Étude de la complexité de la décomposition orthogonale d'une matrice sur plusieurs modèles d'architectures parallèles

El Mostafa Daoudi

▶ **To cite this version:**

El Mostafa Daoudi. Étude de la complexité de la décomposition orthogonale d'une matrice sur plusieurs modèles d'architectures parallèles. Modélisation et simulation. Institut National Polytechnique de Grenoble - INPG, 1989. Français. NNT : . tel-00332433

HAL Id: tel-00332433

https://theses.hal.science/tel-00332433

Submitted on 21 Oct 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.

TU GOXL

THESE

présentée par

El Mostafa DAOUDI

pour obtenir le titre de DOCTEUR de

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

Spécialité
Mathématiques Appliquées

ETUDE DE LA COMPLEXITE DE LA DECOMPOSITION ORTHOGONALE D'UNE MATRICE SUR PLUSIEURS MODELES D'ARCHITECTURES PARALLELES

Date de soutenance: 12 Mai 1989

Composition du jury:

Président: **F. ROBERT**

Examinateurs: **M. COSNARD**
P. QUINTON
Y. ROBERT
M. TCHUENTE

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

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Président : Georges LESPINARD

Année 1988

Professeurs des Universités

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

Professeur Université des Sciences Sociales (Grenoble II)

BOLLIET Louis

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

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

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

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

Directeurs de recherche 2ème Classe

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

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

E.N.S.E.E.G
CHATILLON Christian
HAMMOU Abdelkader
MARTIN GARIN Régina
SARRAZIN Pierre
SIMON Jean-Paul

E.N.S.E.R.G
BOREL Joseph

E.N.S.I.E.G
DESCHIZEAUX Pierre
GLANGEAUD François
PERARD Jacques
REINISCH Raymond

E.N.S.H.G
ROWE Alain

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

E.F.P.
CHARUEL Robert

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

Laboratoires extérieurs

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

UNIVERSITE Joseph FOURIER (GRENOBLE I)

Président de l'Université :
M. PAYAN Jean Jacques

Année Universitaire 1987 - 1988

MEMBRES DU CORPS ENSEIGNANT DE SCIENCES ET DE GEOGRAPHIE

PROFESSEURS DE 1ère Classe

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

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

PROFESSEURS de 2ème Classe

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

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

MEMBRES DU CORPS ENSEIGNANT DE L' IUT 1

PROFESSEURS de 1ère Classe

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

PROFESSEURS de 2ème classe

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

PROFESSEURS DE PHARMACIE

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

MEMBRES DU CORPS ENSEIGNANT DE MEDECINE

PROFESSEURS CLASSE EXEPTIONNELLE ET 1ère CLASSE

AMBLARD Pierre	Dermatologie	C.H.R.G.
AMBROISE-THOMAS Pierre	Parasitologie	C.H.R.G.
BEAUDOING André	Pédiatrie-Puericulture	C.H.R.G.
BEZEZ Henri	Orthopédie-Traumatologie	Hopital SUD
BONNET Jean-Louis	Ophtalmologie	C.H.R.G.
BOUCHET Yves	Anatomie	Faculté La Merci
	Chirurgie Générale et Digestive	C.H.R.G.
BUTEL Jean	Orthopédie-Traumatologie	C.H.R.G.
CHAMBAZ Edmond	Biochimie	C.H.R.G.
CHAMPETIER Jean	Anatomie-Topographique et Appliquée	C.H.R.G.
CHARACHON Robert	O.R.L.	C.H.R.G.
COLOMB Maurice	Immunologie	Hopital sud
COUDERC Pierre	Anatomie-Pathologique	C.H.R.G.
DELORMAS Pierre	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.

A la mémoire de mon père,
A ma mère,
A mes frères et mes sœurs.

J'adresse toute ma gratitude à Michel COSNARD, Professeur à l'ENS Lyon, pour m'avoir guidé tout au long de ce travail. Je tiens à lui exprimer toute ma reconnaissance pour toute son aide qu'il a pu m'apporter, ses encouragements et sa compétence que j'ai beaucoup appréciés.

J'exprime toute ma reconnaissance à tous les membres du jury:

Monsieur François ROBERT, Professeur à l'INP de Grenoble, pour l'honneur qu'il me fait en présidant le jury de cette thèse. J'ai apprécié beaucoup sa compétence scientifique et sa chaleur humaine.

Monsieur Patrice QUINTON, Directeur de recherche à l'IRISA, Rennes, pour l'honneur qu'il me fait en se déplaçant pour participer à ce jury.

Monsieur Yves ROBERT, Professeur à l'ENS Lyon, pour ses encouragements et ses conseils. Je tiens à le remercier également d'avoir accepté de juger ce travail.

Maurice TCHUENTE, Professeur à l'Université de Yaoundé, pour l'honneur qu'il me fait en acceptant de juger ce travail.

Mes vifs remerciements vont à tous les membres de l'équipe d'Algorithmique parallèle et calcul formel pour leur gentillesse et leur sympathie, avec une mention particulière pour Bernard TOURANCHEAU.

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

INTRODUCTION ET RAPPEL

PREMIERE PARTIE

COMPLEXITE DE LA DECOMPOSITION ORTHOGONALE SUR UNE ARCHITECTURE A MEMOIRE PARTAGEE

Chapitre III

Chapitre IV

DEUXIEME PARTIE

COMPLEXITE DE LA DECOMPOSITION ORTHOGONALE SUR UNE ARCHITECTURE A MEMOIRE DISTRIBUEE

Chapitre V

Chapitre VI

Chapitre VII

INTRODUCTION ET RAPPEL

INTRODUCTION

Soit A une matrice rectangulaire de taille (m,n). La décomposition orthogonale de A (appelée aussi décomposition QR) a la forme QA=R où Q est une matrice carrée orthogonale de taille m qui verifie $QQ^t=Q^tQ=I$, I est la matrice identité, et R est une matrice triangulaire supérieure de taille (m,n). Elle est utilisée dans la résolution de nombreux problèmes d'algèbre linéaire: résolution des systèmes linéaires, Ax=b où b est un vecteur de taille m, calcul des valeurs propres...[GVL]. Les deux méthodes les plus utilisées pour calculer cette décomposition sont: la méthode de Householder [Hou] et la méthode de Givens [Giv] qui feront l'objet de notre étude dans cette thèse. Le choix entre ces deux méthodes dépend de l'architecture utilisée et de l'application dans laquelle elle sont employées. Pour une exécution sur un ordinateur monoprocesseur, la méthode de Givens a été écartée initialement en faveur de la méthode de Householder à cause de son coût trés élevé: $3n^2(m-n/3)$ opérations arithmétiques et $mn-n(n+1)/2$ racines carrées contre $2n^2(m-n/3)$ opérations arithmétiques et seulement n racines carrées. Avec la version sans racine carrée proposée par Gentleman [Gen] qui nécessite un coût comparable à celui de la méthode de Householder, la méthode de Givens a reconnu un regain d'intérêt.

Pour résoudre le système linéaire Ax=b, nous bordons A par le vecteur b et nous effectuons sur b les mêmes transformations que sur A pour transformer le système initial en un autre, triangulaire supérieur, qu'on résoud par des méthodes existantes [ED], [Hel], [LC], [Saa 85], [Sam 77]. Lorsque A est rectangulaire c'est une étape clé dans la résolution du problème des moindres carrées [LH], [GVL]. Dans cette thèse nous n'abordons que l'étape de décomposition.

Le développement des systèmes parallèles a conduit à réévaluer la plupart des algorithmes séquentiels en fonction de nouveaux critères. Les premières études consistent à explorer le parallélisme maximal inhérent dans chaque application. L'architecture considérée est donc à mémoire partagée composée d'un nombre non limité de processeurs. De nombreux travaux ont été consacrés à l'étude de complexité de la décomposition orthogonale sur une telle architecture; nous renvoyons le lecteur à [Hel76], [Sam77], [SK74], [SK78]. Par la suite, les études ont porté sur des architectures multiprocesseurs à mémoire partagée composée d'un nombre limité de processeurs [CDMR], [CMR], [CR], [LKK], [MC], [SK78], [BGH], [VL], [SVL], [CRB]. Plusieurs implémentations pour des réseaux systoliques ont été proposées dans la littérature voir [BBK], [CR86], [GK]. Avec le développement des architectures à mémoire distribuée plusieurs algorithmes parallèles ont été présentés dans [KAP], [Cos], [CDa], [CDT], [CP], [Dua], [PJV],

[RTo], [Sam82], [Sam85].

Dans cette thèse, nous étudions la complexité de la décomposition orthogonale sur deux modèles d'architectures parallèles: modèle à mémoire partagée et modèle à mémoire distribuée.

Le chapitre I est consacré a un rappel sur les architectures multiprocesseurs et sur les algorithmes séquentiels des méthodes de Givens et de Householder qui calculent la décomposition orthogonale et qui feront l'objet de notre étude. Les chapitres suivants sont réparties en deux parties suivant le modèle d'architecture utilisée.

Dans la première partie nous nous intéressons à la complexité de la décomposition de Givens en parallèle sur une architecture à mémoire partagée, pour la parallélisation de la méthode de Householder sur une telle architecture nous renvoyons à [MR], [RTr] et [Sam77].

Tout d'abord, nous supposons que l'unité est le temps nécessaire pour effectuer une rotation indépendamment de la longueur des vecteurs qui la composent. La parallélisation de la méthode de Givens consiste à affecter des rotations indépendantes à des processeurs différents. Chaque rotation utilise 2 lignes par conséquent au maximum $\lfloor m/2 \rfloor$ rotations peuvent être exécutées simultanément.

Les principaux résultats de complexité apparus sont obtenus pour $p=\lfloor m/2 \rfloor$ (parallélisme maximal). En particulier le problème de complexité en temps a été résolu [CR], [CMR] (dans le cas des matrices carrées $T_{opt}=2n-o(n)$). Trois principaux algorithmes ont été proposé dans la littérature: l'algorithme de Sameh et Kuck [SK78], les algoritmes de Fibonacci [MC] et l'algorithme Glouton [CR], [MC]. Une description detaillée de ces algorithmes est présentée dans le chapitre II. Nous introduisons également de nouveaux algorithmes et montrons d'autres résultats qui nous permettent de contrôler l'activité des processeurs.

Dans le cas $p<\lfloor m/2 \rfloor$, le problème d'optimalité est resté ouvert (construction est temps d'exécution de l'algorithme optimal). Le chapitre III résoud entièrement le problème dans le cas des matrices carrées. Nous montrons que le nombre minimum de processeurs qui permet de calculer la décomposition de Givens en temps optimal est égal à $p_{opt} = n/(2+\sqrt{2})+o(n)$. En suite nous montrons comment construire un algorithme asymptotiquement optimal pour p quelconque.

Nous passons ensuite à une analyse plus fine de la complexité. L'unité est prise égale au temps d'exécution d'une opération élémentaire (+,-,*,/) ou d'une évaluation d'une racine carrée. Nous supposons tout d'abord que le nombre de processeurs est non limité. Chaque processeur exécute une opération élémentaire.

Sameh et Kuck [SK78] ont proposé une version parallèle pour calculer la décomposition standard de Givens d'une matrice carrée qui est en $12n+o(n)$ en utilisant $3n^2/2+o(n^2)$ processeurs. Dans le chapitre V nous proposons une autre version parallèle qui est en $8n-o(n)$ en utilisant seulement $3n^2/4+o(n^2)$ processeurs. Ensuite nous limitons le nombre de prcesseurs à O(n). Chaque processeur exécute une rotation. Donc le temps d'exécution de la rotation dépend de la position de l'élément annulé. A l'aide du formalisme du graphe des tâches [CR87], [Kum] nous proposons un ordonnancement qui améliore les résultats proposés dans la littérature.

L'approche que nous adoptons dans la deuxième partie, consacrée au modèle d'architecture à mémoire distribuée, prend en compte les coûts de communication. Les processeurs travaillent en échangeant des messages. Par conséquent le temps d'exécution dépend de la position du processeur dans le réseau.

Dans [Sam85], A. Sameh propose plusieurs algorithmes pour résoudre des problèmes d'algèbre linéaire sur un anneau de processeurs. Certains des algorithmes que nous présentons sont inspirés par ces résultas. Nous commençons tout d'abord par une étude macroscopique pour montrer l'influence des coûts de communication sur les resultats de complexité. Nous supposons que chaque processeur, en une unité de temps, lit ses données, effectue son traitement local et transfère les résultats vers d'autres processeurs. En collaboration avec Cosnard et Tourancheau [CDT] nous avons montré que la complexité en temps, $T_{opt}=2n+o(n)$, obtenue pour les mémoires partagées n'a pas changé sur le réseau linéaire et sur l'anneau communiquant avec l'hôte par l'intermédiaire d'un seul processeur. En revanche p_{opt} passe de $n/(2+\sqrt{2})$ à $n/2$. Lorsque le nombre p de processeurs est inférieur à n/2 nous montrons comment obtenir une borne inférieure du temps optimal avec p processeurs. Le problème d'optimalité avec p processeurs (construction et temps d'exécution de l'algorithme optimal) reste ouvert.

Nous montrons ensuite l'importance des coûts de communication avec l'extérieur (hôte), en supposant que l'anneau communique avec l'hôte par deux processeurs, dans ce cas nous montrons que le nombre minimum de processeurs qui permet de calculer la décomposition de Givens en T_{opt} est égal à n/3. Dans la cas p fixé ($p<n/3$), nous montrons comment construire des algorithmes qui s'exécutent en temps minimum avec p processeurs.

Dans les chapitres VI et VII nous comparons différentes implémentations parallèles des méthodes de Householder et de Givens [CDa], [CP], [PJV] sur une architecture multiprocesseur à mémoire distribuée et présentons les résultats expérimentaux obtenus sur l'hypercube T20 de FPS installé au laboratoire TIM3 IMAG.

Notations:

$\lfloor a \rfloor$ est la partie entière inférieure de a: $\lfloor a \rfloor \leq a < \lfloor a \rfloor + 1$
$\lceil a \rceil$ est la partie entière supérieure de a: $\lceil a \rceil - 1 \leq a < \lceil a \rceil$
O(f(n))=g(n)*f(n) avec g tend vers une constante lorsque n tend vers l'infini
o(f(n))=g(n)*f(n) avec g(n) tend vers 0 lorsque n tend vers l'infinie
log(x) est le logarithme en base 2 de x

CHAPITRE I

RAPPEL SUR LE PARALLELISME ET SUR LA DECOMPOSITION ORTHOGONALE

I.1. RAPPEL SUR LE PARALLELISME

1.1. Introduction

L'évolution de l'architecture des ordinateurs est une conséquence du développement de la technologie des circuits intégrés VLSI (Very Large Scale Integration). Elle a permis la réalisation de nouveaux ordinateurs pour répondre à des besoins réels en puissance de calcul et pour résoudre de gros problèmes. Cette évolution a donné naissance à plusieurs types d'architectures [HB]: vectorielles (ou pipelinées), systoliques, data flow, multiprocesseurs à mémoire partagée et plus récemment, à mémoire distribuée.

Le but des architectures vectorielles est d'obtenir une grande puissance de traitement en divisant les unités de calcul en plusieurs étages séparés par des cellules de mémorisation, et en affectant à chaque étage, l'exécution d'une partie d'une opération. Par conséquent, l'exécution d'une nouvelle opération peut commencer avant que la précédente ne soit terminée. Théoriquement, une machine vectorielle avec une unité de calcul divisée en k étages est asymptotiquement k fois plus rapide qu'une machine séquentielle (pouvu qu'il n'y ait pas de blocage).

Le gain de temps dans les architectures multiprocesseurs est obtenu en multipliant le nombre de processeurs (chaque processeur peut être lui même un processeur vectoriel). La parallélisation dans ce type d'architectures consiste à diviser le programme en tâches indépendantes et à affecter ces tâches aux différents processeurs. Les processeurs fonctionnent soit en mode asynchrone (chaque processeur possède sa propre unité de contrôle) soit en mode synchrone (tous les processeurs sont dirigés par la même unité de contrôle). Quant à la mémoire, elle peut être globale (partagée) et dans ce cas tous les échanges entre processeurs sont réalisés par son intermédiaire. Elle peut être aussi distribuée, chaque processeur a alors sa propre mémoire (mémoire locale) et dans ce cas les communications se

font par l'intermédiaire d'un réseau d'interconnexion qui relie physiquement les processeurs entre eux. Pour plus de détail sur les autres types d'architectures nous renvoyons à [HB], [KL].

L'algorithmique n'a pas échappé à l'effet de cette évolution qui a conduit à réévaluer la plupart des algorithmes usuels en fonction de nouveaux critères. Donc la structure des algorithmes a beaucoup changé, elle dépend essentiellement de l'architecture sur laquelle ils doivent être implementés. Naturellement il est apparu de nouveaux langages pour permettre à l'utilisateur d'expliciter le parallélisme et pour gérer les communications. Exemples de langages parallèles: OCCAM [Wil], LESTAP [FH].

1.2. Classification des architectures parallèles

Il y a de nombreuses classifications des architectures muliprocesseurs [Ens], [Fen]. La plus connue est celle de Flynn [Flyn]. Elle est basée sur la multiplicité des flux de données et des flux d'instructions. On distingue notamment les deux structures suivantes:

SIMD (Single Instruction Multiple Data): Tous les processeurs reçoivent la même instruction (Single Instruction), envoyée par l'unité de contrôle, et l'exécutent sur leurs propres données (Multiple Data). L'exécution est dirigée par une unité de contôle commune à tous les processeurs et donc tous les processeurs exécutent la même instruction au même instant. On obtient un fonctionnement synchrone. Exemple de machines SIMD: Illiac IV, BSP, STARAN, MPP, DAP. On inclut aussi dans cette catégorie les réseaux cellulaires et systoliques [Kun].

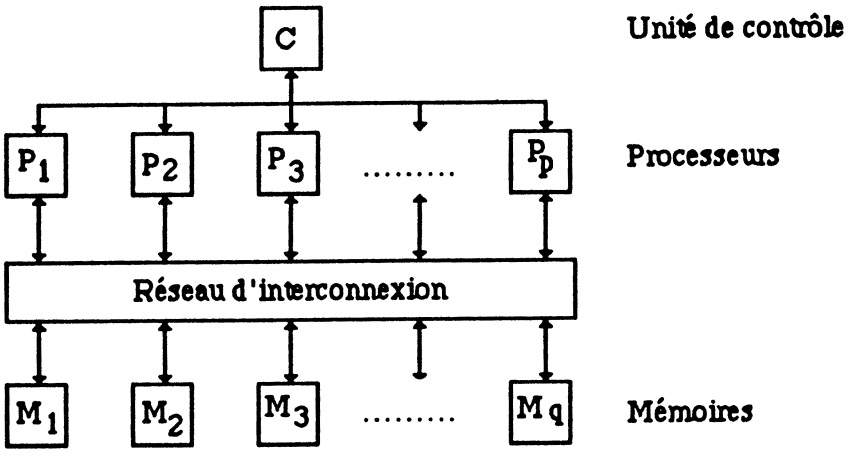


Figure 1: Modèle SIMD

MIMD (Multiple Instruction Multiple Data): Contrairement au cas précédent les processeurs exécutent des tâches différentes (Multiple Instruction) sur des données différentes (Multiple Data). Chaque processeur dispose d'une mémoire locale de taille petite ou grande suivant le type d'architecture(à mémoire partagée ou à mémoire distribuée), dispose de sa propre unité de contrôle lui permettant un fonctionnement totalement indépendant des autres processeurs. On obtient donc un fonctionnement asynchrone. Exemple de machines MIMD: IBM 3090, Cray X-MP et Y-MP, Hypercube series T de FPS et iPSC d'Intel.

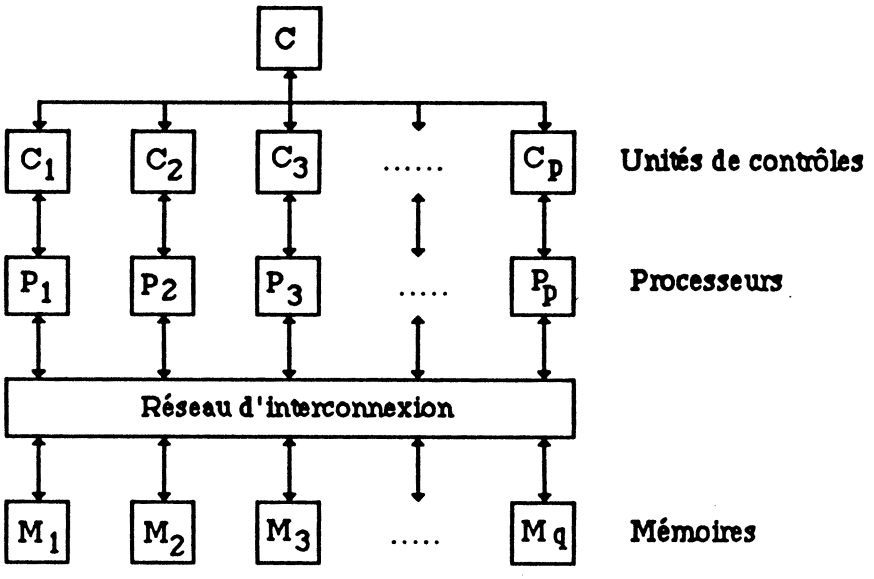


Figure 2: Modèle MIMD

1.3. Mesure du parallélisme.

Un moyen pour paralléliser des algorithmes est de détecter les tâches indépendantes puis de les affecter à des processeurs différents. Il s'agit donc de restructurer les algorithmes séquentiels existants pour les adapter aux architectures parallèles sur lesquelles ils seront exécutés.

Il est évident qu'un système utilisant p processeurs exécute au plus p fois plus vite le même algorithme qu'un système utilisant un seul processeur. En général, dans la pratique, cette borne n'est jamais atteinte (cela est dû à plusieurs caractéristiques propres au système ou à l'algorithme lui même: conflits d'accés à la mémoire, circulations des données, synchronisations dues aux communications, parties séquentielles des algorithmes).

Notations et définitions

Soient p le nombre de processeurs et T(p) le temps d'exécution d'un algorithme sur

un système utilisant p processeurs. On définit le facteur d'accélération S(p) qui permet de mesurer l'amélioration en temps de l'utilisation de p processeurs [Sch] par:

$$S(p) = \frac{T(1)}{T(p)} \text{ avec } S(p) \leq p$$

Nous introduisons aussi l'efficacité qui mesure le taux moyen d'utilisation des p processeurs par:

$$E(p) = \frac{S(p)}{p} \text{ avec } E(p) \leq 1$$

Soit T_{opt} le temps d'exécution minimum lorsque le nombre de processeurs est non limité. Un algorithme est optimal si son temps d'exécution est égal à T_{opt}, et asymptotiquement optimal si son temps d'exécution est de l'ordre de T_{opt} (lorsque la taille du problème est trés grande). On note p_{opt} le nombre minimum de processeurs qui permet d'exécuter l'algoritme en T_{opt}.

Lorsque le nombre de processeurs est limité à p, on désigne par $T_{opt}(p)$ le temps d'exécution minimum avec p processeurs, on a donc la relation:

$$T_{opt} \leq T_{opt}(p) \leq T(p)$$

Un algorithme est dit optimal avec p processeurs si son temps d'exécution est égal à $T_{opt}(p)$, et est dit asymptotiquement optimal avec p processeurs si son temps d'exécution est de l'ordre de $T_{opt}(p)$ lorsque la taille du problème est très grande.

