair que nous ne pourrions envisager de calculer la forme normale d'Hermite d'une matrice 160x160 en séquentiel. Le gain en place mémoire et en puissance de calcul apporté par le parallélisme nous a permis de repousser considérablement les limites du problème.

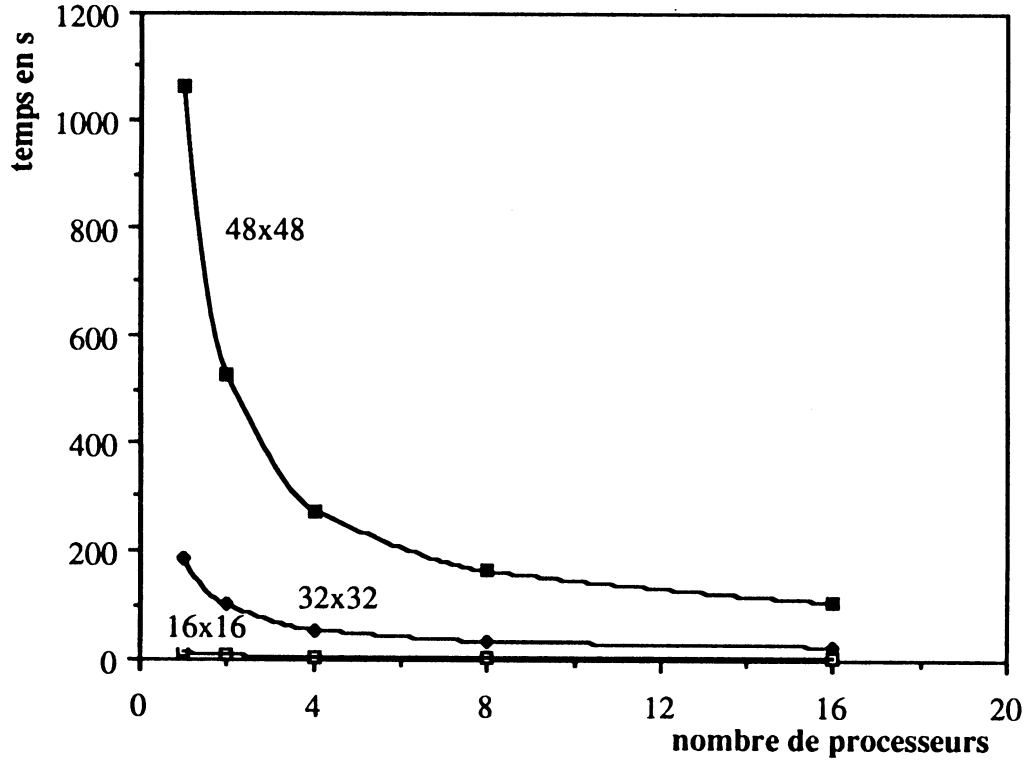


Figure III.3 : Temps d'exécution de HL pour des petites dimensions

La figure III.3 montre que la parallélisation reste très intéressante, même pour des matrices de dimension relativement faible.

Nous comparons sur la figure III.4 les résultats obtenus en faisant varier la taille des données.

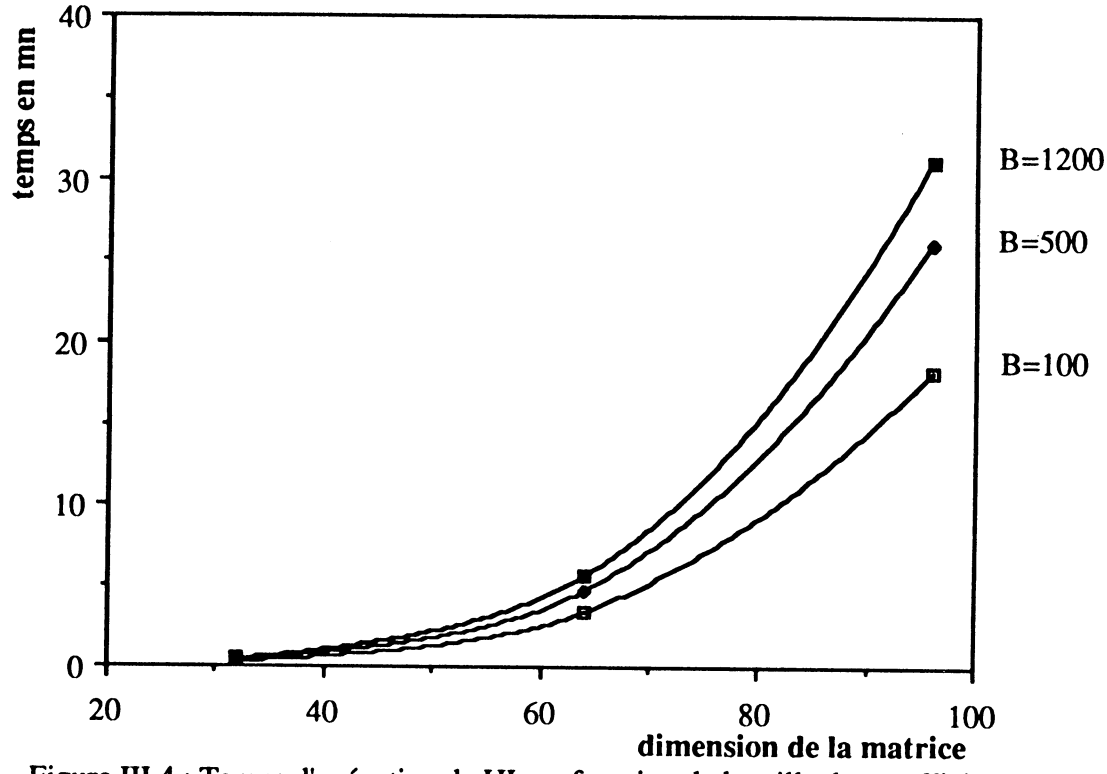


Figure III.4 : Temps d'exécution de HL en fonction de la taille des coefficients

Pour donner une idée plus précise de la taille des coefficients manipulés en cours d'exécution nous donnons ci-dessous un tableau récapitulatif de la taille des déterminants des matrices considérées, en nombre de mots de 32 bits.

Dimension de la matrice	borne sur les coefficients	taille du déterminant
32	100	8
"	500	10
"	1200	12
64	100	17
"	500	21
"	1200	24
96	100	26
"	500	32
"	1200	64

Conclusion quant à l'influence de la taille des coefficients :

On retrouve donc un comportement similaire pour différentes tailles de coefficients.

III.1.2.2. Efficacité

III.1.2.2.1. Efficacité classique

L'efficacité est la quantité qui nous permet d'évaluer la qualité d'une parallélisation. Elle est définie comme le rapport:

$$e = \frac{\text{temps de l'algorithme sur un processeur}}{\text{p x temps de l'algorithme sur p processeurs}}$$

Pour des matrices de petites dimensions l'efficacité est déjà très proche de 1 (maximum théorique). Les problèmes de plus grande dimension ne pouvant être traités sur un processeur, nous n'avons pu en calculer l'efficacité. Mais il est clair que, au vu des résultats obtenus (figure III.1 et III.2), cette efficacité ne peut que s'améliorer quand la dimension des matrices croît.

Comme les courbes suivantes le prouvent, on trouve un comportement semblable pour des données de tailles différentes.

Conclusion :

On peut donc considérer que la parallélisation de cet algorithme est très intéressante, tant au point de vue gain en temps de calcul que gain en dimension des problèmes traités.

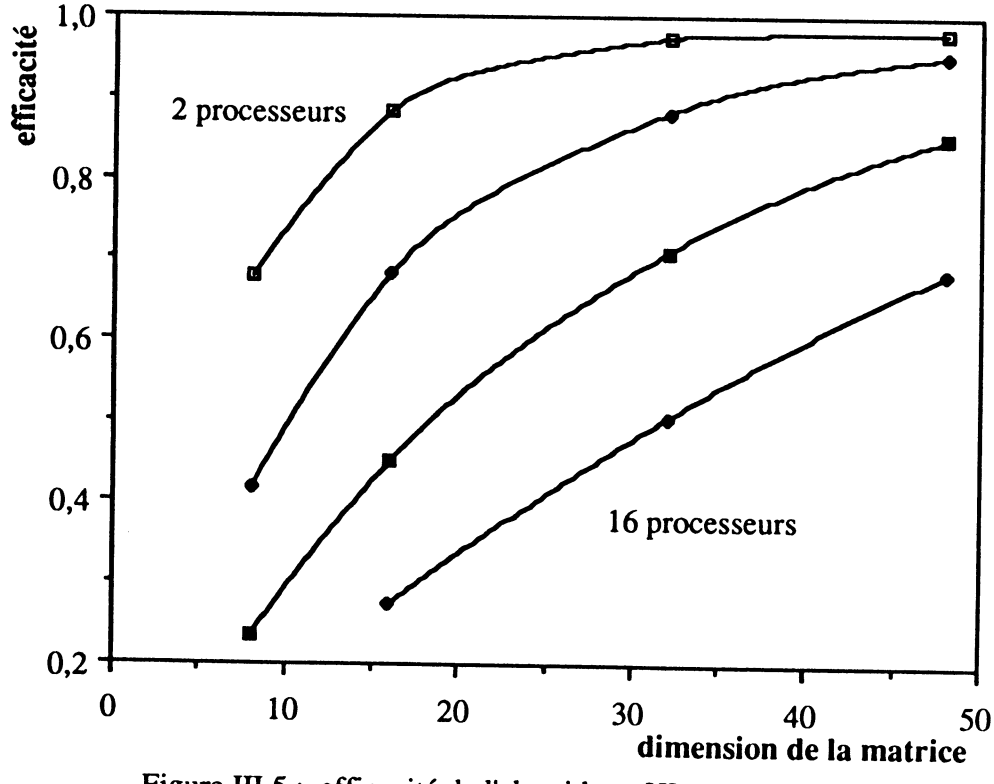


Figure III.5 : efficacité de l'algorithme HL pour B=100

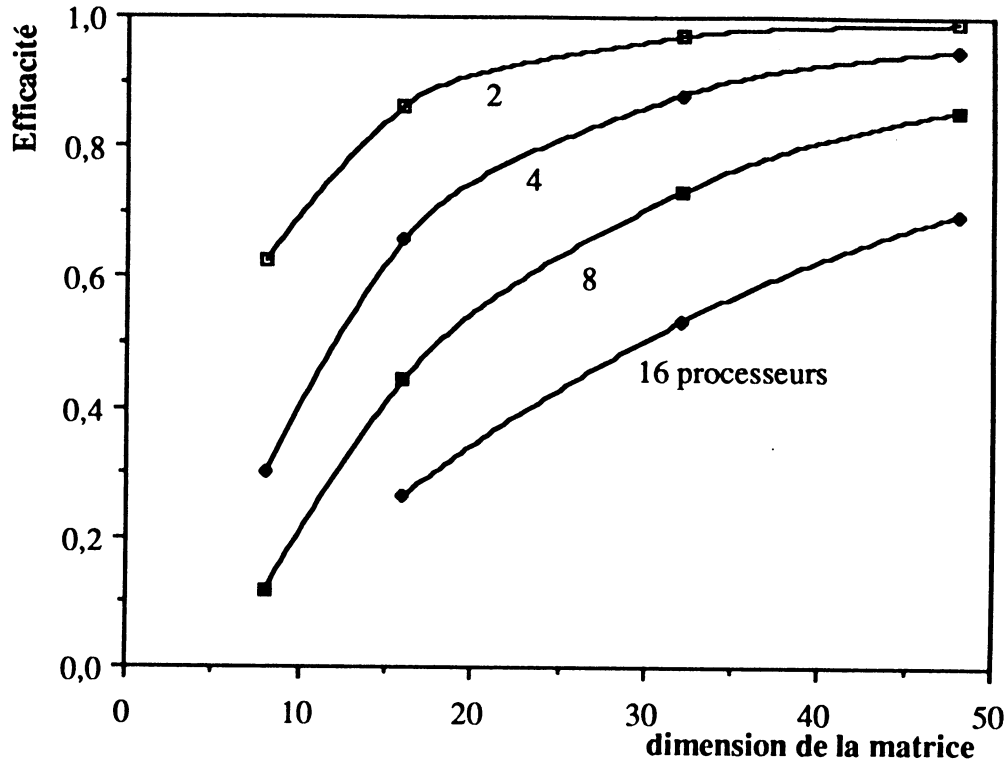


Figure III.6 : efficacité de HL pour B=1200

III.1.2.2.2. Efficacité de Gustafson

La définition de l'efficacité précédente ne permet pas de prendre en compte le fait que le parallélisme permet de traiter des problèmes de taille d'autant plus importante que le nombre de processeurs est grand.

Ainsi, on ne peut sur 32 processeurs évaluer l'efficacité que sur la comparaison entre le meilleur algorithme séquentiel sur 1 processeur pour une matrice de taille inférieure à 64. Il est pourtant clair que sur une matrice de cette taille, de nombreux processeurs (parmi les 32 processeurs utilisés par l'algorithme parallèle) seront mal utilisés.
Bien plus, le fait que l'on peut traiter une matrice de taille 192 ne peut être pris en compte dans le calcul de l'efficacité.