1.4. Modèle à mémoire partagée

L'architecture de base des systèmes multiprocesseurs à mémoire partagée, comme le montre la figure 3, est composée de trois parties: une grande mémoire commune dans laquelle on stocke toutes les données, un ensemble de processeurs comportant chacun un peu de mémoire pour le stockage local, ce qui permet de diminuer les échanges et un réseau d'interconnexion qui joue un rôle primordial dans les échanges entre les processeurs et la mémoire.

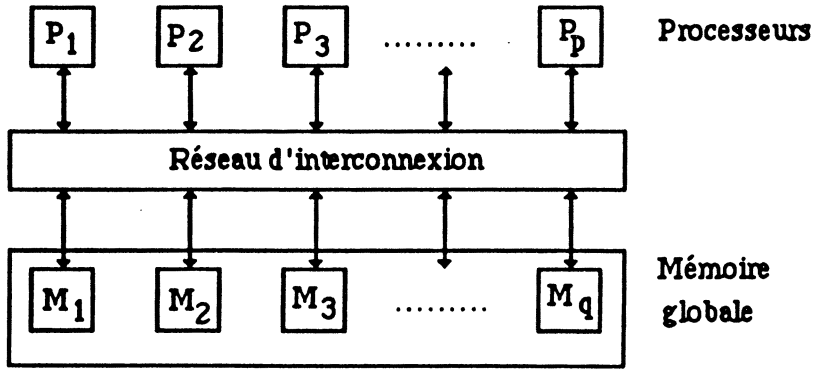


Figure 3: Architecture de base des machines à mémoire partagée

Les processeurs travaillent en partageant les données qui sont rangées dans la mémoire commune. Chaque processeur effectue les actions suivantes: il lit en mémoire les données dont il a besoin, effectue son traitement sur ses données et enfin écrit les résultats en mémoire.

1.4.1. Organisation

Chaque processeur ne peut accéder directement qu'à sa propre mémoire et communique avec les autres processeurs par l'intermédiaire de la mémoire partagée qui est accessible aux processeurs par l'intermédiaire d'un réseau d'interconnexion. Les problèmes architecturaux résident essentiellement dans les difficultés d'accès à cette mémoire. Pour améliorer le débit de la mémoire vers les processeurs, de nombreuses solutions ont été proposées: partage en plusieurs bancs mémoires pour permettre l'accès simultané des processeurs, hiérarchisation de la mémoire par l'introduction de caches, généralisation de l'emploi d'opérateurs "pipeline", conception de réseaux d'interconnexion performants pour faciliter les échanges et éviter les perturbations dues aux conflit d'accès à la mémoire ou à l'attente des communications. Le plus simple est de relier tous les processeurs par un bus en temps partagé. L'inconvénient de ce type de réseau est la limitation du nombre de processeurs que peut comporter le système. Le réseau le plus intéressant et le plus flexible est le crossbar qui assure un débit maximum entre les processeurs et la mémoire. La topologie de ce réseau a la forme d'une grille qui relie chaque processeur à chaque banc mémoire. Ce type de réseau limite le nombre de processeurs. En effet le nombre de croisements est trés grand et proportionnel au produit du nombre de processeurs par le nombre de bancs mémoires. D'autres solutions intermédiaires ont été développées pour la réalisation des réseaux d'interconnexion performants (pour plus d'informations nous renvoyons à [HB], [AB]).

1.4.2. Graphe des tâches

La méthodologie de conception d'algorithmes parallèles sur les architectures à mémoire partagée repose essentiellement sur des techniques d'ordonnancement [CDe], [Kum]. Dans ce paragraphe, nous faisons une présentation directement inspirée du travail de Cosnard et Robert [CR87]. On commence par partitionner le problème en tâches avec des contraintes de précédence entre elles puis on affecte les tâches aux processeurs en respectant ces contraintes. Pour cela, le programmeur doit disposer d'un langage spécialisé lui permettant de découper et de définir des relations de chronologie entre les tâches. Une autre approche qu'on appelle "programmation dynamique" a pour but d'épargner au programmeur ce travail de découpage et d'ordonnancement, en automatisant le processus d'allocation.

Une **tâche T** est une unité indivisible de traitement qui effectue deux types

d'actions: une purement locale consiste en l'exécution de la tâche dans le processeur et la deuxième consiste à communiquer avec les autres tâches à travers la mémoire. Elle est donc caractérisée uniquement par son comportement extérieur: entrées, sorties et par son temps d'exécution. Par conséquent on peut représenter la tâche T par un triplet (E_T,S_T,t_T)

E_T=Ensemble des entrées de la tâche T

S_T=Ensemble des sorties de la tâche T

t_T=Temps d'exécution de la tâche T

Le **temps d'exécution** comporte en général le temps de l'arithmétique et le temps de communication entre le processeur et la mémoire. Comme les communications se font par l'intermédiaire de la mémoire cela peut créer des conflits lorsque le nombre de tâches indépendantes est grand et laisser les processeurs inactifs. Par conséquent cela limite le nombre de processeurs que peut comporter une machine à mémoire partagée.

Un **algorithme séquentiel** est un ensemble A de tâches T_i muni d'une relation d'ordre total noté "$\rightarrow$" tel que $T_i \rightarrow T_j$ signifie que l'exécution de la tâche T_j doit être terminée avant que ne débute l'exécution de la tâche T_j.

$$A=(T_1,T_2,\ldots,T_n, \rightarrow)$$

Un **système de tâches** $S=(T_1,T_2,\ldots,T_n, <<)$ est un ensemble de tâches muni d'une relation d'ordre partiel noté <<: $T_i<<T_k$ $(i\neq k)$ signifie que l'exécution de la tâche T_i doit être terminée avant que l'exécution de la tâche T_k ne puisse commencer.

Soit $S=(T_1,T_2,\ldots,T_n, <<)$ un système de tâches. Deux tâches T_i et T_k sont **indépendantes** si elles ne modifient aucune variable commune.

Deux tâches T_i et T_k sont **consécutives** si et seulement si les sorties de l'une sont les entrées de l'autre c'est à dire $T_i<<T_k$ $(i\neq k)$ et il n'existe aucune autre tâche T_j telle que $T_i<<T_k<<T_j$.

Un système de tâches $S=(T_1,T_2,\ldots,T_n, <<)$ est un **système de précédence** 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

Le **graphe de précédence** G (ou graphe de tâches) associé à un système de précédence $S=(T_1,T_2,\ldots,T_n, <<)$ est défini de la façon suivante:
- Les sommets du graphe sont les tâches de S
- T_i et T_j sont reliées par une arête si et seulement si elles sont consécutives.

Un **ordonnancement** compatible avec un système de précédence $S=(T_1,T_2,\ldots,T_n, <<)$ est une application définie par:

$$\text{Ord: } \{T_1,T_2,\ldots,T_n, <<\} \longrightarrow \{1,2,\ldots,p\} \times N$$
$$T_k \longrightarrow (\text{prc}(T_k), \text{tps}(T_k))$$

où $\text{prc}(T_k)$ est le processeur qui exécute T_k et $\text{tps}(T_k)$ est la date du début de l'exécution de T_k.

telle que:

(i) $T_i << T_k$ implique $\text{tps}(T_i)+t_i \leq \text{tps}(T_k)$ (t_i est le temps d'exécution de la tâche T_i). Ceci signifie que si T_i et T_k ne sont pas indépendantes l'exécution de T_k ne pourra débuter qu'à l'issue de l'exécution de T_i.

(ii) $\text{prc}(T_i)=\text{prc}(T_k)$ implique $\text{tps}(T_i)+t_i \leq \text{tps}(T_k)$ ou $\text{tps}(T_k)+t_k \leq \text{tps}(T_i)$ ceci signifie qu'un processeur ne sera pas affecté en même temps à deux tâches différentes.

Un **algorithme parallèle** associé à un algorithme séquentiel $A=(T_1,\ldots,T_n, \rightarrow)$ est un couple (S,Ord) formé par le système de précédence S associé à A et par un ordonnancement Ord compatible avec S. Le temps d'un algorithme sera donc le temps de fin d'exécution de la dernière tâche exécutée.

Pour une description complète de cette méthodologie nous renvoyons à [CR87].

1.5. Modèle à mémoire distribuée

Le point sensible des architectures multiprocesseurs à mémoire partagée est le réseau d'interconnexion. Les difficultés d'accès à la mémoire limitent le nombre de processeurs: au delà de quelques dizaines, les performances du réseau d'interconnexion se dégradent. Pour obtenir des machines à parallélisme massif, on a donc recours à des architectures où la mémoire est décentralisée et les connexions limitées: chaque processeur dispose d'une mémoire locale à accès rapide et n'est connecté qu'à un certain nombre de processeurs voisins. L'architecture de base des machines à mémoire distribuée, comme le montre la figure 4, est composée de

deux parties: Un ensemble de processeurs auxquels on associe à chacun une mémoire privée suffisamment grande et un réseau de connexion qui relie physiquement les processeurs entre eux.

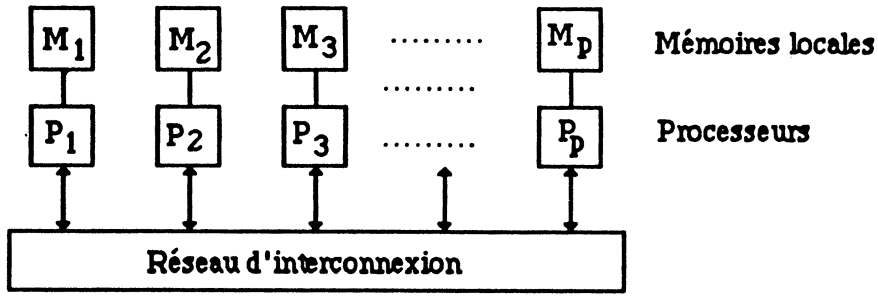


Figure 4: Architecture de base des modèles à mémoire distribuée

Les processeurs fonctionnent en mode asynchrone (type MIMD). La seule synchronisation est due au protocole de communication envoyer/recevoir. Le processeur P_i peut envoyer un message au processeur P_j si et seulement si P_i et P_j sont voisins (reliés par un canal de communication) et que P_j est prêt à lire. De nombreux prototypes ont vu le jours et les réseaux systoliques en sont directement inspirés [Kun]. Par contre ce n'est que récemment que sont apparus les premiers supercalculateurs construits suivant ce principe. Leur programmation est encore malaisée et, en dehors des algorithmes systoliques [Quin], [DI], il n'existe pas de méthodologie de conception d'algorithmes parallèles pour ces machines.

1.5.1. Hypercube: un exemple d'architecture à mémoire distribuée

Plusieurs types d'architectures à mémoire distribuée ont été développées et diffèrent suivant la façon de relier les processeurs entre eux. Il se peut qu'un processeur ait besoin de communiquer avec tous les autres. La solution qui apparait parfaite est celle qui relie chaque processeur à tous les autres. L'inconvénient c'est le nombre de connexions entre processeurs qui rend impossible la réalisation d'un tel réseau pour un grand nombre de processeurs.

Une topologie assez satisfaisante est appelée hypercube est présentée sur la figure 5. Elle présente plusieurs propriétés [SS], [Sei] elle permet d'avoir un grand nombre de processeurs en utilisant seulement un nombre relativement petit de connexions par processeurs (le nombre de connexions croit d'une façon logarithmique avec le nombre p de processeurs). Concernant les communications la topologie hypercube assure la rapidité des échanges entre deux noeuds quelconques en $O(\log_2(p))$. A partir de ce principe de connexions, les societés Intel et FPS ont réalisé des machines hypercubes à mémoire distribuée, iPSC d'Intel et séries T de FPS .

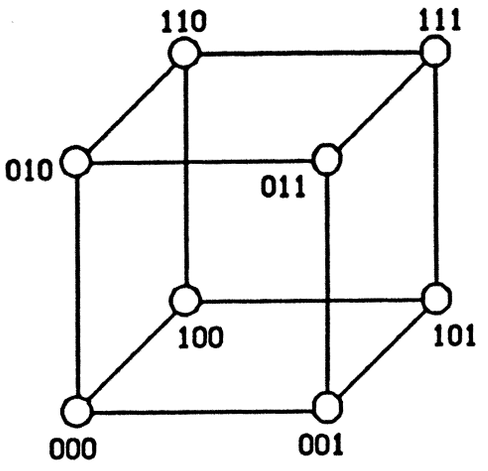


Figure 5: Un hypercube de dimension 3

Un hypercube de dimension n (appelé aussi n-cube) comporte 2^n processeurs (noeuds) numérotés de 0 a 2^n-1 comme le montre la figure 5. Deux processeurs sont connectés si et seulement si leur numéro en représentation binaire diffère d'un seul bit. Par conséquent chaque noeud a exactement n voisins. L'hypercube est défini par récurrence: un hypercube d'ordre 0 est un seul processeur, un hypercube d'ordre (n+1) est obtenu à partir de deux hypercubes d'ordre n chacun, en reliant 2 à 2 les processeurs de même rang dans les 2 hypercubes. Par conséquent pour doubler le nombre de processeurs il suffit de rajouter une seule connexion par processeur. Par un simple réarrangement des positions des processeurs on peut immerger facilement d'autres topologies dans l'hypercube [SS]: anneau, réseau linéaire, arbres,....

1.5.2. Modèle d'évaluation

Nous supposons que les processeurs sont identiques et qu'ils travaillent en échangeant des méssages. Chaque processeur lit sur un ou plusieurs canaux de communication les données dont il a besoin en provenance d'autres processeurs, effectue son traitement, puis transfère les résultats en direction des processeurs qui les demandent. Dans ce cas le temps d'exécution ne dépend plus uniquement du temps d'exécution de chaque tâche, mais aussi de la place dans le réseau, du processeur qui exécute cette tâche.

Notations et définitions

Le temps nécessaire pour envoyer/recevoir n données consécutives est défini dans [Saad85] par $(n\tau_c+\beta_c)$ où β_c est le temps d'initialisation et τ_c est le temps de transfert d'une donnée.

Nous supposons que chaque processeur possède une unité vectorielle avec un additionneur/soustracteur et un multiplieur/diviseur. Les deux unités vectorielles peuvent être enchainées. Nous supposons que le temps pour réaliser une opération vectorielle entre deux vecteurs de longueur n chacun est égal à $(\beta_a+n\tau_a)$ où β_a est le temps d'initialisation et τ_a est le temps du cycle de l'unité vectorielle.

Evaluation de l'accélération

L'évaluation du gain qu'apporte l'utilisation d'un système multiprocesseur composé de p pocesseurs est donnée par le facteur d'accélération $S(p) = T(1)/T(p)$ où $T(p)$, $1\leq p$, est le temps d'exécution d'un algorithme utilisant p processeurs. Traditionnellement pour calculer ce facteur, on fixe la taille du problème et on fait varier le nombre de processeurs. Cependant pour les architectures à mémoire distribuée le problème est différent. En effet, comme la mémoire est distribuée entre les processeurs, la taille du problème augmente avec le nombre de processeurs. Par conséquent, faute de place mémoire, on ne peut plus exécuter ce nouveau problème sur un seul processeur.

Récemment Gustafson [Gus] a proposé un nouveau moyen pour calculer l'accélération sur les architectures à mémoire distribuée, en fixant le nombre de processeurs et non la taille du problème [Amd], de la façon suivante: On calcule le temps moyen d'une opération arithmétique $\tau_{max}(p)$, pour $1\leq p$, que nécessite l'algorithme exécuté dans la plus large taille. L'accélération de Gustafson est égale à $\tau_{max}(1)/\tau_{max}(p)$.

I.2. RAPPEL SUR LA DECOMPOSITION QR

Soit A une matrice rectangulaire de taille (m,n). La décomposition orthogonale (appelée aussi décomposition QR) permet de réduire A en une matrice triangulaire supérieure par une transformation orthogonale. Cette décomposition a la forme $QA=R$ où Q est une matrice carrée orthogonale de taille m, i.e $QQ^t=Q^tQ=I$, et R est une matrice triangulaire supérieure de taille (m,n). C'est une méthode très importante qui intervient dans la solution de nombreux problèmes d'analyse numérique. L'avantage d'utiliser une telle décomposition réside dans sa stabilité numérique. Quant au coût de la méthode, il est généralement plus élevé par rapport à la méthode de Gauss, par exemple. Elle est utilisée pour la résolution de systèmes linéaires, particulièrement pour la résolution des problèmes des moindres carrés lorsque le nombre d'équations est trés grand par rapport au nombre d'inconnues. Elle intervient aussi dans les problèmes de calcul des valeurs propres (algorithme QR).

Plusieurs méthodes de calcul de la décomposition QR ont été proposées dans la littérature. Les plus utilisées sont celles qui permettent d'obtenir Q par des multiplications successives de matrices simples et orthogonales. Ces matrices servant a prémultiplier A pour annuler les éléments situés sous la diagonale principale de A pour obtenir R. Par conséquent le calcul explicite de Q n'est pas nécessaire. Parmi ces méthodes; la méthode de Givens, et la méthode de Householder [GVL].

Dans le cas de la résolution du système linéaire: (1) $Ax=b$, où A est une matrice de rang n et b est un vecteur de taille m, on borde la matrice A par le vecteur b puis on effectue sur b les mêmes transformations que sur A pour transformer le système (1) en un système triangulaire supérieur: (2) $Rx=b_r$ où $b_r=Qb$. Deux cas sont possibles:

(i) Si A est carrée alors on résoud directement le système triangulaire (2) par la méthode de la remontée (back-substitution).

(ii) Si A est rectangulaire de taille (m,n), on transforme le système (1) en un autre problème équivalent qui consiste à trouver une solution approchée de la solution exacte au sens de la norme euclidienne c'est à dire trouver x_m tel que:
$\|Ax_m-b\|_2=\min\|Ax-b\|_2$, $\|.\|_2$ est la norme euclidienne, c'est le problème des moindres carrées. Puisque la matrice Q est orthogonale alors $\|Q\|_2=1$. Par conséquent $\|Ax-b\|^2_2=\|QAx-Qb\|^2_2=\|Rx-b_r\|^2_2$ avec $R=(R_1,0)^t$ et $b_r=(c,d)^t$ où c et d sont deux vecteurs de taille n et (m-n) respectivement et R_1 est triangulaire supérieure de taille n. On trouve donc $\|Ax-b\|^2_2=\|Rx-b_r\|^2_2 = \|R_1x-c\|^2_2 + \|d\|^2_2$

Il est clair que, si rang(A)=rang(R_1)=n alors le minimum est atteint par x_m, solution unique de R_1x_m-c=0. Dans ce cas on trouve $\|Ax_m-b\|^2_2=\|d\|^2_2$, $\|d\|_2$ est l'écart par rapport la norme euclidienne entre la solution exacte de (1) et la solution approchée réalisée par x_m.

Pour en savoir plus sur l'utilisation de la décomposition QR dans d'autres problèmes d'analyse numérique nous renvoyons à [GVL], [HL]

Dans les deux paragraphes suivants nous présentons deux méthodes pour calculer la décomposition orthogonale; la méthode de Givens et la méthode de Householder.

2.1. Méthode de Givens

La méthode de Givens permet de calculer la décomposition orthogonale en utilisant des rotations planes [Giv], [GVL].

2.1.a. Méthode standard de Givens (avec racine carrée)

Définition 1: Dans IR^m x IR^m, une rotation d'angle θ dans le plan (i,j) est définie par une matrice orthogonale de taille n:

$$\Omega_{i,j} = \begin{array}{c} \\ j \\ \\ i \\ \\ \end{array} \begin{array}{c} \begin{array}{ccc} j & & i \end{array} \\ \begin{bmatrix} I & 0 & 0 \\ 0 & c_j & s_j \\ 0 & I & 0 \\ 0 & -s_j & c_j \\ 0 & 0 & I \end{bmatrix} \end{array} \quad \text{avec } c_j=\cos\theta \text{ et } s_j=\sin\theta$$

Soit R(i,j,k), i≠j, 1≤i,j≤m et 1≤k≤n, la rotation dans le plan (i,j) pour annuler l'élément en position (i,k). R(i,j,k) est appelée rotation de Givens. Elle est déterminée par la matrice de rotation $P^k_{i,j}= \Omega_{i,j}$ et par le produit $P^k_{i,j}A$. Supposons que les éléments $a_{i,q}$ et $a_{j,q}$ pour 1≤q<k sont tous nuls alors R(i,j,k) est définie par:

$$t=\sqrt{a^2_{j,k} + a^2_{i,k}}$$

$$c_j=\frac{a_{j,k}}{t} \quad , \quad s_j=\frac{a_{i,k}}{t}$$

$$a_{j,k}=t$$

Pour h=k+1 **à** n **faire**

$$a_{j,h} = c_j a_{j,h} + s_j a_{i,h}$$

$$a_{i,h} = -s_j a_{j,h} + c_j a_{i,h}$$

fpour

R(i,j,k) ne modifie que les lignes i et j de A. Elle nécessite 4(n-k)+2 multiplications, 2(n-k)+1 additions, 2 divisions et une évaluation d'une racine carrée.

La décomposition QR de A par la méthode de Givens est obtenue en $mn-\frac{n(n+1)}{2}$ étapes. Chaque étape consiste à annuler un élément sous la diagonale principale en multipliant la matrice A à gauche par une rotation plane. Généralement on annule par colonne et de bas en haut. Soit Q_k le produit de (m-k) rotations planes pour annuler les éléments de la colonne k qui sont situés sous la diagonale principale. Il est clair que Q_k est orthogonale (produit de matrices orthogonales). Par conséquent la matrice Q est obtenue par la relation:

(1) $$Q=Q_n\, Q_{n-1}\cdots Q_2\, Q_1$$

Et R est obtenue par la relation de récurrence

(2) $$A=A_1,\ldots,A_{k+1}=Q_kA_k,\ldots,R=Q_nA_n$$

Le choix des rotations n'est pas unique. Par conséquent plusieurs algorithmes séquentiels sont possibles. Choisissons par exemple, la rotation R(i,k,k) pour annuler l'élément en position (i,k). Dans ce cas la décomposition de Givens est accomplie par l'algorithme suivant:

```
Pour k=1 à n faire
    Pour i=m à k+1 faire
        R(i,k,k)
    fpour
fpour
```

Le nombre d'opérations arithmétiques que nécessite la méthode de Givens est asymptotiquement égal à $2n^2(m-\frac{n}{3})$ multiplications, $n^2(m-\frac{n}{3})$ additions, $2mn-n^2$ divisions et $mn-\frac{n^2}{2}$ racines carrées donc: $T(1)= 3n^2(m-\frac{n}{3})+o(mn^2)$ avec $mn-\frac{n^2}{2}+o(n^2)$ racines carrées.

2.1.b. Méthode de Givens sans racine carrée

Gentleman [Gen73, 75] a proposé une version de la méthode de Givens, appelée "Fast Givens transformation", qui réduit à la moitié le nombre de multiplications de la méthode standard et qui, de plus, évite les racines carrées. Cette méthode utilise les rotations planes pour calculer une matrice diagonale D, une matrice triangulaire supérieure R et une matrice orthogonale Q tel que $QA=D^{1/2}R$.