Pour pallier à ce problème typique des architectures distribuées, J.L. Gustafson [GUS88] [TOU89] propose une généralisation de l'efficacité de Lahmdal.

Soit C(n) la complexité arithmétique (nombre d'opérations de base, i.e. sur des mots machines) de l'algorithme séquentiel pour le problème de taille n; ici, pour la forme normale d'Hermite, on a (Partie I) : $C(n) = n^3 . d^2$

Pour comparer le temps parallèle $T_{//p}(n)$ obtenu sur p processeurs, au temps séquentiel $T_{seq}(n)$ sur 1 processeur, l'accélération classique peut aussi être définie comme suit :

$$A(n,p) = \frac{\tau_n(1)}{\tau_n(p)} \qquad \text{avec} \begin{cases} \tau_n(1) = \dfrac{T_{seq}(n)}{C(n)} \\ \tau_n(p) = \dfrac{T_{//p}(n)}{C(n)} \end{cases}$$

$\tau_n(p)$ représente alors le temps moyen d'une opération arithmétique pour le problème de taille n.
Il est alors possible de considérer le gain apporté du fait que l'on peut traiter avec plus de processeurs (et donc un espace mémoire plus grand) un problème de taille plus importante : il suffit de comparer les temps moyens d'une opération arithmétique pour le problème de taille maximale pouvant être traité sur 1 processeur ou sur p processeurs.
Soit $\tau_{max}(p)$ le temps moyen d'une opération arithmétique pour le problème de taille maximale pouvant être traité sur p processeurs.
$\tau_{max}(p)$ permet alors de prendre en compte tous les temps de communication ou de contrôle, mais aussi les temps de gestion mémoire, et le fait essentiel que l'espace mémoire est multiplié par un facteur p sur p processeurs -par rapport à un processeur-.

L'accélération de Gustafson est alors définie ainsi :

$$\boxed{A_G(p) = \frac{\tau_{max}(1)}{\tau_{max}(p)}}$$

L'efficacité de Gustafson associée est alors :

$$\boxed{e_G(p) = \frac{\tau_{max}(1)}{p.\tau_{max}(p)}}$$

Détermination de la taille maximale du problème

Etant donné un nombre p de processeurs, on cherche la taille n_{max} maximale du problème pouvant être traité sur ces p processeurs, les coefficients de la matrice en entrée étant borné par B=100.
La complexité du problème est ici $n^3.d^2$, (n taille de la matrice, d taille du déterminant). La mémoire physique sur chaque processeur du FPS-T40 est limitée à 1 Mo. On en déduit donc, en tenant compte de la place mémoire occupée par le programme et les tampons de communication, que n et d doivent vérifier l'équation :

$$n_{max} = 256.\sqrt{\frac{p}{d+1}} \qquad \text{(où } n_{max} \text{ est un multiple de p)}$$

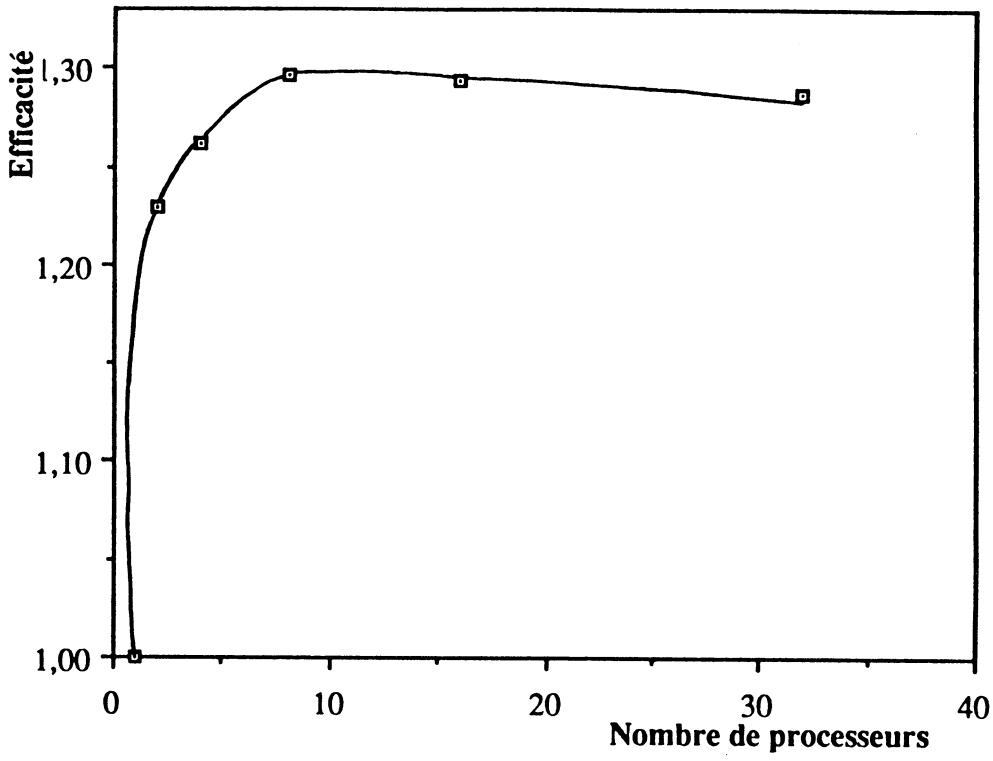


Figure III.7 : efficacité de Gustafson pour HL avec B=100

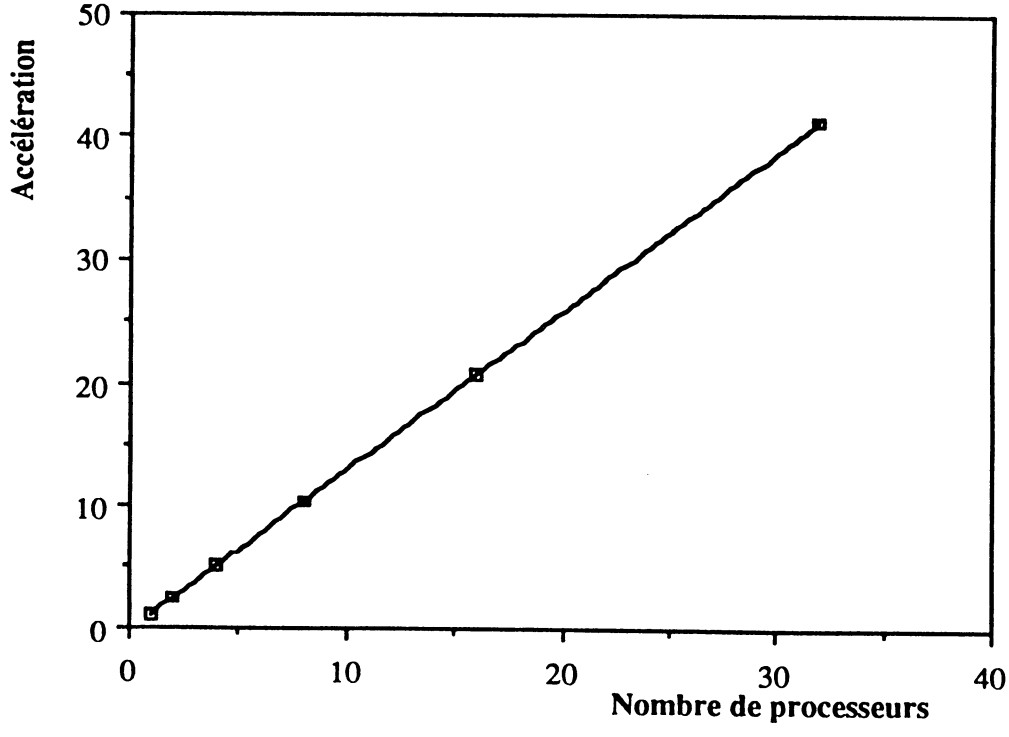


Figure III.8 : accélération de Gustafson pour HL avec B=100

Résultats expérimentaux obtenus :

p	n_{max}	Temps (s)	C(n)	$\tau_{max}(p)$	Accélération	Efficacité
1	64	3970	$7,6.10^{7}$	$5,2.10^{-5}$	1	1
2	80	7259	$3,5.10^{8}$	$2,1.10^{-5}$	2,47	1,23
4	100	7590	$7,3.10^{8}$	$1,03.10^{-5}$	5,05	1,26
8	120	9425	$1,9.10^{9}$	5.10^{-6}	10,4	1,297
16	160	19918	$7,9.10^{9}$	$2,5.10^{-6}$	20,7	1,294
32	192	26400	$2,1.10^{10}$	$1,3.10^{-6}$	**41,2**	1,287

Tableau Récapitulatif des différentes valeurs moyennes expérimentales

Conclusions

On peut remarquer que l'accélération est linéaire, et l'expérience prouve donc que la parallélisation de ce problème est très intéressante.

La forme générale de la courbe d'efficacité est bien celle attendue. L'efficacité la plus intéressante obtenue est (d'après le tableau ci-dessus) celle effectuée pour une matrice de dimension 120, sur l'hypercube de dimension 3.

III.1.2.3. Mémoire

Meilleure stratégie pour le choix de l'allocateur mémoire

L'allocation mémoire d'un objet est dynamique, c'est à dire qu'elle est effectuée directement lorsque lors d'une opération on génère un nouvel objet (par exemple, lors des opérations arithmétiques).
Pour la libération d'un objet temporaire (coefficient intermédiaire), a priori deux choix sont possibles :

- effectuer les libérations de manière *explicite* : on utilise alors une fonction de libération qui permet de libérer un objet dès que l'on sait qu'il ne sera plus utilisé par la suite. Cela est en accord avec la stratégie choisie dans la version du système PAC que nous avons utilisée [ROC89].
- effectuer les libérations de manière *implicite* : lorsqu'il n'y a plus de place pour les allocations, le gestionnaire fait appel à un ramasse-miettes qui libère alors toutes les places mémoire occupées par des objets qui ne sont pas référencés -même indirectement- par les variables manipulées dans le programme (garbage-collector).

Nous appellerons place mémoire virtuelle cumulée la place mémoire qui serait nécessaire au stockage de tous les coefficients intermédiaires.

Nous remarquons expérimentalement (fig III.9) que la place virtuelle cumulée lors de l'exécution de l'algorithme est particulièrement importante. La place utilisée pour une matrice 160x160 est de 171 mégaoctets par processeur, soit un total de 5,5 Gigaoctets. Cela est considérable devant la mémoire disponible sur un processeur qui n'est que de 1 mégaoctet, même si la place réellement utilisée à un instant de l'exécution est, bien sûr, bien moindre. La gestion mémoire intervient donc de façon non négligeable dans les temps de calcul.

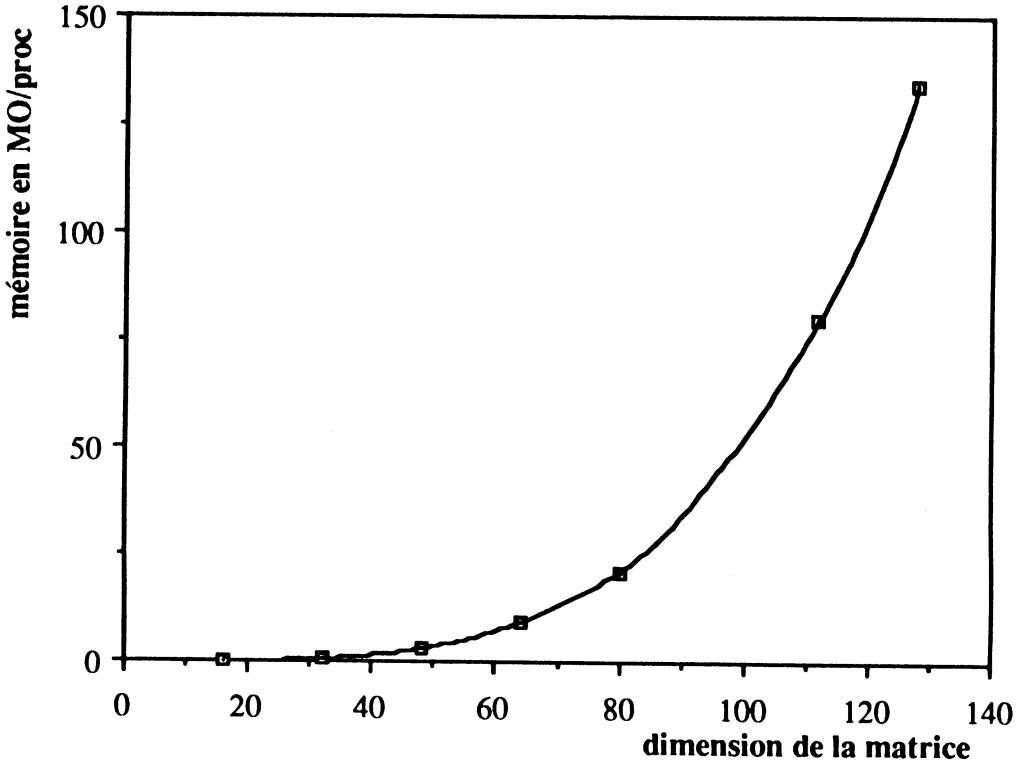


Figure III.9 : Mémoire virtuelle cumulée dans l'algorithme HL

Conclusion

Il apparaît donc clairement que la place mémoire occupée pour le calcul de la forme normale d'Hermite est considérable, et que, par suite, en cas de libération implicite, le système serait obligé de générer de nombreux appels au ramasse-miettes (au moins 200 par processeur -et c'est une borne bien inférieure- pour le plus gros exemple traité : 160x160 sur 32 processeurs). Outre le coût additif que cela entraîne sur chaque processeur, l'effet de désynchronisation entre les processeurs, engendré par ces appels, peut s'avérer important.