Supposons que $A=D^{1/2}B$ (initialement on prend D=I et B=A) alors on a:

$$j: 0,\ldots\ldots,0, \sqrt{d_j}\, b_{j,k}, \sqrt{d_j}\, b_{j,k}, \ldots\ldots \sqrt{d_j}\, b_{j,n}$$
$$i: 0,\ldots\ldots,0, \sqrt{d_i}\, b_{i,k}, \sqrt{d_i}\, b_{i,k}, \ldots\ldots \sqrt{d_i}\, b_{i,n}$$

Soit la rotation R(i,j,k) pour annuler l'élément $a_{i,k} = \sqrt{d_i}\, b_{i,k}$ on trouve donc aprés modification des lignes i et j par la rotation:

$$j: 0,\ldots\ldots,0, \sqrt{d_j}\, b_{j,k}, \sqrt{d_j}\, b_{j,k+1}, \ldots\ldots \sqrt{d_j}\, b_{j,n}$$
$$i: 0,\ldots\ldots,0, \quad 0 \quad , \sqrt{d_i}\, b_{i,k+1}, \ldots\ldots \sqrt{d_i}\, b_{i,n}$$

Avec:

$$\alpha=\frac{b_{i,k}}{b_{j,k}} \quad , \quad \beta=\frac{d_i}{d_j} * \alpha \quad , \quad \gamma=1+\alpha\beta$$

$$d_i=\frac{d_i}{\gamma}$$

$$d_j=\frac{d_j}{\gamma}$$

$b_{j,k}=b_{j,k}*\gamma$

Pour h=k+1 **à** n **faire**

$b_{j,h} = b_{j,h} + \beta b_{i,h}$

$b_{i,h} = b_{i,h} - \alpha b_{j,h}$

fpour

R(i,j,k) nécessite 2(n-k)+1 multiplications, 2(n-k)+1 additions et 4 divisions.

Le nombre d'opérations arithmétiques que nécessite la méthode de Givens sans racine carrée est asymptotiquement égal à $n^2(m - \frac{n}{3})$ multiplications, $n^2(m - \frac{n}{3})$ additions et $4mn-2n^2$ divisions donc: $T(1)= 2n^2(m - \frac{n}{3}) + o(mn^2)$

2.2. Méthode de Householder

La méthode de Householder [Hou], [GVL] permet de calculer la décomposition orthogonale de A par des multiplications successives à gauche de A par des transformations de la formes $H=I-2uu^t$ où u est un vecteur unitaire, $u^tu=1$, de IR^m. La matrice H est symétrique et orthogonale. Elle est appelée transformation de Householder.

Proposition 1: Pour tout vecteur a, non nul, élément de IR^m, il existe un vecteur

unitaire u élément de IR^m qui définit une matrice H de la forme $H=I-2uu^t$, tel que Ha soit colinéaire à e_1 (e_1 est la première colonne de l'identité).

Preuve: voir [Hou], [GVL]: On trouve $u=\frac{1}{\mu}(a-\|a\|_2 e_1)$ avec $\mu^2=2\|a\|_2(\|a\|_2-y_1)$, a_1 est la première composante du vecteur a et $Ha=\|a\|_2 e_1$ (ou $Ha=-\|a\|_2 e_1$, pour les critères de choix dans la pratique entre $\|a\|_2$ et $-\|a\|_2$ nous renvoyons à [GVL])

Remarque 1: La matrice H est complètement déterminée par le vecteur u. En effet pour tout x on a:

$$Hx=x-2(uu^tx)=x-2(u^tx)u$$

Remarque 2: Pour éviter de calculer μ, qui demande l'évaluation d'une racine carrée, on pose $v=\mu u$. Dans ce cas pour tout x on a:

$$Hx=x-2(uu^tx)=x-2\frac{v^tx}{\mu^2}v$$

La décomposition QR par la méthode de Householder d'une matrice rectangulaire de taille (m,n) est obtenue en n étapes. Chaque étape k consiste à appliquer la transformation de Householder, pour annuler les (m-k) éléments de la colonne k qui sont situés sous la diagonale, de la façon suivante:

(i) On détermine la transformation de Householder P_k qui permet d'annuler les (m-k) dernières composantes de la colonne k donc on a:

$$P_k(a_{k,k}, a_{k+1,k},\ldots, a_{n,k})^t=(r_{k,k},0,\ldots\ldots, 0)$$

(ii) On forme la matrice $H_k=diag(I_k,P_k)$ puis on effectue le produit H_kA_k.

La matrice Q est donc obtenue par la relation:

(3) $$Q=H_n H_{n-1}\ldots H_2 H_1$$

La matrice R est obtenue à partir de la relation de récurrence

(4) $$A=A_1,\ldots,A_{k+1}=H_kA_k,\ldots,R=H_nA_n$$

La décomposition de Householder est accomplie par l'algorithme séquentiel suivant:

Pour k=1 à n **faire**

$a_k=(a_{k,k}, a_{k+1,k},\ldots\ldots, a_{n,k})^t$

$b=\|a_k\|_2$

$v=a_k-be_1$

$n(v)=2b(b-a_{k,k})$

$n(v)=1/n(v)$

Pour j=k+1 à n **faire**

$s=2n(v)\langle v,a_j\rangle$ * $a_j=(a_{k,j}, a_{k+1,j},\ldots\ldots, a_{n,j})^t$

Pour i=k à m **faire**

$a_{i,j} = a_{i,j} - sv_i$

fpour

fpour

fpour

Le nombre d'opérations que nécessite la méthode de Householder est asymptotiquement égal à $n^2(m - \frac{n}{3})$ multiplications, $n^2(m- \frac{n}{3})$ additions, n divisions et n racines carrées. Donc le coût de la méthode est égal $T(1)= 2n^2(m- \frac{n}{3})+o(mn^2)$.

PREMIERE PARTIE

COMPLEXITE DE LA DECOMPOSITION ORTHOGONALE SUR UNE ARCHITECTURE A MEMOIRE PARTAGEE

INTRODUCTION ET NOTATIONS

1. INTRODUCTION

Nous étudions la complexité de la décomposition QR d'une matrice dense rectangulaire de taille (m,n) par la méthode de Givens [GVL] sur une architecture multiprocesseur à mémoire partagée.

Dans les chapitres II et III nous supposons que chaque processeur effectue trois types d'actions en une unité de temps, qu'on note "pas": lecture de deux lignes de la mémoire, calcul et application d'une rotation pour annuler un élément et enfin écriture des deux nouvelles lignes en mémoire. Le temps d'exécution de l'algorithme sera donc mesuré en nombre de pas. Sur une architecture monoprocesseur la méthode de Givens nécessite donc $T(1)=mn- n(n+1)/2$ pas.

Les deux lignes qui composent la rotation sont toutes les deux modifiées et de plus ce sont les seules modifiées par cette rotation. Donc au plus $\lfloor m/2 \rfloor$ rotations peuvent être exécutées simultanément. Il est inutile par conséquent de disposer de plus de $\lfloor m/2 \rfloor$ processeurs. Le principe de la parallélisation de la méthode de Givens consiste à affecter des rotations indépendantes à des processeurs différents, sous la condition que tout élément déjà annulé sera conservé. Cette hypothèse n'est pas restrictive: en effet dans [CMR] on montre que le fait d'annuler des éléments qui seront détruits ultérieurement n'améliore pas le temps d'exécution. De plus l'ordre d'annulation de bas en haut et de gauche à droite ne modifie pas les resultats de complexité. Nous étudions le temps minimum (complexité temporelle) de ce problème dans le cas où le nombre de processeurs p est quelconque (p inférieur à $\lfloor m/2 \rfloor$) et lorsque $p=\lfloor m/2 \rfloor$ (parallélisme maximal). Dans le chapitre II nous présentons les résultats obtenus par divers auteurs dans le cas où $p=\lfloor m/2 \rfloor$. Le chapitre III résoud entièrement le problème, pour des matrices carrées, dans le cas $p<\lfloor m/2 \rfloor$.

Dans le chapitre IV nous supposons que le temps d'exécution de la rotation dépend de la longueur des deux lignes qui la composent. L'unité qu'on note "flops" sera donc le temps d'exécution d'une opération élémentaire (+,-,*,/) ou d'une racine carrée. Deux cas seront étudiés. Dans le premier cas nous supposons que le nombre de processeurs est non limité, dans le deuxième cas nous bornons le nombre de processeurs par un facteur linéaire, $p=O(n)$, pour calculer la décomposition de Givens d'une matrice carrée de taille n.

2. NOTATIONS ET DEFINITIONS

* r(i,t) est le nombre d'éléments qu'on annule au pas t dans la colonne i.

* s(i,t) est le nombre d'éléments nuls dans la colonne i après le pas t.
  $s(i,t)=r(i,1)+r(i,2)+\cdots+r(i,t)$ avec $s(i,0)=0$ et $s(0,t)=m$.

* On définit l'**écart** au temps t entre les colonnes i et i-1 par $e(i,t)=s(i-1,t)-s(i,t)$. On dit qu'au temps t, la colonne i est d'écart e(i,t).

* La colonne i est dite **active** au temps t si et seulement si on peut annuler au moins un élément dans cette colonne, c'est à dire si $e(i,t)\geq 2$. Elle est dite **saturée** si et seulement si $s(i,t)=n-i$ et elle est dite **nulle** si et seulement si tous les éléments sur cette colonne sont nuls.

* Un élément en position (i,j), annulé au pas k, sera noté en cette position par l'entier k sur les figures qui illustrent les algorithmes.

Soient p le nombre de processeurs et T(p) le temps d'exécution avec p processeurs. On définit les quantités suivantes:

* ac(t) est le nombre de processeurs actifs au temps t (le nombre d'éléments annulés durant le t-ème pas).

* in(t) est le nombre de processeurs inactifs au temps t.

On a donc les relations:

(1) $$in(t)=p-ac(t)$$

(2) $$\sum_{t=1}^{T(p)} in(t) = \sum_{t=1}^{T(p)} p - \sum_{t=1}^{T(p)} ac(t)$$

$\sum_{t=1}^{T(p)} ac(t)$ représentant le nombre d'éléments annulés donc égal à $mn - \frac{n(n+1)}{2}$

Notons par: $$P_{ac} = \sum_{t=1}^{T(p)} ac(t) = \text{surface d'exécution}$$

et par $$P_{in} = \sum_{t=1}^{T(p)} in(t) = \text{surface du repos}$$

Alors, d'après la relation (2) on a la propriété suivante [LKK]:

Propriété 1: $pT(p) = P_{in} + P_{ac}$

CHAPITRE II

COMPLEXITE DE LA DECOMPOSITION DE GIVENS EN PARALLELE CAS $p=\lfloor \frac{m}{2} \rfloor$

Dans ce chapitre nous supposons que $p=\lfloor m/2 \rfloor$ processeurs sont disponibles. Nous présentons les résultats de complexité obtenus dans [SK78], [MC], [CR], [CMR], [CR83], [CR86] avec une description des algorithmes parallèles introduits dans la littérature: algorithme de Sameh et Kuck [SK78], algorithme glouton [MC], [CR] et les algorithmes de Fibonacci [MC]. Nous proposons de nouveaux algorithmes que nous appelons algorithmes de Fibonacci modifié. Dans le paragraphe II.4 nous présentons les résultats d'optimalité.

II.1. ALGORITHME DE SAMEH ET KUCK

Cet algorithme a été proposé par Sameh et Kuck [SK78], il annule de bas en haut et de gauche à droite. Le principe de l'algorithme consiste à annuler l'élément en position (i,k) (k<i) dès que les éléments en positions (i,k-1) et (i-1,k-1) sont nuls en utilisant la rotation R(i,i-1,k). On commence par éliminer l'élément en position (m,1), au pas suivant on annule l'élément en position (m-1,1). Mais au pas 3 on peut annuler simultanément les deux éléments en positions (m,2) et (m-2,1). En effet la rotation R(m,m-1,2) utilisée pour annuler l'élément en position (m-2,1) ne détruit pas les zéros introduits en positions (m,1) et (m-1,1). En général au pas r on annule tous les éléments en positions (i,k) tel que $m+2k-i-1=r$ avec $1 \leq i \leq m, 1 \leq k \leq \min(i-1,n)$ en utilisant les rotations R(i,i-1,k). Dans cet algorithme on distingue deux phases:

* **Première phase:** Quand l'élimination progresse dans la première colonne, le nombre d'éléments annulés augmente de 1 tous les 2 pas. Par conséquent le nombre de processeurs inactifs diminue de 1 tous les deux pas. Le temps d'exécution de cette phase est celui nécessaire pour annuler les éléments de la première colonne jusqu à l'élément en position (3,1), il est donc égal à (m-2). La figure 1.a illustre l'ordre d'élimination de cette phase dans le cas où m=12 et n=5.

* **Deuxième phase:** Elle débute au temps (m-1). Quand on annule les éléments situés dans la sous diagonale, le nombre d'éléments à annuler diminue de 1 tous les deux pas. Par conséquent le nombre de processeurs libérés augmente de 1 tous les deux pas. Le temps d'exécution de cette phase est celui nécessaire pour annuler les éléments dans la sous diagonale donc égal à: n-1 si n=m et à n si m > n. Cette phase est illustrée sur la figure 1.b dans le cas où m=12 et n=5. Dans ce cas, elle commence au temps 11.

*				
+	*			
10	+	*		
9	+	+	*	
8	10	+	+	*
7	9	+	+	+
6	8	10	+	+
5	7	9	+	+
4	6	8	10	+
3	5	7	9	+
2	4	6	8	10
1	3	5	7	9

1.a: Phase 1

*				
11	*			
10	12	*		
9	11	13	*	
8	10	12	14	*
7	9	11	13	15
6	8	10	12	14
5	7	9	11	13
4	6	8	10	12
3	5	7	9	11
2	4	6	8	10
1	3	5	7	9

1.b: Phase 2 (à partir du pas 11)

Figure 1: Les deux phases de l'algorithme de Sameh et Kuck

Le temps d'exécution de l'algorithme de Sameh et Kuck est la somme des temps d'exécution des deux phases. Le nombre de processeurs qui permet d'exécuter cet algorithme est égal à min(n,$\lfloor \frac{m}{2} \rfloor$). Le nombre maximum d'éléments annulés est atteint au temps (m-1).

Théorème 1 [SK78]: Si SK(m/2) est le nombre total de pas pour calculer la décomposition de Givens d'une matrice rectangulaire de taille (m,n), avec m≥n, par l'algorithme de Sameh et Kuck alors:

SK(m/2)=2n-3 si n=m
=m+n-2 si m>n.

II.2. ALGORITHMES DE FIBONACCI

2.1. Algorithmes de Fibonacci d'ordre i

Ces algorithmes ont été proposés par Modi et Clarke [MC]. Ils annulent de bas en haut et de gauche à droite. Le schéma d'élimination des éléments est construit en

deux phases. Avant de décrire ces algorithmes dans le cas **général**, nous présentons tout d'abord l'algorithme de Fibonacci d'ordre 1.

On définit la suite d'entiers $(u_k^1)_{k\in N}$ par: $u_k^1 = u_{k-1}^1 + 1 = k$ pour $k\in N^*$ et $u_0^1 = 0$

Soit $S_k^1 = u_k^1 + u_{k-1}^1 + \ldots + u_1^1 = \frac{k(k+1)}{2}$ pour $k\in N^*$

Première phase: Elle consiste à construire un tableau A^1, triangulaire inférieur, de la façon suivante: on remplit la première colonne de haut en bas: en position (2,1) on met $u_1^1 = 1$ copies de zéro, au dessous on met $u_2^1 = 2$ copies de (-1), par récurrence supposons que la première colonne soit remplie jusqu'à la position $(S_{k-1}^1, 1)$, k>1, au dessous on met u_k^1 copies de -(k-1). La deuxième colonne est obtenue à partir de la première colonne en ajoutant 2 à tous les éléments de la première colonne, et en décalant vers le bas d'une position. Par récurrence la colonne j est obtenue à partir de la colonne (j-1) en ajoutant 2 à tous les éléments de la colonne (j-1), et en décalant vers le bas d'une position comme le montre la figure 2.a dans le cas où m=12 et n=5.

Deuxième phase: Elle donne les pas auquels les éléments sont annulés en ajoutant u=(k+1) à tous les éléments du tableau A^1 où k est tel que $S_k^1 < m-1 \leq S_{k+1}^1$ comme le montre la figure 2.b dans le cas où m=12 et n=5. Dans ce cas on a u=5.

*						*				
0	*					5	*			
-1	2	*				4	7	*		
-1	1	4	*		On ajoute u=5	4	6	9	*	
-2	1	3	6	*	⟶	3	6	8	11	*
-2	0	3	5	8		3	5	8	10	13
-2	0	2	5	7		3	5	7	10	12
-3	0	2	4	7		2	5	7	9	12
-3	-1	2	4	6		2	4	7	9	11
-3	-1	1	4	6		2	4	6	9	11
-3	-1	1	3	6		2	4	6	8	11
-4	-1	1	3	5		1	4	6	8	10

2.a: Phase 1 **2.b**: Phase 2

Figure 2: Les deux phases de définition de l'algorithme de Fibonacci d'ordre 1

Cas général: On définit la suite d'entiers $(u_k^i)_{k\in N}$ par:

$u_k^i = u_{k-1}^i + u_{k-2}^i + \ldots + u_{k-i}^i + 1$ pour $k\in N^*$ et $u_j^i = 0$, pour $j\leq 0$.

Soit $S_k^i = u_k^i + u_{k-1}^i + \ldots + u_1^i$

L'algorithme de Fibonacci d'ordre i est obtenu de la même façon que l'algorithme de Fibonacci d'ordre 1 en remplaçant u_k^1 par u_k^i dans la première phase, pour remplir la première colonne du tableau A^i. Par récurrence la colonne j est obtenue en ajoutant (i+1) à tous les éléments de la colonne (j-1), et en décalant vers le bas d'une position. Quant à la deuxième phase, elle consiste à ajouter u=(k+1) à tous les éléments du tableau A^i où k est tel que $S_k^i < m-1 \leq S_{k+1}^i$.

La figure 3 illustre les deux phases de la construction de l'algorithme de Fibonacci d'ordre 2 pour m=12 et n=5. Dans ce cas, on a u=4.

*						*				
0	*					4	*			
-1	3	*				3	7	*		
-1	2	6	*			3	6	10	*	
-2	2	5	9	*	On ajoute u=4	2	6	9	13	*
-2	1	5	8	12	⟶	2	5	9	12	16
-2	1	4	8	11		2	5	8	12	15
-2	1	4	7	11		2	5	8	11	15
-3	1	4	7	10		1	5	8	11	14
-3	0	4	7	10		1	4	8	11	14
-3	0	3	7	10		1	4	7	11	14
-3	0	3	6	10		1	4	7	10	14

3.a: Phase1 **3.b**: Phase 2

Figure 3: Les deux phases de l'algorithme de Fibonacci d'ordre 2

Pour montrer que les algorithmes construits précédemment, permettent d'annuler les éléments de cette manière, il suffit de montrer q'au temps t on a:

$$s(j,t)-s(j-1,t)=\sum_{\delta=1}^{t}\left[\, r(j-1,\delta)-r(j,\delta)\,\right] \geq 2r(j,t+1)$$

Ce qui nous permet d'annuler, au temps (t+1), r(j,t+1) éléments dans la colonne j.
D'après la construction de l'algorithme on a: $r(j,\delta)=r(j-1,\delta-(i+1))$. Par conséquent $s(j,t)-s(j-1,t) \geq 1+r(j-1,t-i)+r(j-1,t-i+1)+\ldots+r(j-1,t)$ pour $t \geq (i+1)$.

Or $1+r(j-1,t-i+1)+\ldots+r(j-1,t)=1+r(j,t+2)+\ldots+r(j,t+(i+1))=r(j,t+1)$ (construction de l'algorithme).
On déduit que $s(j,t)-s(j-1,t) \geq 2r(j,t+1)$.

Théorème 2 [MC]: Si Fi(m/2) est le nombre total de pas pour calculer la décomposition de Givens d'une matrice rectangulaire de taille (m,n) par l'algorithme de Fibonacci d'ordre i alors asymptotiquement :

* $Fi(m/2)=\lfloor \sqrt{2m} \rfloor + 2(n-1)$ pour i=1
* $Fi(m/2)=\log_{a(i)}m +(i+1)(n-1)$ pour i≥2 où a(i) est la plus grande racine positive de l'équation $x^i=x^{i-1}+...+x+1$.

Démonstration:

Soit t_j le temps nécessaire pour éliminer les éléments des j premières colonnes (le pas au cours duquel on annule l'élément en position (j+1,j)). D'après la construction de l'algorithme de Fibonacci d'ordre i on a la relation:

(1) $$t_{j+1}=t_j+(i+1)$$

Le temps d'exécution est celui nécessaire pour annuler toutes les colonnes donc égal à:

(2) $$\begin{cases} t_n & \text{si } n<m \\ t_{n-1} & \text{si } n=m \end{cases}$$

D'après les relations (1) et (2) on a: $Fi(m/2)=t_1+(i+1)(n-1)$ pour $n < m$

$=t_1+(i+1)(n-2)$ pour $n = m$

Avec $t_1=u=(k+1)$ où k est tel que $S^i_k < m-1 \leq S^i_{k+1}$ [MC]. Asymptotiquement on a:

Si i=1 alors $S^1_k = \frac{k(k+1)}{2}$ et donc $t_1=\sqrt{2m}$

Si i≠1 on a $t_1=\log_{a(i)}m$

a(i) est la plus grande racine positive de l'équation $x^i=x^{i-1}+...+x+1$. Par exemple pour i=2 on trouve $a(i)=\frac{1+\sqrt{5}}{2}$. En général $a(2)\leq a(3)\leq...\rightarrow 2$

Dans ces algorithmes, on ne connait pas le nombre d'éléments qu'on annule à chaque pas et par conséquent on ne connait pas le nombre de processeurs actifs. Le résultat suivant nous permet de contrôler, à chaque pas, l'activité des processeurs pour l'algorithme de Fibonacci d'ordre 1.

Lemme 1: Si on utilise l'algorithme de Fibonacci d'ordre 1 pour calculer la décomposition de Givens d'une matrice rectangulaire de taille $(m,\alpha m)$, avec $0<\alpha\leq 1$, alors asymptotiquement, le nombre d'éléments à annuler simultanément diminue de 1 tous les 4 pas. Par conséquent le nombre de processeurs inactifs augmente de 1 tous les 4 pas.

Démonstration

On pose $n=\alpha m$, $0<\alpha\leq 1$. La démonstration est divisée en deux parties. La première partie consiste à montrer que $P_{in}\geq \frac{n^2}{2} + o(n^2)$. Dans la deuxième partie nous montrons que $P_{in} = \frac{n^2}{2} + o(n^2)$

Première partie:

On remplace i par 1 dans la relation (1) (algorithme de Fibonacci d'ordre 1) on obtient:

(3) $$t_{j+1}=t_j + 2$$

D'autre part comme pour annuler un élément on utilise deux lignes alors au plus $\lfloor \frac{m-j+1}{2} \rfloor$ éléments peuvent être annulés en même temps que l'élément en position (j+1,j) donc:

(4) $$ac(t_j) \leq \lfloor \frac{m-j+1}{2} \rfloor \text{ et } in(t_j) \geq \lfloor \frac{m}{2} \rfloor - \lfloor \frac{m-j+1}{2} \rfloor = m_{in}(t_j)$$

Donc au moins $m_{in}(t_j)$ processeurs sont inactifs au temps t_j.