III.2. Cas polynomial

III.2.1. Programmation

III.2.1.1. Outils arithmétiques

PAC contient des primitives de base pour les opérations sur les polynômes à coefficients rationnels[ROC89]. En outre, des algorithmes de multiplication et de division performants y sont implantés.

III.2.1.2. Primitives de communication

De même que dans le cas entier, il est préférable de concaténer les messages. Nous avons donc défini une primitive permettant l'envoi d'un vecteur de polynômes d'un processeur à l'un de ses voisins immédiats:

oto_VQX(chemin, dim, b_env, b_rec, &buff_env, &buff_rec, dim_vec, &vec_env, &vec_rec)

Les premiers paramètres correspondent aux paramètres décrits dans le cas entier.

Le protocole de communication d'un polynôme est alors:

le processeur émetteur:
- recopie les polynômes un à un dans un tampon buff_env
- envoie la taille du tampon en nombre de mots
- envoie le tampon

le processeur récepteur:
- reçoit la taille du tampon
- reçoit le tampon dans buff_rec
- reconstruit le vecteur de polynômes en mettant à jour les adresses.

Notons qu'en pratique, on alloue dès le début un tampon suffisamment grand dans chaque processeur qui sera utilisé pour toutes les communications (ce tampon pouvant éventuellement être libéré pour en réallouer un de taille plus grande (dynamiquement) au cours de l'algorithme, si sa taille n'est pas suffisante).

Les polynômes reçus étant susceptibles d'être libérés, il est de toute façon nécessaire, après réception dans le tampon, de s'allouer une nouvelle place pour les stocker.

Le coût de l'envoi d'un vecteur de n polynômes de degrés bornés par d et dont les coefficients sont de taille bornée par δ, d'un processeur à son voisin est donc:

$$2\beta + 2\,n\,(d+1)\,T_{déplace}(\delta) + n\,(d+1)\,\delta\,\tau + n\,d\,T_{alloc}(\delta)$$

Nous avons également défini une primitive qui permet de diffuser, depuis un processeur racine, un vecteur de polynômes dans tous les processeurs de l'hypercube en utilisant le schéma habituel.
Seul le processeur racine, au premier envoi, déplace les polynômes pour les mettre dans le tampon. Les autres processeurs reçoivent dans leur tampon et communiquent, s'il y a lieu, directement le contenu de ce tampon à leurs voisins en dimension supérieure.

De plus, chaque processeur met une seule fois à jour son vecteur et ceci après avoir terminé ses envois.
Il n'y a donc qu'un temps de déplacement et qu'un temps de mise à jour.

Le temps total est alors:

$$n\,(d+1)\,T_{déplace}(\delta) + (2\beta + n\,(d+1)\,\delta\,\tau)\log_2 p + n\,d\,T_{alloc}(\delta) + n\,(d+1)\,T_{déplace}(\delta)$$

Soit,

$$(n\,(d+1)\,(T_{déplace}(\delta) + n\,d\,T_{alloc}(\delta)) + (2\beta + n\,(d+1)\,\delta\,\tau)\log_2 p$$

Le temps de déplacement et d'allocation devient négligeable quand p est grand. Il est donc d'autant plus rentable de concaténer les polynômes que la dimension de l'hypercube est grande.

III.2.1.3. Gestion mémoire

Des primitives de libération mémoire polynome_free et monome_free de la bibliothèque PAC permettent de libérer l'emplacement utilisé pour le stockage des polynômes et des monômes. Elles sont construites à partir de la primitive free du langage C. D'autre part, des additions et des multiplications de polynômes avec libération des opérandes sont à notre disposition.

III.2.1.4. Initialisation de la matrice

A partir d'une borne sur les degrés et les coefficients des polynômes, on construit une matrice de polynômes en utilisant le générateur aléatoire pour le cas entier.

Nous traitons également des matrices du type (A-xI), où A est une matrice à coefficients entiers.

III.2.2. Résultats expérimentaux

III.2.2.1.Dans Q[x]

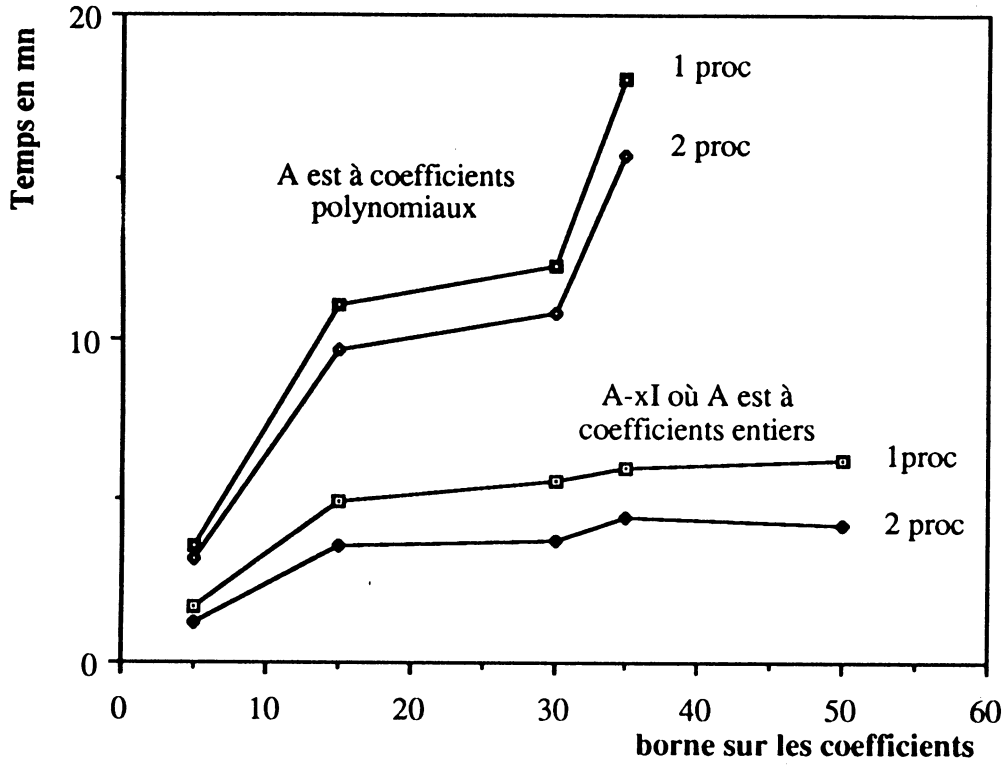


Figure III.10 : Temps d'exécution pour des matrices d'ordre 6
Les coefficients sont des polynômes de degré 1

Il est clair que les matrices ayant des polynômes uniquement sur la diagonale donnent de meilleurs résultats. Cette remarque est essentielle pour le calcul de la forme de Jordan. L'accroissement des coefficients est alors moins significatif et les calculs sont mieux répartis.

Les processeurs qui possèdent les dernières lignes sont toujours défavorisés. Les coefficients, en fin d'algorithme sont énormes et, en particulier pour la dernière ligne, à la dernière étape, seul le processeur qui la possède travaille.
Les calculs, selon la ligne considérée, sont donc de coûts très différents et les processeurs ne sont donc pas du tout synchronisés.

Les calculs n'étant pas homogènes, le traitement parallèle devrait être effectué à un niveau plus bas. Cependant, le coût des communications étant important sur le FPS T40, il n'est pas approprié de faire un découpage de la matrice au dessous de la ligne ou de la colonne.

D'autre part, le coût de l'algorithme étant très important de part la taille des coefficients intermédiaires, nous ne pouvons traiter que des petites matrices sur un petit nombre de processeurs, et les capacités de la machine sont loin d'être mises à profit.
Cet algorithme nécessiterait d'être parallélisé au niveau de l'opération entre deux coefficients, déjà fort complexe, avec une répartition dynamique des tâches pour espérer de meilleures performances.

III.2.2.2.Dans $\mathbb{Z}/p\mathbb{Z}[x]$

Afin d'étudier l'influence du grossissement des coefficients des polynômes intermédiaires nous avons programmé l'algorithme pour des matrices à coefficients dans $\mathbb{Z}/p\mathbb{Z}[x]$.

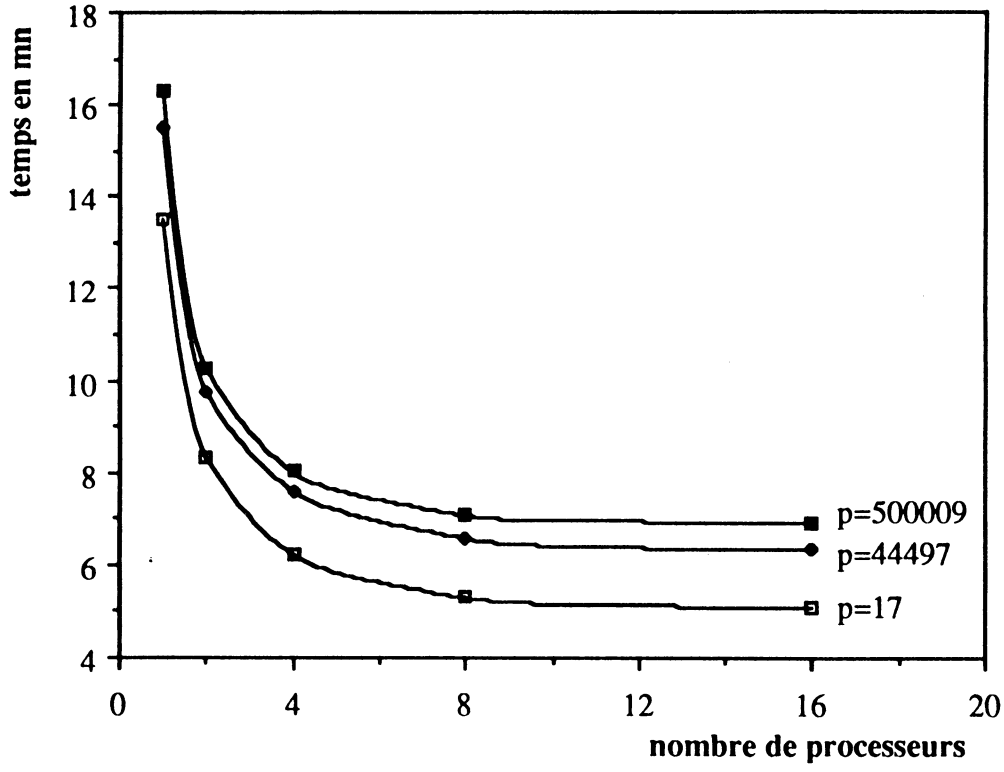


Figure III.11 : Temps d'exécution dans $\mathbb{Z}/p\mathbb{Z}[x]$

Les trois courbes obtenues pour des p distincts présentent un comportement similaire. La parallélisation apporte un gain de temps jusqu'à 16 processeurs, mais l'efficacité est assez faible.
Les temps obtenus sont cependant beaucoup plus faibles que dans $\mathbb{Q}[x]$. Il est impossible, dans $\mathbb{Q}[x]$, de traiter des matrices d'ordre 16 en un temps raisonnable.

Il est clair que le gain apporté par le parallélisme est plus net que dans le cas des polynômes à coefficients dans $\mathbb{Q}[x]$. Même si les degrés des polynômes augmentent en cours d'exécution, les opérations de base sur leurs coefficients sont quasiment en temps constant (seul le calcul de l'inverse modulo dépend de manière sensible des opérandes). Le déséquilibre entre les tâches est donc moins important que dans le cas des polynômes de $\mathbb{Q}[x]$.

Exemple sur 4 processeurs: matrice 16x16 dans $\mathbb{Z}/17\mathbb{Z}[x]$

processeur n°	0	1	2	3
t_com	231	258	300	354
t_calcul	133	106	58	11

t_com est le temps de communication, y compris les temps de latence, en secondes, et t_calcul est le temps de calcul en secondes.

Dans tous les exemples traités, il apparaît que le processeur P_{p-1} (qui possède la dernière ligne) est celui qui effectue le plus de calculs et le moins de communications. C'est un aspect qui n'apparaissait guère dans le cas entier, car toutes les opérations étaient de même taille, et même si le processeur P_{p-1} effectuait davantage de calculs, les calculs étant de même coût, le décalage était moins net.

CONCLUSION

Le calcul de la forme normale d'Hermite s'avère être un problème très intéressant. D'une part ses similarités algorithmiques avec les problèmes de calcul de factorisation orthogonale (Householder, Givens) qui ont fait l'objet d'études conséquentes en calcul numérique, en font un problème test pour le calcul Formel, et un moyen de comparaison entre les deux méthodologies: il apparaît que, sous certaines hypothèses, certaines stratégies de parallélisation s'avèrent intéressantes dans les deux cas.