A partir des relations (3) et (4) on déduit que:

(5) $$t_{j+2}=t_j +4 \text{ et } in(t_{j+2}) \geq \lfloor \frac{m}{2} \rfloor - \lfloor \frac{m-j+1}{2} \rfloor +1 = \min(t_j)+1$$

Donc au temps $t_{j+2}=t_j +4$, le nombre $m_{in}(t_j)$, qui minore le nombre de processeurs inactifs au temps t_j, a augmenté de1. Par conséquent, à partir de t_1, au moins un processeur se libère tous les 4 pas, c'est à dire

$in(t)\geq 0$ pour $t_1\leq t<t_1+4=t_3$

$in(t)\geq 1$ pour $t_3\leq t<t_3+4=t_5$

...

$in(t)\geq k$ pour $t_{2k+1}\leq t<t_{2k+1}+4=t_{2(k+1)+1}$

...

On déduit donc que $P_{in}= \sum_{t=1}^{2n+o(n))} in(t) \geq \frac{n^2}{2} + o(n^2)$

Deuxième partie

On applique la propriété 1 on obtient $\frac{m}{2} Fi(m/2)=P_{ac}+P_{in}$

avec $Fi(m/2) = \sqrt{2m} + 2n$ et $P_{ac}=mn - \frac{n(n+1)}{2}$

On déduit donc que $P_{in} = \frac{n^2}{2} + \frac{m}{2}\sqrt{2m} + o(n^2)$

Comme $n=\alpha m$, avec $0<\alpha\leq 1$, on conclut que $P_{in}=\frac{n^2}{2} + o(n^2)$

2.2. Algorithmes de Fibonacci modifié d'ordre i

Dans ce paragraphe nous introduisons des algorithmes basés sur le même principe que les algorithmes de Fibonacci. Le but est de construire un schéma d'élimination des éléments en deux phases. Dans la **première phase** nous éliminons uniquement sur les $\min(n,\lfloor \frac{m}{2} \rfloor -1)$ premières colonnes de telle façon qu'à la fin de cette phase l'écart des $\lfloor \frac{m}{2} \rfloor$ premières colonnes soit égal à 2. Dans la **deuxième phase** nous éliminons avec le même principe que la deuxième phase de l'algorithme de Sameh et Kuck.

Description des deux phases

* **Première phase:** Elle est construite en 2 étapes.

Dans <u>la première étape</u> on remplit la première colonne selon le même principe que les algorithmes de Fibonacci d'ordre i mais en commençant à partir de la position (3,1) (on veut que l'écart de la colonne 1 soit égal à 2). La colonne j est obtenue à partir de la colonne (j-1) en ajoutant (i+1) à tous les éléments de la colonne (j-1) en décalant vers le bas de deux positions (l'écart de la colonne j soit égal à 2).

Dans <u>la deuxième étape</u> on augmente tous les éléments de u=(k+1) où k est tel que $S^i_k < m-2 \leq S^i_{k+1}$. Si $T_1(m/2)$ est le temps d'exécution de la première phase alors asymptotiquement on a:

1) i = 1

$$\begin{cases} T_1(m/2) = \sqrt{2m-2} + 2\left(\frac{m}{2} - 2\right) & \text{si } n>\lfloor \frac{m}{2} \rfloor \\ T_1(m/2) = \sqrt{2m-2} + 2(n-1) & \text{si } n\leq\lfloor \frac{m}{2} \rfloor \end{cases}$$

2) $i \neq 1$

$$\begin{cases} T_1(m/2) = \log_{a(i)}(m) + (i+1)\left(\frac{m}{2} - 2\right) & \text{si } n>\lfloor \frac{m}{2} \rfloor \\ T_1(m/2) = \log_{a(i)}(m) + (i+1)(n-1) & \text{si } n\leq\lfloor \frac{m}{2} \rfloor \end{cases}$$

où a(i) est la plus grande racine positive de l'équation $x^i=x^{i-1}+\ldots+x+1$.

En effet:

Soit f_j le temps nécessaire à l'élimination des éléments des j premières colonnes (la date à laquelle l'élément en position (j+2,j) est annulé). D'après la construction de cette phase on a la relation:

(6) $$f_{j+1}=f_j+(i+1)$$

D'autre part on sait que le temps d'exécution de la première phase est celui pour exécuter les $\min(n,\lfloor \frac{m}{2} \rfloor -1)$ premières colonnes donc égal à:

$$\begin{cases} f_{m/2-1} & \text{si } n > \lfloor m/2 \rfloor \\ f_n & \text{si } n \leq \lfloor m/2 \rfloor \end{cases}$$

Donc d'aprés la relation (6) on a:

$$\begin{cases} T_1(m/2) = f_1 + (i+1)\left(\frac{m}{2} - 2\right) & \text{si } n > \lfloor \frac{m}{2} \rfloor \\ T_1(m/2) = f_1 + (i+1)(n-1) & \text{si } n \leq \lfloor \frac{m}{2} \rfloor \end{cases}$$

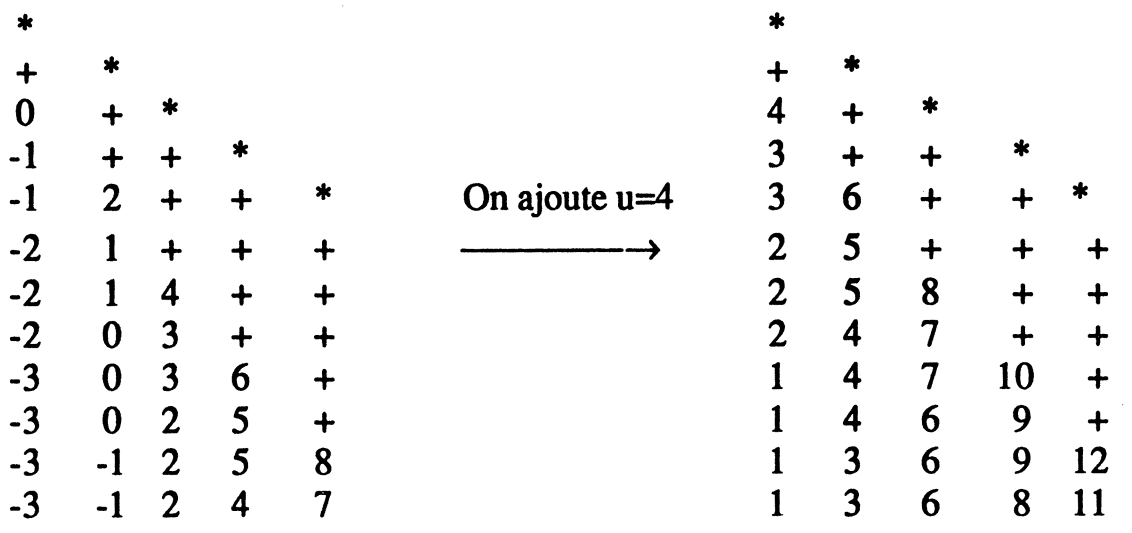
Il suffit donc de calculer une valeur asymptotique de f_1 qui est obtenue de la même façon que dans le théorème 2 [MC].

Si i=1 on a $f_1=\sqrt{2m-2}$.

Si i≠1 on a $f_1=\log_{a(i)}m$ où a(i) est la plus grande racine positive de l'équation

$$x^i=x^{i-1}+\ldots+x+1$$ [MC].

La figure 4 illustre les deux étapes de la première phase de la construction de l'algorithme de Fibonacci modifié d'ordre 1 pour m=12 et n=5. Dans ce cas on a $f_1=u=4$ et $T_1(m/2)=12$.



4.a: Etape 1 **4.b:** Etape 2

Figure 4: La première phase de l'algorithme de Fibonacci modifié d'ordre 1

* **Deuxième phase:** On procède comme dans la deuxième phase de l'algorithme de Sameh et Kuck en commençant au temps $T_1(m/2)+1$ (au temps 13 sur la figure 5). Donc le temps d'exécution $T_2(m/2)$ de cette phase est celui pour annuler tous les éléments situés dans la sous diagonale principale donc égal à:

$T_2(m/2)=n-1$ si $n=m$

$= n$ si $m>n$

La figure 5 illustre la phase 2 dans le cas où m=12 et n=5.

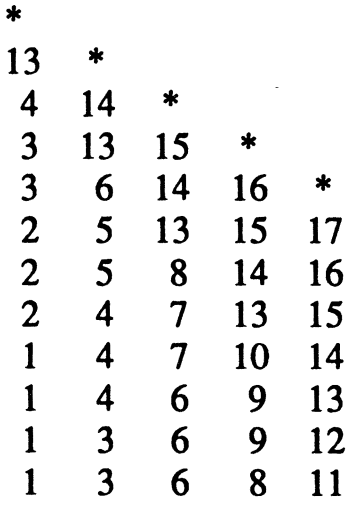


Figure 5: Phase 2 de l'algorithme de Fibonacci modifié d'ordre 1 (à partir du pas 13)

Théorème 3: Si Fim(m/2) est le nombre total de pas pour calculer la décomposition de Givens d'une matrice rectangulaire de taille (m,n) par l'algorithme de Fibonacci modifié d'ordre i alors asymptotiquement on a:

* Si $n > \lfloor \frac{m}{2} \rfloor$ alors Fim(m/2)=$\sqrt{2m-2}$+m+n-4 pour i=1

$=\log_{a(i)}m + (i+1)\frac{m}{2} + n - 2(i+1)$ pour i≠1

* Si $n \leq \lfloor \frac{m}{2} \rfloor$ alors Fim(m/2)=$\sqrt{2m-2}$+3(n-1) pour i=1

$=\log_{a(i)}m + (i+2)(n-1)$ pour i≠1

où a(i) est la plus grande racine positive de l'équation $x^i=x^{i-1}+\ldots+x+1$.

Démonstration:

Le temps d'exécution de l'algorithme est la somme des temps d'exécution des deux phases donc égal à $T_1(m/2)+T_2(m/2)$

Remarques 1: Pour des matrices carrées on a:

* Le temps d'exécution de l'algorithme de Fibonacci d'ordre 1 est asymptotiquement égal au temps d'exécution de l'algorithme de Fibonacci modifié d'ordre 1.

* Pour i≠1, le temps d'exécution de l'algorithme de Fibonacci modifié d'ordre i est meilleur que celui de Fibonacci d'ordre i.

Le resultat suivant nous permet de contrôler l'activité des processeurs dans la première phase de l'algorithme de Fibonacci modifié d'ordre 1.

Lemme 2: Dans la première phase de l'algorithme de Fibonacci modifié d'ordre 1 pour calculer la décomposition de Givens d'une matrice rectangulaire de taille (m,αm) avec 0<α≤1, asymptotiquement, le mombre d'éléments à annuler simultanément diminue de 1 tous les 2 pas. Par conséquent le nombre de processeurs inactifs augmente de 1 tous les 2 pas.

Démonstration

On pose n=αm, 0<α≤1. Le principe de la démonstration est le même que celui du lemme 1.

Au plus $\lfloor \frac{m-2(j-1)}{2} \rfloor$ éléments peuvent être annulés au temps f_j (relation (6)). Donc

(7) $$ac(f_j) \leq \lfloor \frac{m-2(j-1)}{2} \rfloor \text{ et } in(f_j) \geq \lfloor \frac{m}{2} \rfloor - \lfloor \frac{m-2(j-1)}{2} \rfloor = m_{in}(f_j)$$

Donc au temps f_j , au moins $m_{in}(f_j)$ processeurs sont inactifs.

D'aprés les relations (6) et (7) on déduit que:

(8) $$f_{j+1}=f_j+2 \text{ et } in(f_{j+1}) \geq \lfloor \frac{m}{2} \rfloor - \lfloor \frac{m-2(j-1)}{2} \rfloor + 1 = m_{in}(f_{j+1}) = m_{in}(f_j)+1$$

Donc au temps $f_{j+1}=f_j+2$ le nombre $m_{in}(f_j)$, qui minore le nombre de processeurs inactifs au temps f_j, a augmenté de 1. Par conséquent, à partir de f_1, au moins un processeurs se libère tous les 2 pas.

Le reste de la démonstration est le même que dans le lemme 1. Il suffit de remplacer $F_i(m/2)$ par $T_1(m/2)$ (le temps d'exécution de la première phase de l'algorithme de Fibonacci modifié d'ordre 1).

II.3. ALGORITHME GLOUTON

L'algorithme glouton a été introduit indépendamment par Cosnard et Robert [CR] et par Modi et Clarke [MC]. Le principe de cet algorithme consiste à annuler à chaque pas le maximum d'éléments possibles. Les éléments d'une même colonne sont annulés de bas en haut et dans chaque ligne de gauche à droite. De plus tous les zéros déjà introduits sont conservés. Il est défini par:

$$r(i,t)=\lfloor \frac{s(i-1,t-1) - s(i,t-1)}{2} \rfloor$$

L'algorithme est illustré sur la figure 6 dans le cas où m=12 et n=5

*				
4	*			
3	6	*		
2	5	8	*	
2	4	7	10	*
2	4	6	9	12
1	3	6	8	11
1	3	5	7	10
1	3	5	7	9
1	2	4	6	8
1	2	4	6	8
1	2	3	5	7

Figure 6: Algorithme glouton

Ci dessous nous résumons les résultats de complexité de l'algorithme glouton obtenus dans [CR], [CMR], [MC]

Théorème 4: Si G(m/2) le nombre de pas pour calculer la décomposition de Givens d'une matrice rectangulaire de taille (m,n) par l'algorithme glouton alors:
1- G(m/2) = 2n-o(n) si m=n
2- G(m/2)=$\log_2 m+(n-1)\log_2(\log_2 m)$ +o(n) si n est fixe et m tend vers l'infini
3- G(m/2)=2n+o(n) si m=o(n^2)

II.4. RESULTATS D'OPTIMALITE

Soit T_{opt} le nombre minimum de pas pour calculer la décomposition de Givens d'une matrice rectangulaire de taille (m,n). Un algorithme est dit optimal si son temps d'exécution est égal à T_{opt} est dit asymptotiquement optimal si son temps d'exécution est asymptotiquement égal à T_{opt}.

Théorème 5 [CMR]**:** Quels que soient m et n, l'algorithme glouton est optimal pour calculer la décomposition de Givens.

Remarque 2: L'algorithme glouton n'est pas le seul algorithme optimal comme le montre l'exemple de la figure 7 dans le cas où m=12 et n=6.

*					
4	*				
3	6	*			
2	5	8	*		
2	4	7	10	*	
2	4	6	9	12	*
1	3	6	8	11	14
1	3	5	7	10	13
1	3	5	7	9	12
1	2	4	6	8	11
1	2	4	6	8	10
1	2	3	5	7	9

7.a: L'agorithme glouton

*					
9	*				
3	10	*			
2	9	11	*		
2	4	10	12	*	
2	4	9	11	13	*
1	3	6	10	12	14
1	3	5	9	11	13
1	3	5	7	10	12
1	2	4	6	9	11
1	2	4	6	8	10
1	2	3	5	7	9

7.b: Un algorithme optimal

Figure 7: Deux algorithmes optimaux

En utilisant les résultats des théorèmes 4 et 5 on a:

Théorème 6:

1- $T_{opt} = 2n-o(n)$ si $m=n$

2- $T_{opt} = \log_2 m + o(\log_2 m)$ si n est fixe et m tend vers l'infini

3- $T_{opt} = 2n+o(n)$ si $m=o(n^2)$

4- $2n+o(n) \leq T_{opt} \leq 3n+o(n)$ si $m=o(n^k)$ $k \geq 2$

Théorème 7:

1. Si m=n alors l'algorithme de Sameh et Kuck est asymptotiquement optimal.
2. Si m=bn, $b \geq 1$, alors l'algorithme de Fibonacci d'ordre 1 est asymptotiquement optimal.
3. Si $m=o(n^2)$, alors l'algorithme de Fibonacci d'ordre 1 est asymptotiquement optimal.
4. Si $m=o(n^k)$, $k>2$, alors l'algorithme de Fibonacci d'ordre 2 a une efficacité asymptotique égal à 0.666....

Dans la pratique, le théorème 7 nous permet de sélectionner les meilleurs algorithmes présentés suivant les valeurs de n. Pour des matrices carrées, il est préférable d'implémenter l'algorithme de Sameh et Kuck. En effet l'algorithme de Sameh et Kuck n'est pas seulement asymptotiquement optimal mais demande moins de contrôle (pour annuler en position (i,k) en utilise la rotation R(i,i-1,k)). Lorsque m et n sont proportionnels, l'algorithme de Fibonacci d'ordre 1 est asymptotiquement optimal. Son temps d'exécution est égal à 2n comparé à celui de Sameh et Kuck qui est égal à m+n-2. Si $m>n^2$, les algorithmes de Sameh et Kuck et de Fibonacci d'ordre 1 ne sont pas optimaux, par contre l'algorithme glouton est optimal mais on ne connait pas son temps d'exécution. Dans ce cas la meilleure borne (connue) du temps d'exécution est obtenue par l'algorithme de Fibonacci d'ordre 2. Elle est asymptotiquement égale à 3n.

CHAPITRE III

COMPLEXITE DE LA DECOMPOSITION DE GIVENS EN PARALLELE CAS $p < \lfloor \frac{m}{2} \rfloor$

III.1. INTRODUCTION

Le cas étudié précédemment est peu réaliste car, dans la pratique, les systèmes qu'on résoud sont très grands alors que le nombre de processeurs est relativement petit. A titre d'exemple l'ordinateur IBM 3090 possède 6 processeurs. L'objet de ce chapitre est donc d'étudier la complexité de la décomposition de Givens en parallèle lorsque le nombre p de processeurs est inférieur à $\lfloor m/2 \rfloor$ ($p<\lfloor m/2 \rfloor$).

Dans ce cas le problème d'optimalité est resté ouvert (construction et temps d'exécution de l'algorithme optimal). Dans le paragraphe III.2 nous présentons quelques algorithmes parallèles à p processeurs: algorithme de Sameh et Kuck et algorithmes glouton. Dans le paragraphe III.3 nous présentons des résultats de complexité dans le cas des matrices carrées de taille n: nous calculons tout d'abord le nombre minimum de pas $B_{inf}(p)$, et nous en déduisons que le nombre minimun de processeurs p_{opt} qui permet de calculer la décomposition de Givens en T_{opt}, vérifie la relation: $\frac{n}{2+\sqrt{2}} + o(n) \leq p_{opt} \leq \frac{n}{2}$

Nous montrons comment construire un algorithme asymptotiquement optimal pour $p=\frac{n}{2+\sqrt{2}}$. Nous en déduisons que $p_{opt}=\frac{n}{2+\sqrt{2}} + o(n)$. Enfin nous construisons un algorithme asymptotiquement optimal pour n et p quelconques.

Le paragraphe III.4 a pour objectif de construire des algorithmes asymptotiquement optimaux pour $p \leq \frac{n}{4}$ et pour $p \geq \frac{n}{3}$. Ceci nous permet de sélectionner les meilleurs algorithmes suivant la valeur de p.

III.2. QUELQUES ALGORITHMES A p PROCESSEURS

2.1 Parallélisation de l'algorithme de Sameh et Kuck avec p processeurs.

Soit A une matrice carrée de taille n, on suppose que n=pq. Le principe de cet algorithme consiste à:

a. Décomposer la matrice A en blocs A_k, pour $0 \leq k \leq q-2$, chacun est composé de p colonnes consécutives à partir de la colonne kp+1 et des (m-kp) dernières lignes sauf le bloc A_{q-2} qui est composé des 2p dernières colonnes et des 2p dernières lignes comme le montre la figure 1.

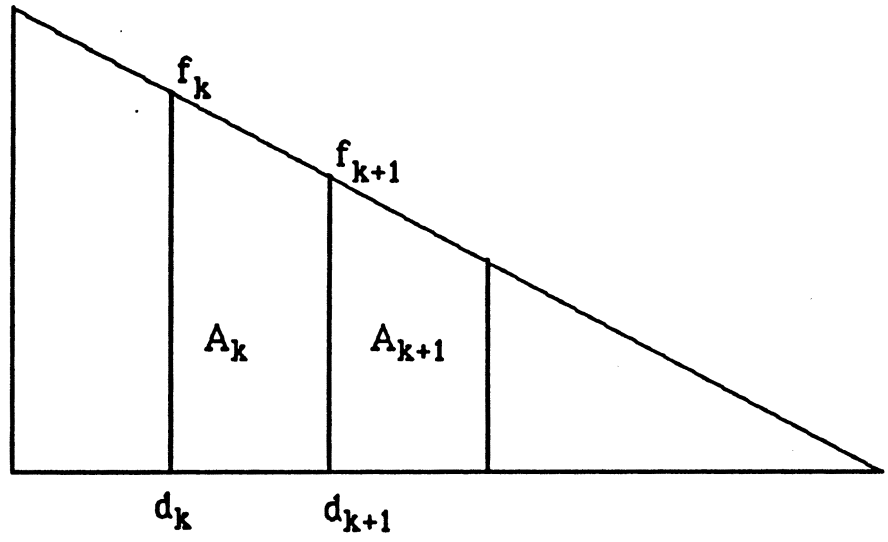


Figure 1: Décomposition en blocs A_k

b. Appliquer l'algorithme de Sameh et Kuck aux blocs A_k de la façon suivante:

1. On commence par éliminer dans le bloc A_0

2. On élimine dans le bloc A_{k+1} à partir de la fin de traitement de la première colonne de A_k comme le montre l'exemple illustré dans la figure 2 dans le cas où n=12 et p=3.

*											
11	*										
10	12	*									
9	11	13	*								
8	10	12	19	*							
7	9	11	18	20	*						
6	8	10	17	19	21	*					
5	7	9	16	18	20	24	*				
4	6	8	15	17	19	23	25	*			
3	5	7	14	16	18	22	24	26	*		
2	4	6	13	15	17	21	23	25	27	*	
1	3	5	12	14	16	20	22	24	26	28	*
	A_0			A_1				A_2			

Figure 2: Algorithme de Sameh et Kuck à p processeurs

Théorème 1: Si SK(p) est le nombre de pas pour calculer la décomposition de Givens de A par l'algorithme de Sameh et Kuck à p processeurs alors

$$SK(p)=\frac{n^2}{2p}+\frac{n}{2}+p-\frac{n}{p}-1$$

Démonstration

$SK(p)=(d_{q-2}-1)+(4p-3)$ où d_{q-2} est le temps auquel on commence le traitement de A_{q-2} et (4p-3) est le nombre de pas nécessaire pour annuler les éléments situés dans A_{q-2} par l'algorithme de Sameh et Kuck.

Soient d_{k-1} la date à laquelle on commence à traiter le bloc A_{k-1} et f_{k-1} la date à laquelle on termine le traitement de la première colonne de A_{k-1}. Puisque les éléments de la première colonne de A_{k-1} sont annulés séquentiellements alors on a:

(1) $$f_{k-1}= (d_{k-1}-1)+(n-(k-1)p-1)=n-(k-1)p+d_{k-1}-2.$$

Au pas $f_{k-1}+1$ on commence le traitement de A_k on a donc:

(2) $$d_k=f_{k-1}+1=n-(k-1)p+d_{k-1}-1$$

De (1) et (2) on déduit la relation de récurrence entre d_k et d_{k-1}

(3) $$d_k-d_{k-1}=n-(k-1)p-1 \qquad \text{pour } k=1,\ldots,q-2 \text{ et } d_0=1.$$

On obtient donc:

(4) $$d_k-d_0=k(n-1)-p\frac{k(k-1)}{2}$$

En remplaçant k par q-2 dans la relation (4) on obtient:

$d_{q-2}=\frac{(q+1)n}{2}-q-3p+3$ par conséquent $SK(p)=\frac{(q+1)}{2}n+p-q-1=\frac{n^2}{2p}+\frac{n}{2}+p-\frac{n}{p}-1$

Corollaire 1: Si p est d'ordre strictement inférieur à n alors l'algorithme de Sameh et Kuck à p processeurs est asymptotiquement optimal. Son temps d'exécution est égal à $SK(p) = \frac{n^2}{2p} + o(n)$.

Démonstration

Si p est d'ordre < n alors $\frac{n^2}{2p}$ est d'ordre > n par conséquent

$$SK(p)=\frac{n^2}{2p}+\frac{n}{2}+p-q-1=\frac{n^2}{2p}+o(n)$$

2.2. Algorithmes glouton à p processeurs

Le principe de l'algorithme glouton à p processeurs [MC],[CDR] consiste à annuler à chaque pas le maximum d'éléments avec p processeurs. Les éléments sont annulés de bas en haut et de gauche à droite en conservant tous ceux qui sont déjà annulés.

Définition 1: Un algorithme de décomposition QR par la méthode de Givens est dit glouton à p processeurs si:

$$\sum_{j=1}^{n} r(j,t)=\min(p\ ,\ \sum_{j=1}^{n} \lfloor \frac{s(j-1,t-1)-s(j,t-1)}{2} \rfloor)$$

Il est clair qu'il existe plusieurs façons pour calculer cette somme et, par conséquent, plusieurs algorithmes glouton à p processeurs contrairement au cas $p=\lfloor \frac{m}{2} \rfloor$. Nous présentons deux algorithmes notés algorithmes glouton vertical et glouton horizontal.

Algorithme glouton vertical

Cet algorithme consiste à annuler les éléments de bas en haut et avec une priorité pour annuler sur les premières colonnes. Il est défini donc par:

$$r(i,t)=\min(p-\sum_{j=1}^{i-1} r(j,t),\ \lfloor \frac{s(i-1,t-1)-s(i,t-1)}{2} \rfloor)$$

*											
5	*										
4	8	*									
3	7	11	*								
3	7	10	14	*							
3	6	10	13	16	*						
2	6	9	12	15	18	*					
2	6	9	12	15	17	20	*				
2	5	9	12	14	17	19	22	*			
1	5	8	11	14	16	18	21	23	*		
1	4	8	11	13	16	18	20	22	24	*	
1	4	7	10	13	15	17	19	21	23	25	*

Figure 3: Le glouton vertical pour m=n=12 et p=3

Algorithme glouton horizontal

Cet algorithme consiste à annuler les éléments de bas en haut et par priorité d'annuler sur les dernières lignes, il est défini par:

$$r(i,t)=\min\left(p-\sum_{j=i+1}^{n} r(j,t),\ \left\lfloor \frac{s(i-1,t-1)-s(i,t-1)}{2} \right\rfloor\right)$$

*											
15	*										
13	16	*									
11	14	17	*								
5	12	15	18	*							
4	7	13	16	19	*						
3	6	9	14	17	20	*					
2	5	8	10	15	18	21	*				
2	4	7	9	11	16	19	22	*			
1	3	6	8	10	12	17	20	23	*		
1	3	5	7	9	11	13	18	21	24	*	
1	2	4	6	8	10	12	14	19	22	25	*

Figure 4: Le glouton horizontal pour m=n=12 et p=3

Remarque 1: Ni le glouton vertical ni le glouton horizontal ne sont optimaux comme le montre l'exemple illustré par la figure 5 dans le cas où m=n=12 et p=3.

*											
7	*										
6	10	*									
5	9	13	*								
4	8	12	15	*							
4	7	11	14	17	*						
2	6	10	13	16	19	*					
2	5	9	12	15	18	20	*				
2	4	8	11	14	17	19	21	*			
1	3	7	10	13	16	18	20	22	*		
1	3	6	9	12	15	17	19	21	23	*	
1	3	5	8	11	14	16	18	20	22	24	*

Figure 5: L'algorithme glouton optimal (voir plus loin)

Théorème 2: Etant donnés m, n et p. Il existe un algorithme glouton à p processeurs qui permet de calculer la décomposition de Givens de manière optimale.

Pour la démonstration nous renvoyons à [CDR]

III.3. ETUDE DE LA COMPLEXITE

Nous nous limitons au cas des matrices carrées. Soient p le nombre de processeurs et n la taille de la matrice. Nous désignons par $T_{opt}(p)$ le temps optimal pour calculer la décomposition de Givens avec p processeurs et par T_{opt} le temps minimum avec un nombre de processeurs illimité. Remarquons que nous avons les relations suivantes:

$$T_{opt} = T_{opt}(\frac{n}{2}) \leq T_{opt}(p)$$

$T_{opt} = 2n\text{-}o(n)$ (théorème 6 ch. II)

3.1. Une borne inférieure

Dans ce paragraphe nous montrons comment obtenir une borne inférieure de $T_{opt}(p)$.

Théorème 3: Pour $1 \leq p \leq \frac{n}{2}$ on a: $T_{opt}(p) \geq \frac{n(n-1)}{2p} + p - 1 = B_{inf}(p)$

Démonstration:

Soit T(p) le temps d'exécution d'un algorithme avec p processeurs. D'après la propriété 1 [Intro. et Not.] on a $T(p)= \frac{P_{in} + P_{ac}}{p}$ avec $P_{ac} = \frac{n(n-1)}{2}$. On a donc:

$$T(p) = \frac{n(n-1)}{2p} + \frac{P_{in}}{p}$$

Dans la suite nous montrons que $P_{in} \geq p(p-1)$ d'où $T(p) \geq \frac{n(n-1)}{2p} + p - 1$ par conséquent

$$T_{opt}(p) \geq \frac{n(n-1)}{2p} + p - 1 = B_{inf}(p)$$

Soit t_i la date de fin de traitement de la i ième colonne (le pas au cours duquel l'élément en position (i+1,i) est annulé). Pour annuler un élément on utilise deux lignes, par conséquent au plus min(p, $\lfloor \frac{n-i+1}{2} \rfloor$) éléments peuvent être annulés au moment où on annule l'élément en position (i+1,i). Donc on a:

$$ac(t_i) \leq \min(p, \lfloor \frac{n-i+1}{2} \rfloor) \text{ d'où } in(t_i) \geq p - \min(p, \lfloor \frac{n-i+1}{2} \rfloor)$$

ac(t) est le nombre de processeurs actifs et in(t) est le nombre de processeurs inactifs au temps t.

Pour $i \geq n-2p+1$, on a $\min(p, \lfloor \frac{n-i+1}{2} \rfloor) = \lfloor \frac{n-i+1}{2} \rfloor$ donc $in(t_i) \geq p - \lfloor \frac{n-i+1}{2} \rfloor$

Donc $$\sum_{i=n-2p+1}^{n-1} in(t_i) \geq \sum_{i=n-2p+1}^{n-1} \left(p - \lfloor \frac{n-i+1}{2} \rfloor \right) = p(p-1)$$

Par conséquent

$$P_{in} \geq \sum_{i=n-2p+1}^{n-1} in(t_i) \geq p(p-1)$$

Corollaire 2: Soit p_{opt} le nombre minimum de processeurs qui permet de calculer la décomposition de Givens d'une matrice carrée de taille n en T_{opt}. p_{opt} vérifie la relation: $\frac{n}{2+\sqrt{2}} + o(n) \leq p_{opt} \leq \frac{n}{2}$

Démonstration

1. Borne supérieure: $p_{opt} \leq \frac{n}{2}$, car on sait calculer la décomposition de Givens en T_{opt} avec $\frac{n}{2}$ processeurs par l'algorithme glouton [ch. II].

2. Borne inférieure: On cherche pour quelles valeurs de $p \leq \frac{n}{2}$ on a $T_{opt} \leq B_{inf}(p)$.

Asymptotiquement on résoud $2n \leq \frac{n^2}{2p} + p$. On obtient $p \leq \frac{n}{2+\sqrt{2}}$

De **1** et **2** on déduit que $\frac{n}{2+\sqrt{2}} + o(n) \leq p_{opt} \leq \frac{n}{2}$

cqfd

Dans le paragraphe 3.2 nous montrons comment construire un algorithme asymptotiquement optimal pour $p=\frac{n}{4}$ qui est une étape intermédiaire pour construire un algorithme asymptotiquement optimal pour $p=\frac{n}{2+\sqrt{2}}$. Dans le paragraphe 3.4 nous généralisons la construction pour p quelconque.

3.2. Algorithme asymptotiquement optimal pour $p=\frac{n}{4}$

Proposition 1: Pour $p=\frac{n}{4}$ on a: $T_{opt}(\frac{n}{4}) = B_{inf}(\frac{n}{4}) + o(n) = \frac{9n}{4} + o(n)$.

Démonstration

On va construire un algorithme asymptotiquement optimal pour $p=\frac{n}{4}$. Pour ce faire, on décompose la matrice A en 5 régions qui sont numerotées sur la figure 6 par I, I', II, III et IV. A la fin de traitement d'une région, l'écart entre ses colonnes est égal à 2.

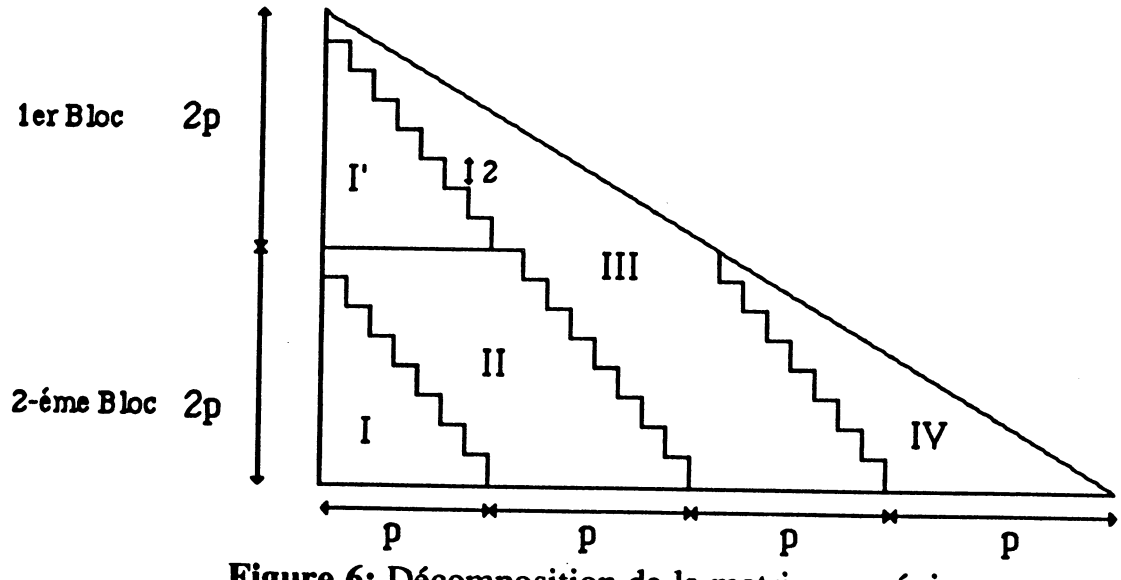


Figure 6: Décomposition de la matrice en régions

Description de l'algorithme:

L'algorithme est construit en 4 phases.

Première phase

Dans cette phase nous annulons les éléments situés dans les régions I et I' simultanément en gardant tous les processeurs actifs à tout moment. Donc si un processeur se libère dans une région on l'affecte dans l'autre.

Lemme 1: Les éléments situés dans les régions I et I' sont annulés avec $p=\frac{n}{4}$ processeurs en efficacité asymptotique égale à 1. Le temps d'exécution est égal à $T_1(\frac{n}{4})= \frac{n}{2} +o(n)$.

Démonstration

* On affecte les p processeurs pour annuler les éléments situés dans la région I' en utilisant la première phase de l'algorithme de Fibonacci modifié d'ordre 1. Le temps d'exécution de I' est égal à $2p+o(p)= \frac{n}{2} +o(n)$.

* Les processeurs inactifs dans la région I' seront affectés à la région I en utilisant la première phase de l'algorithme de Sameh et Kuck en commençant au temps $f_1+1=\lfloor \sqrt{n-2} \rfloor+1$. En effet dans la première phase de l'algorithme de Sameh et Kuck le nombre d'éléments à annuler simultanément augmente de 1 tous les 2 pas alors que le nombre de processeurs inactifs dans la région I' augmente de 1 tous les 2 pas à partir du temps f_1+1 [lemme 2 ch. II]. Par conséquent à chaque pas le nombre de processeurs actifs dans les deux régions est asymptotiquemet égal à p. Le temps d'exécution de I est égal à f_1+2p-2.

On conclut que le temps d'exécution des régions I et I' est égal à $T_1(\frac{n}{4})=\frac{n}{2} +o(n)$. La figure 7.a illustre le schéma d'élimination de la première phase dans le cas où n=12 et p=3. Dans ce cas, on a $f_1=3$.

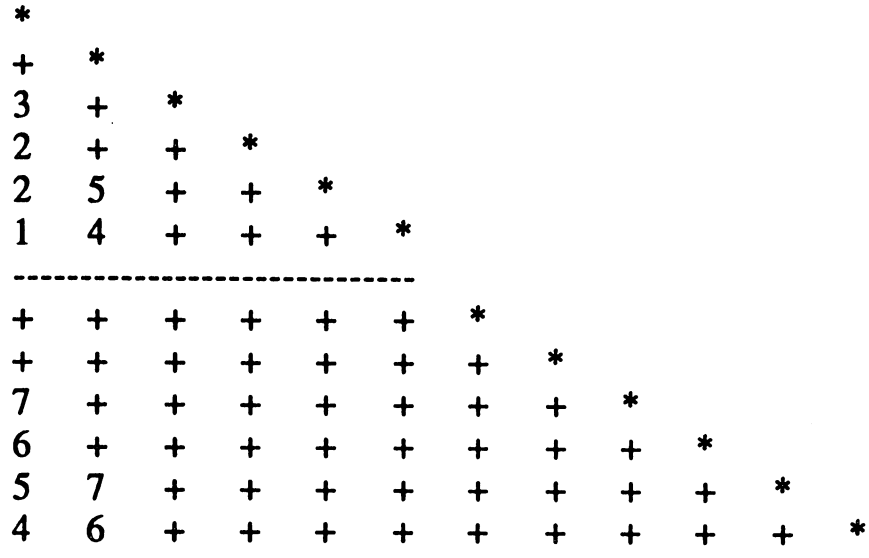


Figure 7.a: La première phase de l'algorithme

Deuxième phase

Lemme 2: Les éléments situés dans la région II sont annulés avec p= $\frac{n}{4}$ processeurs en efficacité 1. Le temps d'exécution est égal à $T_2(\frac{n}{4})=\frac{n}{2}$.

Démonstration

A la fin d'exécution des régions I et I', les p premières colonnes dans chacun des deux blocs sont actives d'écart égal à 2. Le principe d'élimination dans cette deuxième phase consiste à deplacer "l'escalier", dans la région II, vers la droite d'une position tous les 2 pas comme le montre la figure 7.b.

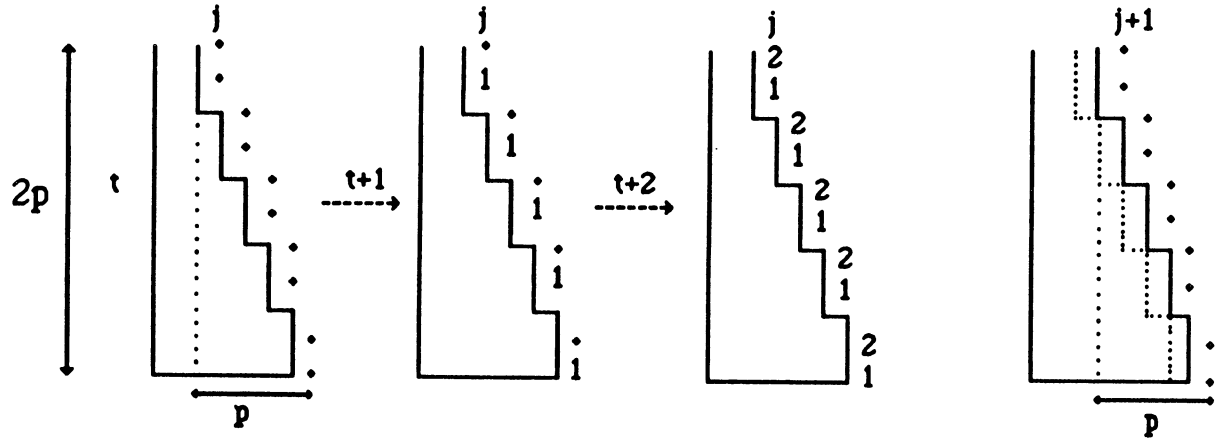


Figure 7.b: Evolution de l'élimination dans la région II

Pour cela on annule, à chaque pas, un élément dans chacune des p colonnes actives du deuxième bloc. Les lignes qui composent le premier bloc vont servir pour

définir les rotations. Supposons qu'au temps t, les j premières colonnes du deuxième bloc sont nulles et les p suivantes sont d'écart égal à 2.

Au temps (t+1) on annule un seul élément sur chacune des colonnes j,...,j+p-1. Pour annuler en position (i,k), i>2p et j≤k≤j+p-1, on utilise la rotation R(i,i-1,k). Les éléments annulés au temps (t+1) sont notés par 1 sur la figure 7.b.

Au temps (t+2) on annule à nouveau sur les colonnes j,...,j+p-1. Sur la colonne j, on annule en position (2p+1,j) en utilisant la rotation R(2p+1,j+2,j). Sur une colonne différente de la colonne j, on annule en position (i,k) en utilisant la rotation R(i,i-1,k). Les éléments annulés au temsp (t+2) sont noté par 2 sur la figure 7.b.

Comme "l'escalier" se déplace de p positions, on conclut que le temps d'exécution de cette phase est égal à 2p.

La figure 7.c illustre les deux premières phases dans le cas où n=12 et p=3. Dans ce cas, la deuxième phase commence au temps 8.

```
*
+   *
3   +   *
2   +   +   *
2   5   +   +   *
1   4   +   +   +   *
-----------------------------
9   11  13  +   +   +   *
8   10  12  +   +   +   +   *
7   9   11  13  +   +   +   +   *
6   8   10  12  +   +   +   +   +   *
5   7   9   11  13  +   +   +   +   +   *
4   6   8   10  12  +   +   +   +   +   +   *
```

Figure 7.c: Les deux premières phases de l'algorithmes pour n=12 et p=3

Troisième phase

Lemme 3: Les éléments situés dans la région III sont annulés avec p= $\frac{n}{4}$ processeurs en efficacité 1. Le temps d'exécution est égal à $T_3(\frac{n}{4}) = \frac{3n}{4}$.

Démonstration

A la fin d'exécution des deux régions I et I', les 2p premières colonnes sont actives

d'écart égal à 2. On élimine dans cette phase en efficacité égale à 1 d'une façon cyclique: si au temps t nous avons annulé sur les j premières colonnes actives alors au temps (t+1) on annule à partir de la colonne (j+1) et on complète à partir de la première colonne active.

Au premier pas, on annule p éléments sur les p premières colonnes, au deuxième pas, on annule sur les p colonnes suivantes. Par contre au troisième pas on annule à partir de la colonne 2 et au quatrième pas on annule (p-1) éléments à partir de la colonne (p+2) et on complète à partir de la colonne 3 (première colonne active) pour annuler un seul élément comme le montre l'exemple sur la figure 7.d. Comme les éléments sont annulés en efficacité égale à 1 alors le temps d'exécution est égal au nombre d'éléments de la région III divisé par p donc égal à $\frac{3n}{4}$.

```
*
14  *
3   16  *
2   14  17  *
2   5   16  19  *
1   4   14  18  20  *
9   11  13  16  19  22  *
8   10  12  15  18  21  +   *
7   9   11  13  17  20  22  +   *
6   8   10  12  15  18  21  +   +   *
5   7   9   11  13  17  20  22  +   +   *
4   6   8   10  12  15  19  21  +   +   +   *
```

Figure 7.d: Les trois premières phases de l'algorithme (Troisième phase commence au temps 14)

Quatrième phase

Lemme 4: Les éléments situés dans la région IV sont annulés en temps optimal. Le temps d'exécution est égal à $T_4(\frac{n}{4}) = \frac{n}{2} - 1$

Démonstration

A la fin d'exécution des régions I, I', II et III les 2p premières colonnes sont saturées et les p suivantes sont d'écart égal à 2. On annule dans la région IV en temps optimal. Sur chaque colonne active on annule un élément. Comme l'écart est égal à 2 alors à chaque pas, la première colonne active se sature. Donc le temps d'exécution est celui pour annuler les 2p-1 dernières colonnes actives donc égal à

$2p-1=\frac{n}{2}-1$.

*											
14	*										
3	16	*									
2	14	17	*								
2	5	16	19	*							
1	4	14	18	20	*						
9	11	13	16	19	22	*					
8	10	12	15	18	21	23	*				
7	9	11	13	17	20	22	24	*			
6	8	10	12	15	18	21	23	25	*		
5	7	9	11	13	17	20	22	24	26	*	
4	6	8	10	12	15	19	21	23	25	27	*

Figure 7.e: Les quatres phases de l'algorithme pour n=12 et p=3
(Quatrième phase commence au temps 23)

Conclusion: Le temps d'exécution de l'algorithme est la somme des temps d'exécution des quatres phases donc est égal à

$$T_1(\frac{n}{4})+T_2(\frac{n}{4})+T_3(\frac{n}{4})+T_4(\frac{n}{4})=\frac{9n}{4}+o(n)=B_{inf}(\frac{n}{4})+o(n).$$

3.3. Un algorithme asymptotiquement optimal pour $p=\frac{n}{2+\sqrt{2}}$

Proposition 2: Pour $p=\frac{n}{2+\sqrt{2}}$ on a: $T_{opt}(\frac{n}{2+\sqrt{2}})=2n+o(n)=B_{inf}(\frac{n}{2+\sqrt{2}})+o(n)$

Démonstration

Nous construisons un algorithme asymptotiquement optimal pour $p=\frac{n}{2+\sqrt{2}}$ qui s'exécute en $B_{inf}(\frac{n}{2+\sqrt{2}})+o(n)=2n+o(n)$. Le principe consiste à:

a. Décomposer la matrice A en trois blocs. Le premier bloc est composé des $\sqrt{2}p$ premières lignes, le deuxième bloc est composé des $2(\sqrt{2}-1)p$ lignes suivantes alors que le troisième bloc est composé des $2(2-\sqrt{2})p$ dernières lignes comme le montre la figure 8 (le deuxième et le troisième blocs sont composés des 2p dernières lignes).