D'autre part ce problème possède de manière intrinsèque et exacerbée une caractéristique bien connue du monde du Calcul Formel: le grossissement des coefficients intermédiaires.

L'étude de la parallélisation doit donc tenir compte de ce problème souvent difficile à modéliser : outre le coût important et sans cesse croissant des calculs, il entraîne de nombreuses difficultés dans la détermination théorique d'une borne pour le conditionnement du problème. Cette contrainte apparaît lors de l'implantation au travers du volume mémoire utilisé particulièrement conséquent .

Ce problème constitue donc un problème test, et permet d'évaluer le comportement d'une machine cible face à un problème du Calcul Formel riche en applications.
Ces deux particularités fondamentales (coût en calcul et volume mémoire) en font un problème directement adapté au traitement sur une machine massivement parallèle, chaque processeur gérant sa propre mémoire et effectuant des calculs indépendants.

Nous n'avons étudié ici que la parallélisation au niveau de l'algorithme. Pour utiliser la puissance maximale d'une machine ayant un grand nombre de processeurs, il est fondamental de disposer de stratégies de répartition dynamique des tâches, seules capables d'assurer un équilibre d'occupation des différents processeurs du réseau et donc de fournir une bonne efficacité. Les problèmes matriciels plus complexes, comme le calcul de la forme de Jordan ou de Smith, ne pourront être effectivement résolus que lorsque ce pas important sera franchi.

BIBLIOGRAPHIE

[BLA67] Blankinship W.A., "A new version of the Euclidean algorithm", Amer. Math. Monthly 70(3), 1967.

[BOR77] Borodin A., "On relating time and space to size and depth", SIAM J. Comp., 6, p. 733-744, 1977.

[BOR82] Borodin A., Von zur Gathen J., Hopcroft J., "Fast Parallel matrix and GCD computations", Inform. Control, 52, p. 241-256, 1982.

[BRA88] Bradford R.J., "On the Computation of Integral Bases and Defects of Integrity", Ph.D Thesis of the University of Bath, 1988

[BRA70] Bradley G.H., "Algorithm and Bound for the Greatest Common Divisor of n Integers", Communications of the ACM 13(4), p. 433-436, July 1970.

[BRA71] Bradley G.H., "Algorithms for Hermite and Smith Normal Matrices and linear Diophantine Equations", Mathematics of Computation 25(116), p. 897-907, October 1971.

[CHO82] Chou T-W.J., & Collins G.E., "Algorithms for the Solution of Systems of Linear Diophantine Equations", SIAM J. Computing 11(4), p. 687-708, November 1982.

[COF73] Coffman E.G. et Denning P.J. "Operating system theory", Prentice Hall, Englewood Cliffs, NJ, 1973.

[COL67] Collins A. E., "Subresultants and reduced polynomial remainder sequences", J. ACM 14 (1), 1967.

[COO83] Cook S.A. "The classification of problems which have fast parallel algorithms", Proc. International Conference on Foundations of Computation Theory, Borgholm, Lectures Notes in Computer Science 158, Springer-Verlag, Berlin - New york, p. 73-93, 1983.

[COO87] Cook S.A. "An overview of computational complexity", ACM Turing Award Lectures, New York - New York, p. 411-433, 1987.

[COR87] Cosnard M., Robert Y., "Algorithmique Parallèle : une étude de complexité", Techniques et Sciences Informatique 6, 2, p. 115-125, 1987.

[DAO89] Daoudi E.M., "Etude de la complexité de la Decomposition Orthogonale d'une matrice sur plusieurs modèles d'Architectures Parallèles", INPG, Grenoble 1989.

[DAV90] Davenport J., Communication personnelle, Janvier 1990.

[DIX82] Dixon J.D., "Exact solution of linear equations using p-adic expansions ", Numerishe Math. 40, p. 137-141, 1982.

[DOM87] Domich P.D. , Kannan R., & Trotter L.E., "Hermite Normal Form Computation using modulo Determinant Arithmetic", Mathematics of Operations Research 12(1), p. 50-59 February 1987.

[DST86] Davenport J., Siret Y., Tournier E., "Calcul Formel : Systèmes et Algorithmes de Manipulations Algébriques", Masson- collection eri, 1986

[EBE85] Eberly W., "Parallel matrix and polynomial arithmetic", Ph. D. Thesis, Toronto, 1985.

[FLY66] Flynn M. J., "Very high speed computing systems", Proc. IEEE, 54, P. 1901-1909, 1966.

[FRU77] Frumkin M. A., "Polynomial Time Algorithms in the Theory of Linear Diophantine Equations , M. Karpinski (Ed.), Fundamentals of Computation Theory, Springer, Berlin and New York, Lectures notes in computer Science 56, p. 386-392.

[GAR72] Garfinkel R., Nemhauser G.L., "Integer Programming", J. Wiley and Sons, New York, 1972.

[GER88] Gerasoulis A., Nelken I., "Gaussian Elimination and Gauss-Jordan on MIMD Architectures", LCSR-TR-105, Rutgers University, 1988.

[GUS88] Gustafson J.L., "Reevaluating Amdhal's Law", Communications of the ACM, 31, 5, p. 532-533, 1988

[HER51] Hermite C., "Sur l'introduction des variables continues dans la théorie des nombres", J. Reine Angew. Math., 41, p. 191-216, 1851.

[HEY70] Heymann M., Thorpe J.A., "Transfer Equivalence of Linear Dynamical Systems", SIAM J. Control, Vol. 8, N°1, 1970.

[HIL00] Hilbert D., Second congrés international des mathématiciens, Paris, Aout 1900.

[Ill85] Illopoulos, "Gaussian Elimination over a Euclidean Ring", Proc. EUROCAL, Buchberger B. (Ed), Springer LNCS 204, p. 29-30, 1985.

[JHH89] Jhonsson S.L. et Ho C.T., "Distributed Routing Algorithms for Broadcasting and Personnalized Communication in Hypercubes", IEEE Trans. on Computers, vol.38, n°9, p. 1249-1268, 1989.

[KAL87] Kaltofen E., Krishnamoorthy M.S., Saunders B.D., "Fast Parallel Computation of Hermite and Smith Forms of Polynomial Matrices", SIAM J. Alg. Disc. Meth, Vol. 8. N°4, octobre 1987.

[KAN80] Kannan R., "The Size of Numbers in the Analysis of Certain Algorithms", PH.D. Thesis, Cornell University, May 1980.

[KAN85] Kannan R., "Solving systems of Linear Equations over Polynomials", Theoretical Computer Science, 39, p. 69-88, 1985.