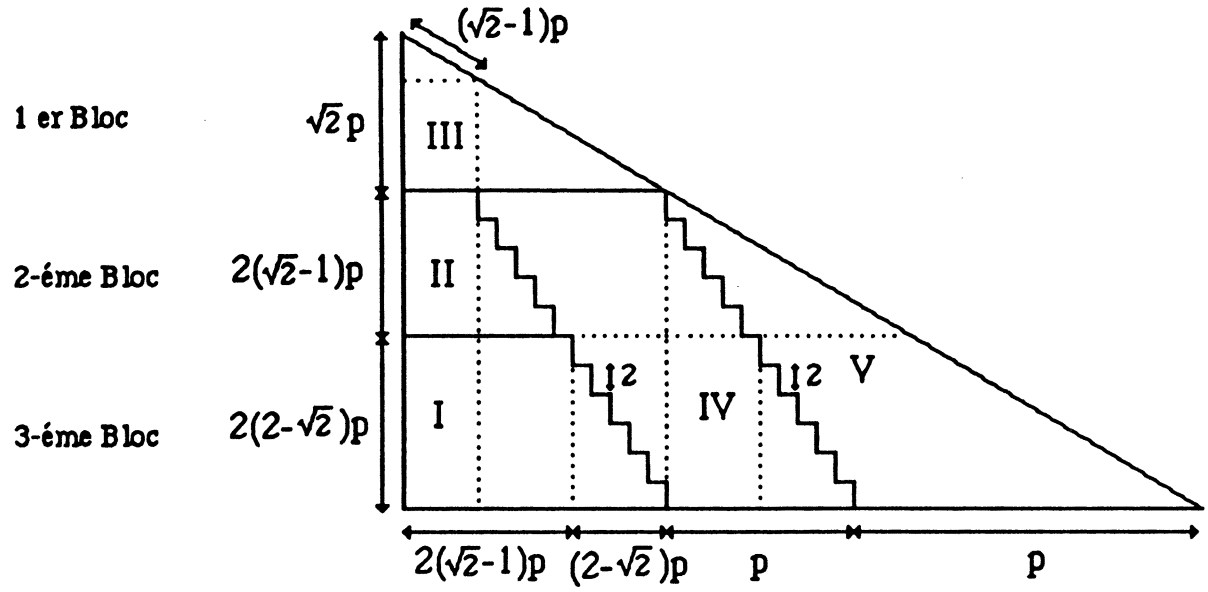


Figure 8: Décomposition en régions

b. Construire l'algorithme en 3 phases. Dans la première phase on annule les éléments situés dans les régions I, II et III en efficacité asymptotique égale à 1. Dans la deuxième phase on annule les éléments situés dans la région IV en efficacité égale à 1. Dans la troisième phase on annule les éléments situés dans la région V en temps optimal. A la fin d'exécution de chaque phase l'écart des colonnes actives est égal à 2.

Description des phases de l'algorithme

Première phase

Lemme 5: Les éléments situés dans les régions I, II et III sont annulés avec $p=\frac{n}{2+\sqrt{2}}$ processeurs en efficacité asymptotique égale à 1. Le temps d'exécution est égal à $T_1(p)=2\sqrt{2}p+o(p)=2(\sqrt{2}-1)n+o(n)$

Démonstration

La région III est de taille $\sqrt{2}p$, donc le temps optimal pour annuler ses éléments est égal à $2\sqrt{2}p+o(p)$ [théorème 6 ch. II]. Le but de cette phase est d'annuler pendant ce temps les éléments situés dans les régions I, II et III. Le processus d'élimination évolue en deux étapes.

Etape 1 de la première phase

Dans cette étape on annule les éléments situés dans les sous régions I_1, II_1 et III_1 comme le montre la figure 9.a. Pour cela, on distribue les processeurs aux sous régions de la façon suivante:

- On affecte $p_1=\frac{2-\sqrt{2}}{2}p$ processeurs à la sous région I_1.

- On affecte le reste des processeurs $p_2=p-p_1=\frac{\sqrt{2}}{2}p$ aux sous régions II_1 et III_1.

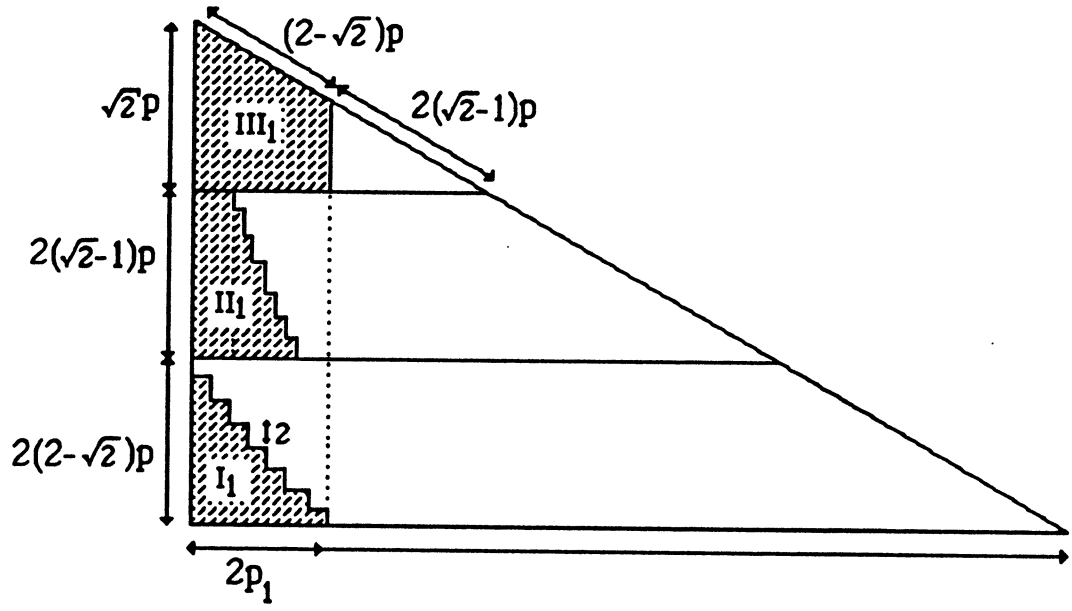


Figure 9.a: Les sous régions exécutées pendant l'étape 1

A la fin d'exécution de l'étape 1 on a:

* Les $(2-\sqrt{2})p$ premières colonnes de la sous région III_1 sont annulées.

* Dans la sous région II_1 (comme le montre la figure 9.b) on a:

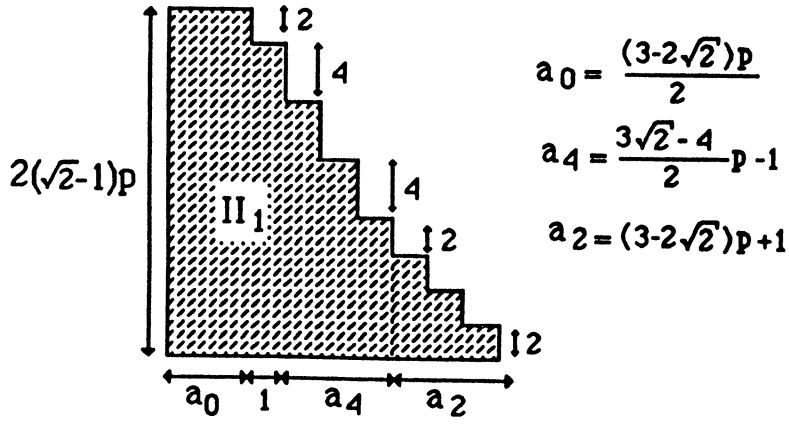


Figure 9.b: La sous région II_1.

- Les $\frac{3-2\sqrt{2}}{2}$p premières colonnes sont nulles.
- La colonne suivante est d'écart égal à 2
- Les $\frac{3\sqrt{2}-4}{2}$p-1 colonnes suivantes sont d'écart égal à 4
- Les $(3-2\sqrt{2})$p +1 colonnes suivantes sont d'écart égal à 2

* Les $(2-\sqrt{2})$p premières colonnes de la sous région I_1 sont d'écart égal à 2.

Lemme 5.1: Les éléments situés dans les sous régions I_1, II_1 et III_1 sont annulés en efficacité asymptotique égale à 1. Le temps d'exécution est égal à

$$t_1(p)=2(2-\sqrt{2})p+o(p).$$

Le processus d'élimination dans ces trois sous régions consiste à:

1. Annuler les éléments de la sous région I_1 indépendamment des deux autres sous régions avec $p_1=\frac{2-\sqrt{2}}{2}$p processeurs en utilisant les deux premières phases de l'algorithme §.3.2. En effet le nombre de lignes qui composent la sous région I_1 est égal à $4p_1$. Le temps d'exécution est donc égal à $t_1(p)=2(2-\sqrt{2})p+o(p)$.

2. Annuler dans la sous région III_1 avec le reste des processeurs $p_2=p-p_1=\frac{\sqrt{2}}{2}$p en utilisant l'algorithme de Fibonacci d'ordre 1. Donc, au temps $2(2-\sqrt{2})p+o(p)$, les $(2-\sqrt{2})$p premières colonnes sont annulées.

3. Annuler dans la sous région II_1 par les processeurs libérés de la région III_1 en commençant au temps $\sqrt{2\sqrt{2}p}$. En effet à partir de $\sqrt{2\sqrt{2}p}$ pas le nombre de processeurs libérés du premier bloc augmente de 1 tous les 4 pas [lemme 1 ch. II]. Le processus d'élimination dans cette région est divisé en deux étapes:

3.a. Pendant les $2(\sqrt{2}-1)p+o(p)$ premiers pas on annule de la façon suivante: les éléments seront annulés par colonne, de bas en haut et de gauche à droite. L'élément en position (i,k) est annulé dès que les éléments en positions:
(i,k-1), (i-1,k-1), (i-2,k-1), (i-3,k-1) sont tous nuls.

Ce processus d'élimination a le même principe que la première phase de l'algorithme de Sameh et Kuck. La différence réside dans le fait que dans cet algorithme le nombre d'éléments à annuler simultanément augmente de 1 tous les 4

pas, le temps pour qu'un processeur se libère du premier bloc, alors que dans celui de Sameh et Kuck le nombre d'éléments à annuler simultanément augmente de 1 tous les 2 pas. Donc au temps $2(\sqrt{2}-1)p+\sqrt{2\sqrt{2}p}$, les $\frac{\sqrt{2}-1}{2}p$ premières colonnes du deuxième bloc sont actives d'écart égal à 4 (représentées par la région II'_1 sur la figure 9.c). Pendant ce temps les $(\sqrt{2}-1)p$ premières colonnes du premier bloc sont annulées (représentées par la région III'_1 sur la figure 9.c).

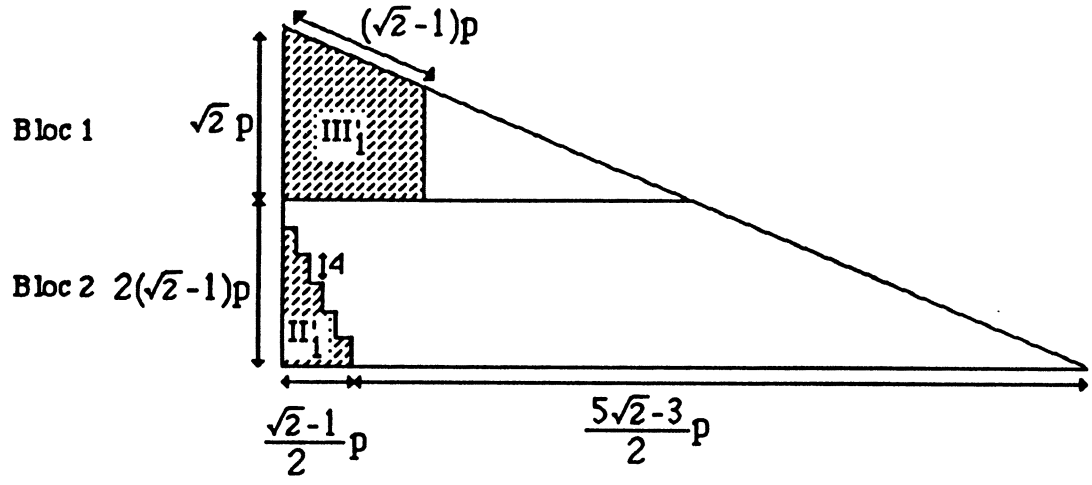


Figure 9.c: Exécution des régions II'_1 et III'_1 (premier et deuxième bloc)

3.b. Cette étape débute au temps $2(\sqrt{2}-1)p+o(p)$ et se termine au temps $t_1(p) = 2(2-\sqrt{2})p+o(p)$, donc elle dure $2(3-2\sqrt{2})p+o(p)$ pas. On élimine dans la sous région II_1 de manière à:

(i) Augmenter le nombre de colonnes actives de 1 tous les 4 pas. Par conséquent, on peut lui affecter le processeur libéré du premier bloc. De cette façon on garde tous les processeurs actifs.

(ii) Diminuer de 1 le nombre de colonnes d'écart égal à 4 et augmenter de 2 le nombre de colonnes d'écart égal à 2 tous les 4 pas.

Posons $q=\frac{\sqrt{2}-1}{2}p$ et $t=2(\sqrt{2}-1)p+o(p)$. Pendant les 2 premiers pas (ce qui ne modifie pas les résultats asymptotiques) on annule 2 éléments sur chaque colonne active. On se ramène donc au cas suivant:

- $e(1,t+2)=2$, la première colonne est d'écart égal à 2
- $e(2,t+2)=\ldots=e(q,t+2)=4$, les q-1 colonnes suivantes sont d'écart égal à 4
- $e(q+1,t+2)=2$, la colonne q+1 est d'écart égal à 2

Considérons maintenant le processus d'élimination suivant: à chaque pas, on annule un élément sur chaque colonne active. Montrons que ce processus d'élimination permet de satisfaire les conditions (i) et (ii).

En effet:

Posons $t=2(\sqrt{2}-1)p+o(p)$.

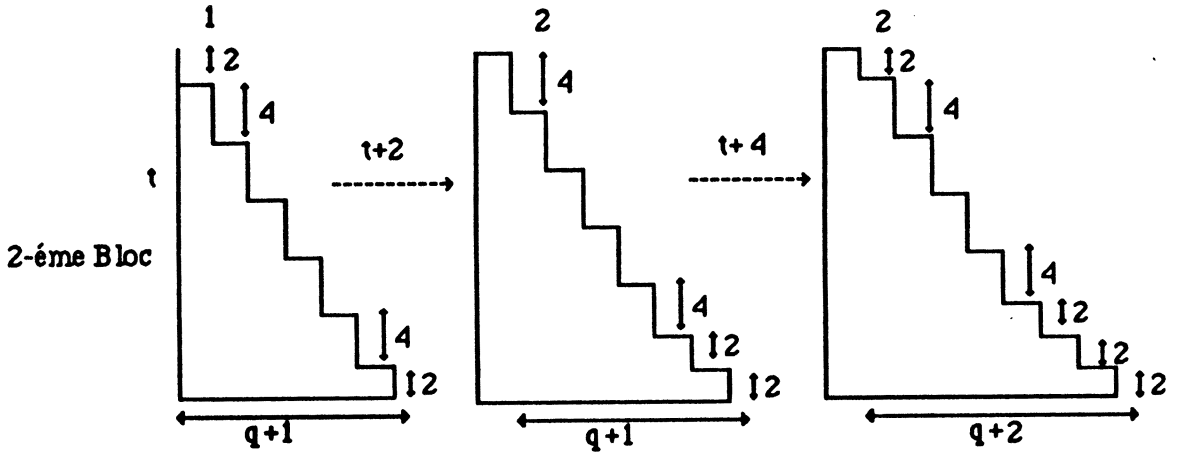


Figure 9.d: Evolution de l'élimination dans la région II_1

Comme le montre la figure 9.d, le principe d'élimination consiste, tous les 4 pas, à deplacer "l'escalier" de la sous région II'_1 vers la droite d'une position (annuler la première colonne active), diminuer de 1 le nombre de "marches" de longueurs égales à 4 et augmenter de 2 le nombre de "marches" de longueurs égales à 2. Par conséquent le nombre de "marches" augmente de 1.

Pendant les deux premiers pas, on annule un élément sur chacune des (q+1) colonnes actives c'est à dire sur les colonnes 1,2,...,q+1. Au premier pas, on utilise R(i,i-1,k) pour annuler l'élément en position (i,k). Au deuxième pas, pour annuler en position ($\sqrt{2}$p+1,1) on utilise R($\sqrt{2}$p+1,1,1) et pour annuler l'élément en position (i,k) (k>2) on utilise R(i,i-1,k). Donc au temps (t+2) on a:

- e(1,t+2)=0, la première colonne active (la colonne 1) s'annule
- e(2,t+2)=...=e(q,t+2)=4, les q-1 colonnes suivantes sont d'écart égal à 4
- e(q+1,t+2)=2, la colonne q+1 est d'écart égal à 2
- e(q+2,t+2)=2, une colonne d'écart égal à 2 se crée (la colonne q+2)

Pendant les deux pas suivants, on annule un élément sur chacune des (q+1) colonnes actives c'est à dire sur les colonnes 2,3,...,q+2. Pour annuler un élément en position(i,k), on utilise la rotation R(i,i-1,k). Donc au pas (t+4) on a:

- e(1,t+2)=0, la première colonne est annulée
- e(2,t+4)=2, l'écart de la colonne 2 a diminué de 2
- e(3,t+4)=...=e(q,t+4)=4, les q-2 colonnes suivantes sont d'écart égal à 4
- e(q+1,t+4)=e(q+2,t+4)=2, les colonnes suivantes sont d'écart égal à 2
- e(q+3,t+4)=2, une colonne d'écart égal à 2 se crée (la colonne q+3)

Donc au temps (t+4), le nombre de colonnes actives d'écart égal à 4 a diminué de 1 et le nombre de colonnes actives d'écart égal à 2 a augmenté de 2. Par conséquent le nombre de colonnes actives a augmenté de 1, donc les conditions (i) et (ii) sont satisfaites.

En utilisant le même principe d'élimination on montre facilement par récurrence que les conditions (i) et (ii) sont satisfaites tous les 4 pas. On déduit donc qu'au temps $t_1(p)=t+2(3-2\sqrt{2})p$ on a:

- Les $\frac{3-2\sqrt{2}}{2}p$ premières colonnes sont nulles.
- La colonne suivante est d'écart égal à 2
- Les $\frac{3\sqrt{2}-4}{2}p$ -1 suivantes sont d'écart égal à 4
- Les $(3-2\sqrt{2})p$ sont d'écart égal à 2

Etape 2 de la première phase

Dans cette étape on annule les éléments situés dans les sous régions I_2, II_2 et III_2 illustrées sur la figure 9.e. Pour cela, on redistribue les processeurs aux sous régions de la façon suivante:

- On affecte $p'_1=2p_1=(2-\sqrt{2})p$ pour annuler dans la sous région I_2
- On affecte le reste des processeurs $p'_2=p-p'_1=(\sqrt{2}-1)p$ pour annuler dans les deux sous régions II_2 et III_2.

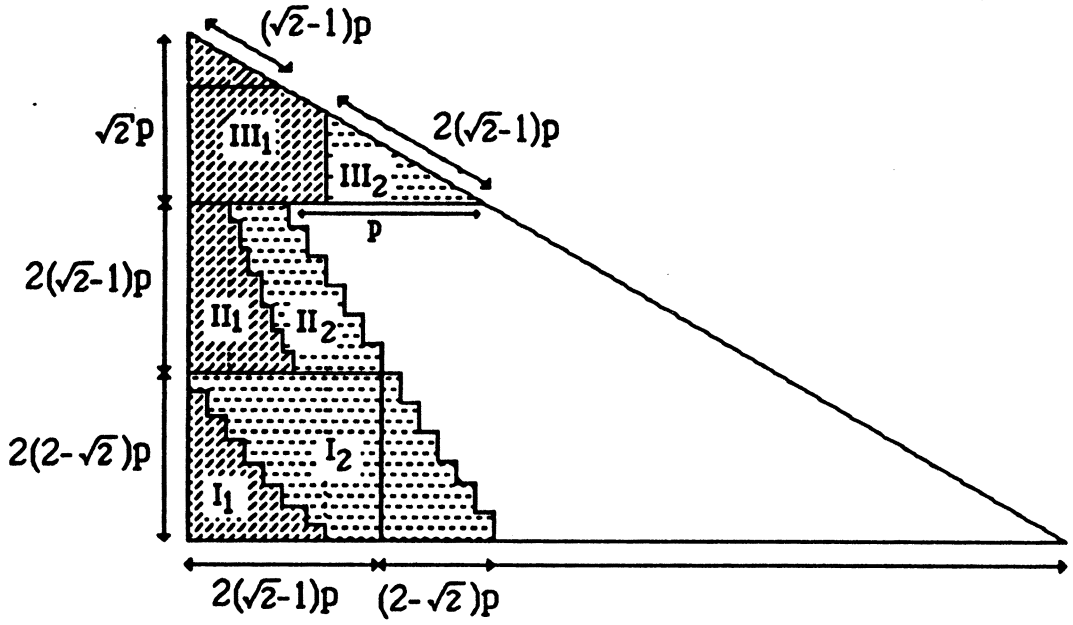


Figure 9.e: Exécution des sous regions I_2, II_2 et III_2

Lemme 5.2: Les éléments situés dans les sous régions I_2, II_2 et III_2 sont annulés en efficacité asymptotique égale à 1. Le temps d'exécution est égal à

$$t_2(p)=4(\sqrt{2}-1)p+o(p).$$

Le processus d'élimination dans ces trois sous régions consiste à:

1. Annuler dans la sous région I_2 en efficacité 1 avec $p'_1=(2-\sqrt{2})p$. En effet à la fin d'exécution de la sous région I_1 les $2p_1$ premières colonnes du troisième bloc sont actives d'écart égal à 2. On élimine donc par le même principe d'élimination que la deuxième phase de l'algorithme § 3.2 (à chaque pas on annule un élément sur chacune des colonnes actives). Le temps d'exécution est égal à $4(\sqrt{2}-1)p$.

2. Annuler dans la sous région III_2 avec le reste des processeurs $p'_2=p-p'_1=(\sqrt{2}-1)p$ en utilisant l'algorithme de Fibonacci d'ordre 1. Le temps d'exécution de la sous région III_2 est donc égal à $4(\sqrt{2}-1)p+o(p)$

3. Annuler dans la sous région II_2 avec les processeurs qui seront libérés de III_2 (Le nombre des processeurs libérés augmente de 1 tous les 4 pas). Le principe d'élimination consiste à éliminer en deux étapes:

3.a. Lorsque le nombre de processeurs affectés dans cette sous région est inférieur au nombre de colonnes actives alors on élimine d'une façon cyclique: si au pas t, on a éliminé sur les j premières colonnes actives alors au pas (t +1) on élimine à partir

de la colonne (j+1) et on complète à partir de la première colonne active. On peut montrer (facile mais long) que le nombre de colonnes d'écart égale à 4 diminue alors que le nombre de colonnes d'écart égal à 2 augmente.

3.b. Lorsque le nombre de processeurs affectés dans cette sous région est égal au nombre de colonnes actives alors on applique le processus d'élimination décrit dans §3.b de l'étape 1 de la première phase.

Conclusion: Le temps d'exécution des régions I, II et III est égal à la somme des temps d'exécution des deux étapes donc égal à $t_1(p)+t_2(p)=2\sqrt{2}p+o(p)$

Deuxième phase

Lemme 6: Les éléments situés dans la région IV sont annulés en efficacité 1. Le temps d'exécution est égal à: $T_2(\frac{n}{2+\sqrt{2}})=(2-\sqrt{2})n$.

Dans cette phase on annule les éléments situés dans la région IV avec le même principe d'élimination que la deuxième phase de l'algorithme § 3.2. Le temps d'exécution est égal à $2p=(2-\sqrt{2})n$.

Troisième phase

Lemme 7: Les éléments situés dans la région V sont annulés en temps optimal. Le temps d'exécution est égal à: $T_3(\frac{n}{2+\sqrt{2}})=(2-\sqrt{2})n-1$.

Dans cette phase on annule les éléments situés dans la région V avec le même principe d'élimination que la quatrième phase de l'algorithme § 3.2. Le temps d'exécution est égal à $2p-1=(2-\sqrt{2})n-1$.

Conclusion: Le temps d'exécution de l'algorithme est égal à la somme des temps d'exécutions des des 3 phases donc égal à:

$$T_{opt}(\frac{n}{2+\sqrt{2}})=2(\sqrt{2}-1)n+(2-\sqrt{2})n+(2-\sqrt{2})n+o(n)=2n+o(n)=B_{inf}(\frac{n}{2+\sqrt{2}})+o(n).$$

Corollaire 3: Le nombre minimum de processeurs qui permet de calculer la décomposition de Givens d'une matrice carrée de taille n en T_{opt} est égal à:

$$p_{opt}=\frac{n}{2+\sqrt{2}}+o(n)$$

Ce resultat est une conséquence directe de l'algorithme décrit pécédemment et du

corollaire 2.

3.4. Un algorithme asymptotiquement optimal pour p quelconque

Etant donnée n et p, nous construisons un algorithme asymptotiquement optimal qui s'exécute en $T_{opt}(p)=B_{inf}(p)$.

Théorème 4: Si $T_{opt}(p)$ est le nombre optimal de pas pour calculer la décomposition de Givens d'une matrice carrée de taille n avec p processeurs alors:

(1) $T_{opt}(p)= 2n+o(n)$ pour $p \geq \frac{n}{2+\sqrt{2}}$

(2) $T_{opt}(p)= \frac{n^2}{2p} +p + o(n)$ pour $1 \leq p \leq \frac{n}{2+\sqrt{2}}$

Démonstration

(1) Si $p \geq \frac{n}{2+\sqrt{2}}$ alors il suffit d'utiliser $p=\frac{n}{2+\sqrt{2}}$ pour construire un algorithme asymptotiquement optimal qui s'exécute en $T_{opt} =2n+o(n)$

(2) Si $p < \frac{n}{2+\sqrt{2}}$ alors nous construisons un algorithme asymptotiquement optimal qui s'exécute en $B_{inf}(p)+o(n) = \frac{n^2}{2p} +p + o(n)$

Avant de construire l'algorithme dans le cas général on va étudier tout d'abord le cas où $n=(2+\sqrt{2} +r)p$ avec $0 \leq r<2$

Proposition 3: Pour $p= \frac{n}{2+\sqrt{2}+r}$ avec $0 \leq r<2$ on a: $T_{opt}(p)=\frac{n^2}{2p} +p + o(n)$

Description de l'algorithme

a. Décomposer la matrice A en trois blocs le premier est composé des $\sqrt{2}p$ premières lignes, le deuxième est composé des rp lignes suivantes alors que le troisième est composé des 2p dernières lignes comme le montre la figure 10.a.

b. Construire un algorithme en 3 phases. Dans les deux premières phases on annule les éléments situés dans les régions I, I', II et II' en efficacité asymptotique égale à 1. Dans la troisième phase on annule les éléments situés dans la région III en temps optimal.

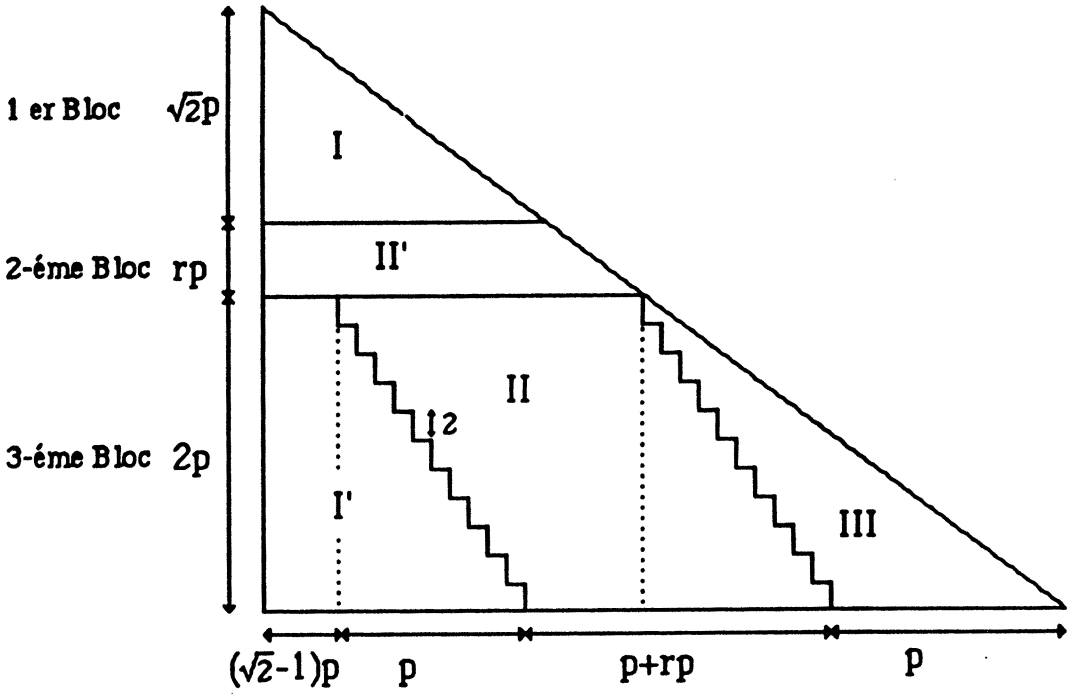


Figure 10.a: Décomposition en régions

Première phase

Lemme 8: Les éléments situés dans les régions I et I' sont annulés avec $p=\frac{n}{2+\sqrt{2}+r}$ processeurs en efficacité asymptotique égale à 1. Le temps d'exécution est égal à: $T_1(p)=2\sqrt{2}p+o(p)$.

On annule les éléments situés dans les régions I et I' par les deux premières phases de l'algorithme § 3.3. En effet le nombre de lignes qui composent le premier et le deuxième bloc est égal à $(2+\sqrt{2})p$.

Deuxième phase

Lemme 9: Les éléments situés dans les régions II et II' sont annulés avec $p=\frac{n}{2+\sqrt{2}+r}$ processeurs en efficacité asymptotique égale à 1. Le temps d'exécution est égal à: $T_2(p)=\sqrt{2}rp+2p+2rp+\frac{r^2}{2}p+o(p)$

Démonstration

Le processus d'élimination dans ces deux régions évolue en quatre étapes:

Première étape: Elle consiste à :

- Eliminer dans le deuxième bloc avec la première phase de l'algorithme de Sameh et Kuck. Donc au pas (rp-2), les $\frac{rp}{2}$ premières colonnes sont d'écart égal à 2.
- Eliminer dans le troisième bloc avec le reste des processeurs de la façon suivante: si au pas t on a annulé jusqu'à la colonne j, alors au pas (t+1) on annule à partir de la colonne (j+1) et on complète à partir de la première colonne active. Comme le nombre de colonnes actives est égal à p alors on élimine en efficacité 1.

Deuxième étape: Elle consiste à:

- Affecter $p_1 = \frac{rp}{2}$ processeurs au deuxième bloc pour annuler avec le même principe d'élimination que la deuxième phase de l'algorithme 3.2
- Affecter le reste des processeurs $p_2 = p - p_1 = \frac{(2-r)p}{2}$ au troisième bloc pour annuler dans la région II de la même façon que la première étape.

Troisième étape: Elle consiste à:

- Annuler dans le deuxième bloc en temps optimal avec le même principe que la deuxième phase de l'algorithme de Sameh et Kuck. Donc tous les 2 pas, un processeur se libère qui sera affecté au troisième bloc. Dans ce dernier, on annule avec le même processus d'élimination que la première étape.

Quatrième étape: Elle consiste à:

Affecter les p proceseurs au troisième bloc. Comme il y a p colonnes actives, on annule alors le reste des éléments situés dans la région II en efficacité 1.

Le temps d'exécution de cette phase est asymptotiquement égal au nombre d'éléments annulés divisé par p donc égal à: $T_2(p) = \sqrt{2}rp + 2p + 2rp + \frac{r^2}{2}p + o(p)$.

Troisième phase

Lemme 10: Les éléments situés dans la région III sont annulés en temps optimal. Le temps d'exécution est égal à 2p-1.

Conclusion: Le temps d'exécution est la somme des temps d'exécutions des 3 phases donc égal à: $T_{opt}(p) = 2\sqrt{2}p + \sqrt{2}rp + 4p + 2rp + \frac{r^2}{2}p + o(n) = \frac{n^2}{2p} + p + o(n)$

CAS GENERAL

On pose que n=(2k+$\sqrt{2}$+r)p avec k entier supérieur à 1 (k>1) et 0≤r<2. Le principe de l'algorithme consiste à:

a. Décomposer la matrice A en (k+1) blocs le premier est composé des ($\sqrt{2}$+r)p premières lignes alors que les k blocs suivants sont composés chacun de 2p lignes comme le montre la figure 11.

b. Construire l'algorithme en 2k phases. Dans la première phase on annule les éléments situés dans la région I en efficacité asymptotique égale à 1. Dans la 2j-ème phase on annule les éléments situés dans les régions 2J et 2J' en efficacité égale à 1 et dans la (2j+1)-ème phase on annule les éléments situés dans la région (2J+1) en efficacité 1. La 2k-ème phase consiste à annuler les éléments de la région K en temps optimal.

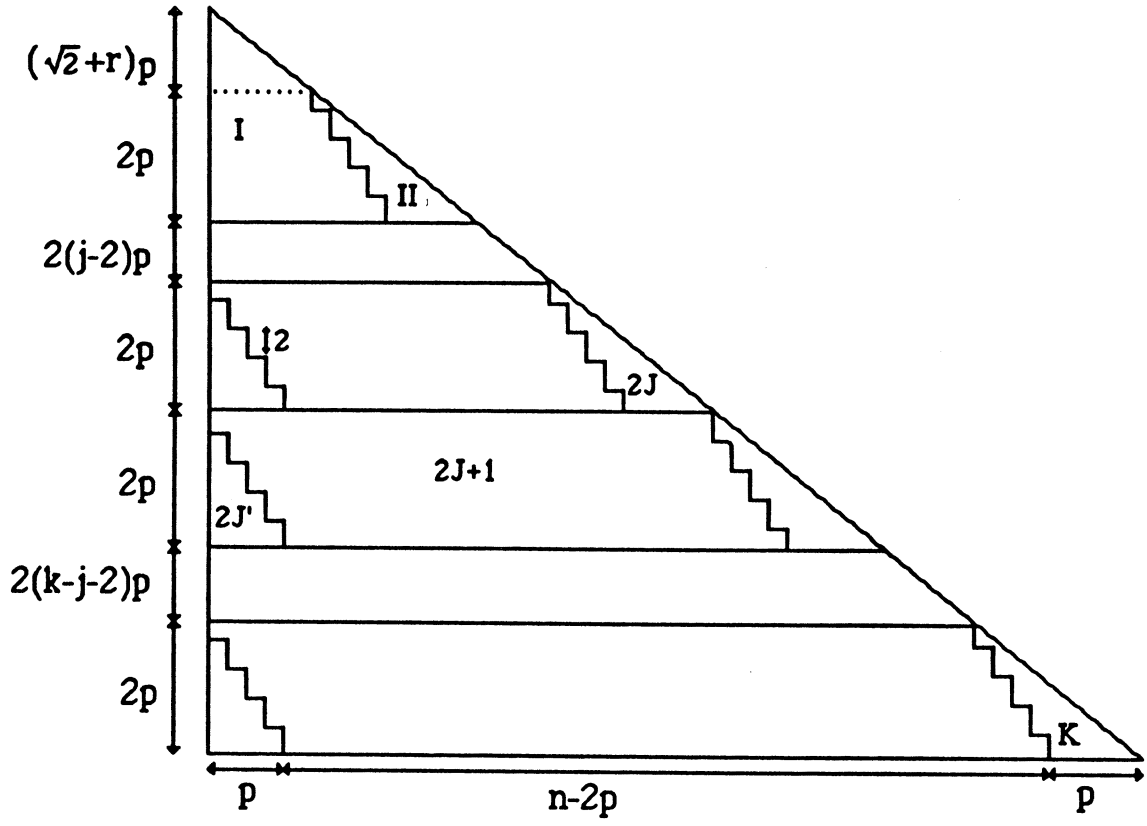


Figure 11: Décomposition en régions

Description des phases de l'algorithme:

Première phase:

On annule les éléments situés dans la région I en efficacité asymptotique égale à 1

en utilisant les deux premières phases de l'algorithme décrit précédemment.

2j-ème phase:

On annule les éléments situés dans la région 2J par la deuxième phase de l'algorithme de Sameh et Kuck. Tous les 2 pas un processeur se libère qu'on utilise pour annuler les éléments situés dans la région 2J' par la première phase de l'algorithme de Sameh et Kuck. Donc les éléments situés dans les régions 2J et 2J' sont annulés en efficacité 1.

(2j+1)-ème phase:

On annule les éléments situés dans la région 2J+1 avec le même principe d'élimination que la deuxième phase de l'algorithme § 3.2 donc en efficacité 1.

2K-ème phase:

On annule les élément situés dans la région K en temps optimal avec le même principe d'élimination que la quatrième phase de l'algorithme § 3.2.

Conclusion:

* Dans les 2k-1 premières phases nous avons annulé en efficacité asymptotique égale à 1. Donc le temps d'exécution est asymptotiquement égal au nombre d'éléments annulés divisé par p. Il est égal à: $\frac{n(n-1)}{2p}$ -(p-1)+o(n)
* Dans la 2k ième phase nous avons annulé en temps optimal égal à 2p-1.

Par conséquent le temps d'exécution de l'algorithme est la somme des temps d'exécutions des k phases, donc égal à $\frac{n(n-1)}{2p}$ +p +o(n) =$B_{inf}(p)$+o(n)

III.4. SELECTION DES ALGORITHMES

La construction de l'algorithme, dans le cas p quelconque, est difficile et longue. L'objectif donc dans ce paragraphe, est de construire des algorithmes asymptotiquement optimaux pour $p \geq \frac{n}{3}$ et pour $1 \leq p \leq \frac{n}{4}$. Ces algorithmes sont moins difficiles à construire. Ceci nous permet de selectionner les meilleurs algorithmes suivant la valeur de p.

4.1. Un algorithme asymptotiquement optimal pour $p \geq \frac{n}{3}$

Nous construisons un algorithme asymptotiquement optimal pour $p=\frac{n}{3}$ qui est en $2n+o(n)$. Si $p>\frac{n}{3}$, il suffit de construire l'algorithme pour $p=\frac{n}{3}$.

Proposition 4: Pour $p=\frac{n}{3}$ on a: $T_{opt}(\frac{n}{3}) \leq 2n - 3$

Le principe consiste à:

a. Décomposer la matrice A en deux blocs le premier est composé des p premières lignes alors que le deuxième est composé des 2p dernières lignes comme le montre la figure 12.

b. Construire un algorithme en 3 phases. Dans la première phase on annule les éléments situés dans les régions I et I', dans la deuxième phase on annule les éléments situés dans la région II en efficacité égale à 1. Dans la troisième phase on annule les éléments situés dans la région III en temps optimal.

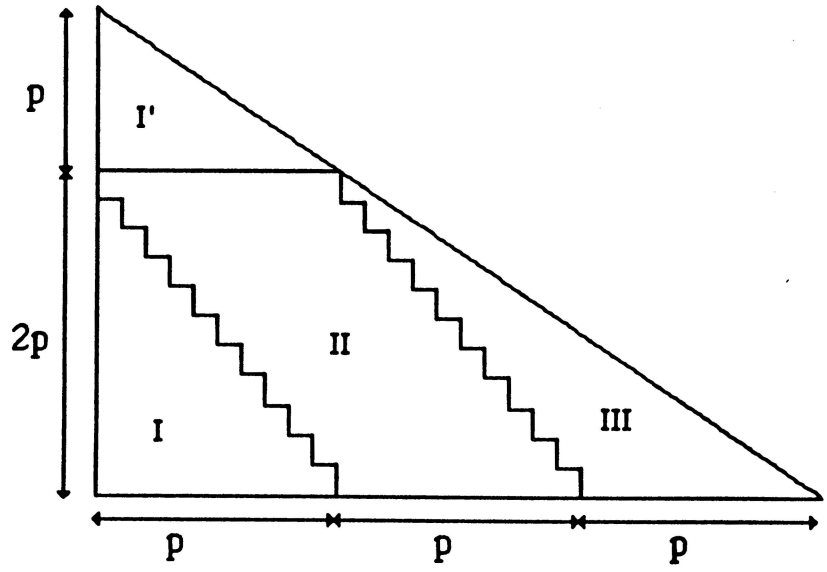


Figure 12: Décomposition en régions

Première phase

Lemme 11: Les éléments situés dans les régions I, I' sont annulés en $T_1(\frac{n}{3})=\frac{2}{3}n - 2$ pas avec $p=\frac{n}{3}$ processeurs.

* Dans chacune des deux régions on annule avec l'algorithme de Sameh et Kuck. Donc dans chacune des deux régions, le nombre d'éléments à annuler

simultanément augmente de 1 tous les 2 pas. Au temps t_1=p-1 (temps d'exécution de la première phase de l'algorithme de Sameh et Kuck dans la région I'), on utilise $\lfloor p/2 \rfloor$ processeurs dans chaque région.

* A partir de t_1+1 le nombre de processeurs actifs dans la région I' diminue de 1 tous les 2 pas (deuxième phase de l'algorithme de Sameh et Kuck). Les proceseurs libérés seront affectés au deuxième bloc car dans la région I, le nombre d'éléments à annuler simultanément augmente de 1 tous les 2 pas (première phase de l'algorithme de Sameh et Kuck). Cette phase est illustrée sur la figure 12.a dans le cas où n=12 et p=4.

```
*
3   *
2    4   *
1    3   5   *
--------------------
+   +   +   +   *
+   +   +   +   +   *
6   +   +   +   +   +   *
5   +   +   +   +   +   +   *
4   6   +   +   +   +   +   +   *
3   5   +   +   +   +   +   +   +   *
2   4   6   +   +   +   +   +   +   +   *
1   3   5   +   +   +   +   +   +   +   +   *
```

Figure 12.a: Ordre d'élimination dans la première phase

Deuxième phase

Lemme 12: Les éléments situés dans la région II sont annulés avec p= $\frac{n}{3}$ processeurs en efficacité égale à 1. Le temps d'exécution est égal à $T_2(\frac{n}{3})=\frac{2n}{3}$.

On annule les éléments situés dans la région II avec le même principe d'élimination que la deuxième phase de l'algorithme § 3.2.

Troisième phase

Lemme 13: Les éléments situés dans la région III sont annulés avec p=$\frac{n}{3}$ processeurs en temps optimal. Le temps d'exécution est égal à $T_3(\frac{n}{3})=\frac{2n}{3}$ -1.

On annule les éléments situés dans la région III avec le même principe d'élimination que la quatrième phase de l'algorithme § 3.2.

La figure 12.b illustre l'algorithme dans le cas où n=12 et p=4. La deuxième phase commence au temps 7 et la troisième phase au temps 15.

```
*
3   *
2   4   *
1   3   5   *
--------------------
8   10  12  14  *
7   9   11  13  15  *
6   8   10  12  14  16  *
5   7   9   11  13  15  17  *
4   6   8   10  12  14  16  18  *
3   5   7   9   11  13  15  17  19  *
2   4   6   8   10  12  14  16  18  20  *
1   3   5   7   9   11  13  15  17  19  21  *
```

Figure 12.b: Ordre d'élimination des éléments

Conclusion: le temps d'exécution est la somme des temps d'exécutions des trois phases donc égal à 2n - 3.

4.2. Un algorithme asymptotiquement optimal pour $p \leq \frac{n}{4}$

Nous supposons que n=4p+rp avec 0≤r. Dans ce paragraphe nous construisons un algorithme asymptotiquement optimal qui s'exécute en $T_{opt}(p)=B_{inf}(p)+o(n)$. Le principe consiste à:

a. Décomposer la matrice en deux blocs. Le premier est composé des 2p premières lignes et le deuxième est composé des restes des lignes comme le montre la figure 13.

b. Construire un algorithme en 3 phases. Les deux premières phases consiste à annuler les éléments situés dans les régions I, I', II et II' en efficacité asymptotique égale à 1. La troisième phase consiste à annuler les éléments situés dans la région III en temps optimal.

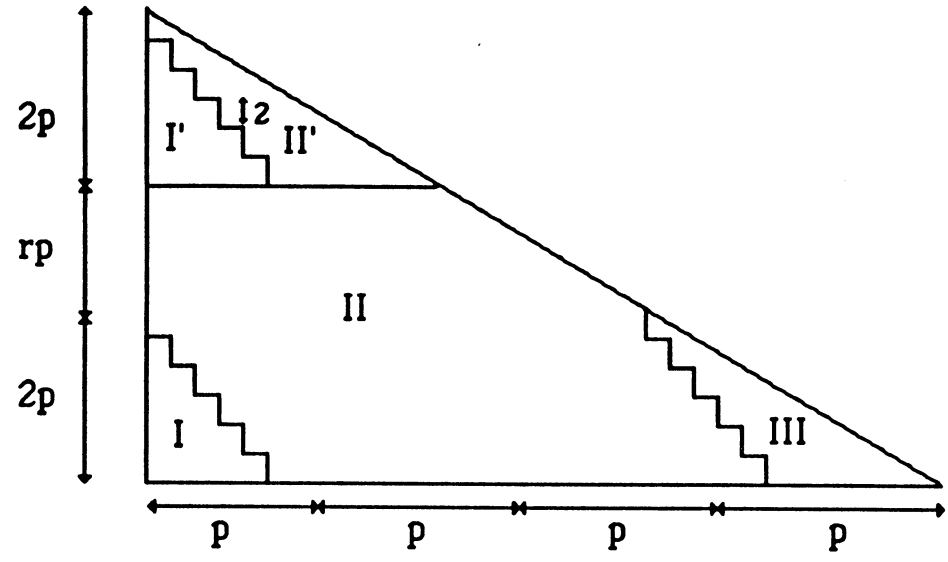


Figure 13: Décomposition en régions

Première phase:

Dans cette phase on annule les éléments situés dans les régions I et I' en efficacité asymptotique égale à 1, en utilisant les deux premières phases de l'algorithme § 3.2. Donc son temps d'exécution est égal à 2p+o(p)

Deuxième phase:

On annule les éléments situés dans les régions II et II' de la façon suivante:

* Dans la région II', par la deuxième phase de l'algorithme de Sameh et Kuck. les processeurs libérés seront affectés au deuxième bloc pour annuler dans la région II. Cette phase est illustrée sur la figure 14. Elle commence au temps 1 et se termine au temps 7 dans le cas où n=16 et p=3.