[KAN79] Kannan R., & Bachem A., "Polynomial Algorithms for Computing the Smith and Hermite Normal Forms of an integer Matrix", SIAM J. Computing 8(4), p. 499-507, November 1979.

[KNU66] Knuth D.E., "The art of computer programming", vol.2, Addison-Wesley, p. 25, 1966.

[KNU81] Knuth D.E. "Seminumerical Algorithms", Addison-Wesley, p. 316-364, 1981.

[KRI85] Krishnamoorthy M.S., & Saunders B.D., "Hermite normal form for matrices of polynomials", 1985.

[KUM82] Kumar S.P., "Parallel algorithms for solving linear equations on MIMD computers", PhD. Thesis, Washington State University, 1982.

[LAM44] Lamé G., "Notes sur la limite du nombre de divisions dans la recherche du Plus Grand Commun Diviseur entre deux nombres entiers", Co. Re. Acad. Sci. Paris 19, 867-870, 1844.

[LKK83] Lord R.E., Kowalik J.S., Kumar S.P., "Solving Linear Algebraïc Equations on a MIMD Computer", J. ACM 30 (1), p. 103-117, 1983.

[LOV86] Lovasz L., "An Algorithmic Theory of Numbers, Graphs and Convexity", SIAM, CBMS-NSF, Regional Conference Series in Applied Mathematics, p. 1-31, 1986.

[MAR64] Marcus M., Ming H., "A survey of matrix theory and matrix inequalities", Allyn and Bacon Ed., p. 44-54, 1964.

[MAT71] Matyasevich Y., "On recursive unsolvability of Hilbert's tenth problem", Proc. Fourth Int. Cong.s, Bucharest 1971.

[NEW72] Newman M., "Integral Matrices", Pure and Applied Mathematics, Academic Press, 1972.

[OZE87] Ozello P., "Calcul exact des formes de Jordan et de Frobenius d'une matrice", thèse INPG, janvier 1987, Grenoble.

[PIP79] Pippenger N., "On simultaneous resource bounds", Proc. 20th IEEE Symp. on Foundations of Computer Society, Los Angeles, p. 307-311, 1979.

[ROC89] Roch J.L., "L'Architecture du Système PAC et son Arithmétique Rationnelle", Thèse INPG, Grenoble, 1989

[RSS87] Roch J.L., Sénéchaud P., Siebert-Roch F., Villard G., "PAC : Parallel Algebraic Computing", Rap. Rech. Imag Grenoble, RR-686 I,1987

[RSS89] Roch J.L., Sénéchaud P., Siebert-Roch F., Villard G., "Computer Algebra on MIMD Machine", SIGSAM Bull., vol. 23, n° 1, 1989

[RUZ81] Ruzzo W.L., "On Uniform Circuit Complexity", Journal of Computer and System Sciences, Vol. 22, p. 365-383, 1981.

[SAA85] Saad Y., "Communication Complexity of Gaussian Elimination Algorithm on Multiprocessors", RR 348, Computer Science Dpt., Yale University, 1985.

[SAA86] Saad Y., "Communication complexity of the Gaussian elimination on a multiprocessor", Lin. Alg. Appl. 77, 1986.

[SAM78] Sameh A.H., Kuck D.J., "On Stable Parallel Linear System Solvers", Journal of the Association for Computing Machinery", Vol. 25, N°1, p. 81-91, 1978.

[SCH85] Schrijver A.,"Theory of linear and integer programming", John Wiley, Chichester, England, 1985.

[TOU89] Tourancheau B., "Algorithmique Parallèle pour les machines à mémoire distribuée (applications aux algorithmes matriciels)", thèse INPG, Grenoble, 1989.

[VIL88] Villard G., "Calcul Formel et Parallélisme : Résolution de Systèmes Linéaires", Thèse INPG, Grenoble, décembre 1988.

[WAT85] Watt S.M., "Bounded Parallelism in Computer Algebra", Ph D Thesis in Computer Science, Waterloo, Ontario, 1985

[WIL55] Wilkinson J.H., "An Assessment of the System of Optimum Coding used in the Pilot Automatic Computing Engine at the National Physical Laboratory", Phil. Trans. Roy. Soc. Ser. A248 p.253, 1955.

INPG **département des études doctorales**

AUTORISATION de SOUTENANCE

VU les dispositions de l'Arrêté du 23 novembre 1988 relatif aux Etudes doctorales

VU les rapports de présentation de

- Monsieur TRYSTRAM Denis
- Monsieur DAVENPORT James

Madame SIEBERG épouse ROCH Françoise

est autorisé(e) à présenter une thèse en soutenance en vue de l'obtention du diplôme de DOCTEUR de L'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE, spécialité

"Mathématiques Appliquées"

Fait à Grenoble, le 19 Janvier 1990

Pour le Président de l'I.N.P.G.
et par délégation,
le Vice-Président
P. VENNEREAU

RESUME :

Cette thèse est consacrée à l'étude de la forme normale d'Hermite et à la conception d'algorithmes parallèles pour son calcul. Nous avons examiné deux cas particuliers : le cas entier et le cas polynomial.
Ce problème présente de manière intrinsèque une caractéristique du calcul formel : le grossissement des coefficients intermédiaires. Cette particularité en fait un exemple test pour évaluer la parallélisation en calcul formel.

Les différents algorithmes séquentiels sont présentés. Nous étudions et comparons leurs complexités.

Puis, une approche parallèle est considérée. Après la présentation des résultats théoriques de $\mathcal{NC}$-réductibilité du problème, nous abordons l'étude de la parallélisation sur modèles expérimentaux. Nous définissons différents algorithmes pour les modèles à mémoire partagée et distribuée.
Une implantation a été réalisée sur un hypercube FPS T40 (32 processeurs). Le calcul de la forme normale d'Hermite d'une matrice 192x192 à coefficients entiers a pu être effectué en 7 heures.

ABSTRACT :

This thesis is dedicated to the study of the Hermite normal form of matrices and to the parallelization of its computation. Both integral and polynomial cases have been examined.
This problem presents intrinsically a classical feature of computer algebra algorithms : the intermediate coefficients swell'. This particularity turns it into a test example to evaluate the gain brought by parallelization to computer algebra.

Different sequential algorithms are presented. Their complexities are studied.

Then, a parallel approach is considered. After the description of $\mathcal{NC}$-reductibility, the parallelization on experimental models is taken up. Different algorithms are developed, suited to distributed and shared memory models.
An implementation has been realized on an hypercube FPS T40 (32 processors). The Hermite normal form of an 192x192 integral matrix has been computed in 7 hours.

MOTS-CLES :

Calcul Formel, Parallélisme, Algèbre linéaire, Réseaux, Forme normale d'Hermite, Polynômes