* Dans la région II, par une élimination cyclique: si au pas t, nous avons annulé dans les j premières colonnes actives, alors au pas (t+1) on annule à partir de la colonne (j+1) et on termine à partir de la première colonne active. Comme le nombre de colonnes actives est supérieur ou égal à p alors on élimine en efficacité égale à 1. Cette phase est illustrée sur la figure 14. Elle commence au temps 8 et se termine au temps 40 dans le cas où n=16 et p=3.

Le temps d'exécution est égal au nombre d'éléments des régions II et II' divisé par p, donc égal à $\frac{n^2}{2p}$ -3p +o(p).

Troisième phase:

On annule dans la région III avec le même principe d'élimination que la quatrième phase de l'algorithme §3.2. Donc son temps d'exécuction est égal à 2p-1. Cette phase est illustrée sur la figure 14. Elle commence au temps 41 dans le cas où n=16 et p=3.

Conclusion: Le temps d'exécution est égal à la somme des temps d'exécutions des 3 phases donc égal à $\frac{n^2}{2p}$ + p+o(n)

```
 *
 8  *
 3  9  *
 2  8  10 *
 2  5  9  11 *
 1  4  8  10 12 *
 ----------------------------
 17 20 24 26 30 34 *
 15 19 22 25 29 32 35 *
 14 17 21 24 27 31 34 37 *
 13 16 19 22 26 29 32 36 38 *
 11 14 18 21 24 28 31 34 37 40 *
 9  13 16 19 23 26 29 33 36 39 41 *
 7  12 15 18 21 25 28 31 35 38 40 42 *
 6  10 13 16 20 23 27 30 33 36 39 41 43 *
 5  7  12 15 18 22 25 28 32 35 38 40 42 44 *
 4  6  11 14 17 20 23 27 30 33 37 39 41 43 45 *
```

Figure 14: Ordre d'élimination des éléments

III.5. CONCLUSION

Nous avons prouvé que le nombre minimum de processeurs qui permet de calculer la décomposition de Givens d'une matrice carrée de taille n en temps optimal T_{opt}=2n+o(n) est égal à $p_{opt}=\frac{n}{2+\sqrt{2}}$ +o(n).

Lorsque p vérifie $1 \leq p \leq \frac{n}{2+\sqrt{2}}$, nous avons montré comment construire un algorithme asymptotiquememnt optimal qui s'exécute en $T_{opt}(p)=\frac{n^2}{2p}$ +p+o(n).

Lorsque $p \geq \frac{n}{2+\sqrt{2}}$ il suffit d'utiliser $p_{opt}=\frac{n}{2+\sqrt{2}}$ processeurs pour construire un algorithme asymptotiquemet optimal qui s'exécute en $T_{opt}=2n+o(n)$.

Dans les paragraphes 4.1 et 4.2 nous avons construit deux algorithmes asymptotiquement optimaux qui nous permettent de selectionner les meilleurs algorithmes parmi les algorithmes asymptotiquement optimaux suivant la valeur de p. Comme tous les algorithmes sont asymptotiquemet optimaux il est préférable de choisir les plus simples et ceux qui demandent moins de contôle. On peut donc conclure:

* Si $1 \leq p \leq \frac{n}{4}$ on choisit l'algorithme § 4.2
* Si $\frac{n}{4} \leq p \leq \frac{n}{2+\sqrt{2}}$ on choisit l'algorithme § 3.4
* Si $\frac{n}{2+\sqrt{2}} \leq p \leq \frac{n}{3}$ on choisit l'algorithme § 3.3
* Si $p > \frac{n}{3}$ on choisit l'algorithme § 4.1

Dans le cas des matrices rectangulaires le problème d'optimalité reste ouvert (construction et temps d'exécution de l'algorithme optimal).

CHAPITRE IV

COMPLEXITE DE LA DECOMPOSITION DE GIVENS SUR DES ARCHITECTURES DE TYPE SIMD ET MIMD

IV.1. INTRODUCTION

Les premières études concernant la complexité des algorithmes avaient pour but l'exploitation maximale du parallélisme inhérent à chaque application en supposant que le système parallèle est composé d'un nombre non limité de processeurs. On suppose que chaque processeur exécute une opération arithmétique (+,-,*,/) en une unité de temps qu'on note "flop".

La résolution du système linéaire Ax=b avec A une matrice carrée de taille n et b est un vecteur de IR^n a fait l'objet de plusieurs études à cause de sa grande importance dans le calcul scientifique. Csanky à développé un algorithme qui calcule A^{-1} en $O(\log_2^2 n)$ en utilisant $O(n^4)$ processeurs, mais en pratique cet algorithme est instable. Le temps d'exécution de la méthode de Gauss sans pivot est égal 3(n-1) flops en utilisant $(n-1)^2$ processeurs. Mais en l'absence d'informations sur la matrice A, le pivotage devient nécessaire. Dans ce cas le temps est en $O(n\log_2 n)$ flops avec $(n-1)^2$ processeurs. La méthode de Householder est en $O(n\log_2 n)$ avec (n-1) évaluations de racines carrées en utilisant n(n-1) processeurs. Sameh et Kuck [SK74] ont proposé un algorithme basé sur les rotations de Givens sans racine carrée qui s'exécute en 8n-7 flops en utilisant n^2 processeurs. Pour la méthode standard de Givens ils ont proposé un autre algorithme parallèle qui est en (10n-15) flops avec (2n-3) évaluations de racines carrées en utilisant $3n^2/2+o(n^2)$ processeurs [SK78]. Pour plus de détail sur la parallélisation de ces algorithmes nous renvoyons à [Hel], [Sam77], [Cos]. Dans le paragraphe IV.2, nous proposons une version parallèle pour calculer la décomposition de Givens avec racine carrée qui s'exécute en 8n-12 en utilisant seulement $3n^2/4+o(n^2)$ processeurs.

Lorsque le nombre de processeurs est limité, certains travaux utilisent le

formalisme de graphe des tâches [LKK], [MR], [CMRT], [CRT], [RTr] pour étudier la complexité des algorithmes d'algèbre linéaire. Pour la factorisation de Householder, on peut étendre les résultats de complexité obtenus pour la méthode de Gauss voir [RTr], [MR]. Le problème est différent pour la décomposition de Givens où peu de résultats sont apparus dans la littérature.

Dans ce chapitre nous nous consacrons à l'étude de la complexité de la décomposition de Givens sur deux types d'architectures à mémoire partagée. Dans le premier modèle, de type SIMD [Flyn], nous supposons que le nombre de processeurs est non limité. Chaque processeur exécute une opération élémentaire ce qui implique que chaque rotation sera exécutée par O(n) processeurs. Dans le deuxième modèle, de type MIMD [Flyn], chaque processeur génère et applique une rotation. Par conséquent le nombre de processeurs est au plus égal à $\lfloor \frac{n}{2} \rfloor$.

IV.2. CAS D' UNE ARCHITECTURE DE TYPE SIMD

Nous supposons que nous disposons d'un système multiprocesseur à mémoire partagée composée d'un nombre non limité de processeurs. Chaque processeur exécute une opération arithmétique (+,-,*,/) ou une racine carrée en une unité de temps. Donc le nombre de processeurs est borné par le nombre d'opérations que nécessite la méthode de Givens.

Soit D_k l'ensemble des rotations disjointes au k ième pas (groupe de rotations disjointes) et soit r_k le cardinal de D_k. La parallélisation de la méthode de Givens consiste, au k-ème pas, à exécuter simultanément les rotations disjointes appartenant à D_k. Soit R(i,j,h), $i \neq j$, $1 \leq i, j \leq m$ et $1 \leq h < i$, la rotation dans le plan (i,j) pour annuler l'élément en position (i,h). R(i,j,h) est représentée matriciellement par $P^h_{i,j}A$, où $P^h_{i,j}$ est la matrice de rotation associée à R(i,j,h). Soit r le nombre total de pas, Q est alors obtenue par la relation:

(1) $$Q=Q_rQ_{r-1}\ldots\ldots Q_2Q_1 \text{ où } Q_k = \prod_{(i,j,h)\in D_k} P^h_{i,j}$$

R est obtenue à partir de la relation de récurrence:

(2) $$A=A_1,\ldots,A_{k+1}=Q_kA_k,\ldots,R=Q_rA_r$$

2.1. Généralisation de la version parallèle proposée par Sameh et Kuck

Nous calculons tout d'abord la rotation et par la suite nous l'appliquons aux deux vecteurs qui la composent. R(i,j,h) est définie par l'algorithme suivant:

Calcul de $P^h_{i,j}$ << $t=\sqrt{a^2_{i,h} + a^2_{j,h}}$

$c_j=\frac{a_{i,h}}{t}$, $s_j=\frac{a_{j,h}}{t}$ >>

Calcul du produit $P^h_{i,j}A_k$ << **Pour** q=h+1 **à** n **faire**

$a_{i,q}:=c_j a_{i,q} + s_j a_{j,q}$

$a_{j,q}:=-s_j a_{i,q} + c_j a_{j,q}$

fpour

$a_{i,h}:=0$ >>

Le nombre d'opérations arithmétiques que nécessite l'algorithme séquentiel est égal à $T(1)=3n^2(m-\frac{n}{3})+o(mn^2)$ flops et $mn - \frac{n(n+1)}{2}$ racines carrées.

Soit D_k l'ensemble des rotations disjointes au k-ème pas. On pose $p_k=4\sum_{(i,j,h)\in D_k}(n-h)$

Lemme 1: En utilisant p_k processeurs , $A_{k+1}=Q_kA_k$ est déterminée en T=5 flops et une évaluation de racine carrée.

Démonstration

De la relation (1), on a $A_{k+1} = \prod_{(i,j,h)\in D_k}(P^h_{i,j}A_k)$. Les produits $P^h_{i,j}A_k$ pour (i,j,h) éléments de D_k sont indépendants, donc peuvent être effectués simultanément. Nous réalisons chacun des produits $P^h_{i,j}A_k$ en deux phases:

* Dans la **première phase**, avec 2 processeurs, nous calculons les coefficients c_j et s_j de $P^h_{i,j}$ de la façon suivante:

1ere étape: on calcule a^2_{ih} et a^2_{jh} en utilisant 2 processeurs
2ième étape: on calcule $a^2_{ih} + a^2_{jh}$ en utilisant 2 processeurs
3ième étape: on calcule $t=\sqrt{a^2_{i,h} + a^2_{j,h}}$ en utilisant 1 processeur
4ième étape: on calcule $c_j = \frac{a_{i,h}}{t}$ et $s_j = \frac{a_{j,h}}{t}$ en utilisant 2 processeurs

Par conséquent avec $2r_k$ processeurs on détermine toutes les matrices de rotations $P^h_{i,j}$, pour (i,j,h) éléments de D_k, en 3 flops et une racine carrée.

* Dans la **deuxième phase**, avec 4(n-h) processeurs, nous effectuons le produit $P^h_{i,j}A_k$ de la façon suivante:

1ere étape: On calcule $c_j a_{i,q}$, $c_j a_{j,q}$, $s_j a_{i,q}$ et $s_j a_{j,q}$ pour $h<q\leq n$ avec $4(n-h)$ processeurs.

2ième étape: On calcule $c_j a_{i,q}+s_j a_{j,q}$ et $-s_j a_{i,q}+c_j a_{j,q}$ pour $h<q\leq n$ avec $2(n-h)$ processeurs.

Par conséquent en 2 flops, nous calculons tous les produits $P^h_{i,j} A_k$ pour (i,j,h) éléments de D_k avec p_k processeurs.

Conclusion: A_{k+1} est déterminée en 5 flops et une évaluation de racine carrée en utilisant $\max(2r_k , p_k) = p_k$ processeurs.

Théorème 1: Le temps d'exécution d'un algorithme permettant de calculer la décomposition de Givens d'une matrice rectangulaire de taille (m,n) en r pas contenant chacune r_k rotations indépendantes, est égal à $T(p)=5r$ flops avec r évaluations de racines carrées en utilisant $p= \max_{1\leq k\leq r} (p_k)$ processeurs

Démonstration

D'aprés le lemme 1, A_k, pour $1\leq k\leq r$, est déterminée en 5 flops et une évaluation de racine carrée en utilisant p_k processeurs. Par conséquent, avec $p= \max_{1\leq k\leq r} (p_k)$ processeurs la décompsition de Givens est obtenue en 5r flops avec r évaluations de racines carrées.

Corollaire 1 [SK78]: Pour m=n, le temps d'exécution de l'algorithme de Sameh et Kuck est égal à $T(p)=10n-15$ flops et $2n-3$ évaluations de racines carrées avec $p=\frac{n(3n-2)}{2}$ processeurs.

En effet

* Le nombre de pas que nécessite l'algorithme de Sameh et Kuck est égal à $(2n-3)$ donc $T(p)=5(2n-3)$ flops avec $(2n-3)$ évaluations de racines carrées.

* Le cardinal le plus important de D_k est égal à $\lfloor \frac{n}{2} \rfloor$ (chapitre II). Il est obtenu pour $k=n-1$, les rotations correspondantes annulent un élément sur chacune des $\lfloor \frac{n}{2} \rfloor$ premières colonnes.

Corollaire 2 [SK78]: Pour m>n, le temps d'exécution de l'algorithme de Sameh et Kuck est égal à T(p)=5(m+n-2) flops et (m+n-2) évaluations de racines carrées avec $p=4nq-2q^2$ processeurs où $q=\min(n, \lfloor \frac{m}{2} \rfloor)$

En effet

* Le nombre de pas que nécessite l'algorithme de Sameh et Kuck est égal à m+n-2, donc T(p)=5(m+n-2) flops avec (m+n-2) évaluations de racines carrées.

* Le cardinal le plus important de D_k égal à $q=\min(n, \lfloor \frac{m}{2} \rfloor)$. Il est obtenu pour k=m-1. Les rotations correspondantes annulent un élément sur chacune des q premières colonnes.

Corollaire 3: Pour m=n le temps d'exécution de l'algorithme glouton est égal à T(p)=10n-o(n) flops et 2n-o(n) racines carrées en utilisant $p=2n^2-o(n^2)$ processeurs.

En effet

* Le nombre de pas que nécessite l'algorithme glouton est égal à 2n-o(n) donc T(p)=10n-o(n) flops et 2n-o(n) évaluations de racines carrées.
* Au premier pas de l'algorithme glouton on annule $\frac{n}{2}$ (n pair) éléments sur la première colonne. Par conséquent p= 2n(n-1). Si n est impair $p=2(n-1)^2$

Corollaire 4: Pour m>n, avec p=2m(n-1), le temps d'exécution de l'algorithme glouton est égal à:
* T(p)=10n-o(n) flops et 2n-o(n) racines carrées si $m=bn^k$ avec $1 \leq k < 2$.
* $T(p)=5\log_2(m)+o(\log_2(m))$ et $\log_2(m)+o(\log_2(m))$ racines carrées si n est fixe et m tend vers l'infini.

2.2. Proposition d'une version parallèle

Dans ce paragraphe nous proposons une version parallèle pour calculer la décomposition de Givens avec racine carrée qui améliore le temps d'exécution proposé par Sameh et Kuck [SK78]. Le principe de cette version est d'effectuer simultanément le calcul et l'application de la rotation aux deux vecteurs qui la composent. On définit donc la rotation R(i,j,h) de la façon suivante:

$t=\sqrt{a_{i,h}^2 + a_{j,h}^2}$

Pour q=h+1 **à** n **faire**

$$a_{i,q} = \frac{a_{i,h}a_{i,q} + a_{j,h}a_{j,q}}{t}$$

$$a_{j,q} = \frac{- a_{j,h}a_{i,q} + a_{i,h}a_{j,q}}{t}$$

fpour

$a_{i,h}:=0$

Le nombre d'opérations arithmétiques que nécessite l'algorithme séquentiel est égal à $T(1)=4n^2(m-\frac{n}{3})+o(mn^2)$ flops avec $mn-\frac{n(n+1)}{2}$ racines carrées.

Soit D_k l'ensemble des rotations disjointes au k-ème pas. On pose $p_k=2\sum_{(i,j,h)\in D_k}(n-h)$

Lemme 2: En utilisant p_k processeurs, $A_{k+1}=Q_kA_k$ est déterminée en T=4 flops.

Démonstration

De la relation (1), on définit $A_{k+1} = \prod_{(i,j,h)\in D_k}(P_{i,j}^h A_k)$. Les produits $P_{i,j}^h A_k$ pour (i,j,h) éléments de D_k sont indépendants donc peuvent être effectués simultanément. Nous réalisons chacun des produits $P_{i,j}^h A_k$ en 4 flops, en utilisant 2(n-h)+2 processeurs, de la façon suivante:

1ere étape: On calcule $a^2_{i,h}$, $a^2_{j,h}$ simultanément avec les produits $a_{i,h}a_{i,q}$ et $a_{j,h}a_{j,q}$ pour $h<q\leq n$ en utilisant 2(n-h)+2 processeurs.

2ième étape: On calcule la somme $a^2_{i,h}+a^2_{j,h}$ simultanément avec les produits $a_{i,h}a_{j,q}$ et $a_{j,h}a_{i,q}$ pour $h<q\leq n$ en utilisant 2(n-h)+1 processeurs.

3ième étape: On calcule $t=\sqrt{a_{i,h}^2 + a_{j,h}^2}$ simultanément avec les sommes $-a_{j,h}a_{i,q} + a_{i,h}a_{j,q}$ et $a_{i,h}a_{i,q} + a_{j,h}a_{j,q}$ pour $h<q\leq n$ en utilisant 2(n-h)+1 processeurs.

4ième étape: On effectue simultanément les divisions $a_{j,q} = \frac{- a_{j,h}a_{i,q} + a_{i,h}a_{j,q}}{t}$ et $a_{i,q} = \frac{a_{i,h}a_{i,q} + a_{j,h}a_{j,q}}{t}$ pour $h<q\leq n$ en utilisant 2(n-h) processeurs.

Par conséquent, au k-ème pas, avec p_k processeurs on peut déterminer tous les produits $P^h_{i,j}A_k$ pour (i,j,h) éléments de D_k en 4 flops.

En utilisant le lemme 2, il est facile de prouver le résultat suivant:

Théorème 2: Le temps d'exécution d'un algorithme permettant de calculer la décomposition de Givens d'une matrice rectangulaire de taille (m,n) en r pas contenant chacune r_k rotations indépendantes, ést égal à T(p)=4r flops en utilisant $p= \max_{1 \le k \le r} (p_k)$ processeurs.

Corollaire 5: Pour m=n le temps d'exécution de l'algorithme de Sameh et Kuck est égal à T(p)=8n-12 flops en utilisant $p= \frac{n(3n-2)}{4}$ processeurs.

Corollaire 6: Pour m=n le temps d'exécution de l'algorithme glouton est égal à T(p)=8n-o(n) en utilisant p=n(n-1) processeurs.

La preuve des corollaire 5 et 6 est identique à celle des corollaires 3 et 4. De même dans le cas des matrices rectangulaires.

2.3. Résultats d'optimalité

Théorème 3: Soit r_{opt} le nombre optimal de pas. Le temps d'exécution optimal T_{opt} pour calculer la décomposition de Givens d'une matrice rectangulaire de taille (m,n) vérifie la relation $T_{opt} \le 4r_{opt}$.

Remarquons que le calcul de la rotation semble nécessiter en parallèle au moins 4 opérations. Nous en déduisons la conjecture suivante: $T_{opt}=4r_{opt}$.

Si la conjecture est exacte alors:
(i) L'algorithme glouton est optimal.
(ii) L'algorithme de Sameh et Kuck est asymptotiquement optimal.

IV.3. CAS D' UNE ARCHITECTURE DE TYPE MIMD

Nous supposons que le nombre de processeurs est limité à O(n). Plus précisément, nous poserons $p=\alpha n$, avec $\alpha \leq 1/2$. Saad[Saa85] montre comment les temps de communication prédominent pour résoudre un problème matriciel de taille n avec n^2 processeurs. Pour rester réaliste nous supposons que nous disposons d'une architecture multiprocesseurs à mémoire partagée de type MIMD donc un fonctionnement asynchrone contrairement au cas précédent. La seule synchronisation est donnée par les contraintes de précédence imposées par l'algorithme. Nous supposons que chaque processeur génére une rotation et l'applique aux deux lignes qui la composent. Nous supposons que le temps de communication est nul ce qui n'est pas de nature à modifier les résultats de complexité. En effet comme les échanges se font par l'intermédiaire de la mémoire partagée, le temps de transfert d'une donnée est indépendant de la localisation des processeurs. De plus nous supposons qu'il n'y a pas de conflit d'accès à la mémoire partagée.

Soient R(i,j,k), $i \neq j$, $1 \leq i, j \leq n$ et $1 \leq k < i$, la rotation dans le plan (i,j) pour annuler l'élément en position (i,k) et T(1) le nombre d'opérations que nécessite la méthode séquentielle de Givens. Le temps d'exécution de la rotation R(i,j,k) est égal à a(n-k)+b flops. Si la racine carrée est évitée alors a=4, b=8 et $T(1)=4n^3/3+o(n^2)$, sinon a=6, b=5 et $T(1)=2n^3+o(n^2)$.

Très peu de résultats de complexité sont apparus pour ce modèle. Lord, Kowalik et Kumar [LKK] ont proposé deux ordonnancements pour calculer la décomposition de Givens pour des matrices carrées de taille n en utilisant le principe d'élimination de l'algorithme de Sameh et Kuck.

* Le premier ordonnancement est en $T(p)=\frac{3an^2}{2}+o(n^2)$ en utilisant $p=\lfloor \frac{n}{2} \rfloor$ processeurs.
* Le deuxième ordonnancement permet de calculer la décomposition de Givens lorsque $p \leq \lfloor \frac{n}{2} \rfloor$. Supposons que p divise n, alors pour $p=\alpha n$, avec $\alpha \leq 1/2$ on a

$T(p)=a(\frac{6\alpha^3+\alpha^2+3\alpha+2}{6\alpha})n^2+o(n^2)$ voir [Kum]

Dans ce chapitre nous proposons un ordonnancement qui améliore les temps d'exécutions proposés dans la littérature. Pour $\frac{n}{3} \leq p \leq \frac{n}{2}$ nous obtenons:

$T(p)=\frac{a}{2}(3n^2 - p^2) + o(n^2)$.

3.1. Etude à l'aide du grâphe des tâches

Soit $T^k_{i,j}$ la tâche qui effectue la rotation de Givens R(i,j,k) pour annuler l'élément en position (i,k). La tâche $T^k_{i,j}$ est caractérisée par:

$E_{T^k_{i,j}} = \{ a_{i,h} / k \le h \le n \} \cup \{ a_{j,h} / k \le h \le n \}$ est l'ensemble des entrées de $T^k_{i,j}$

$S_{T^k_{i,j}} = \{ a_{i,h} / k < h \le n \} \cup \{ a_{j,h} / k \le h \le n \}$ est l'ensemble des sorties de $T^k_{i,j}$

$W(T^k_{i,j}) = a(n-k)+b$ flops est le temps d'exécution de $T^k_{i,j}$.

Chaque processeur commence l'exécution dès qu'il est en possession d'une tâche. Donc la seule synchronisation est due aux contraintes de précédence imposées par l'algorithme. Par conséquent le temps d'exécution est celui du plus long chemin dans le graphe de précédence. Comme le choix des rotations n'est pas unique, on ne connait pas à priori le choix des rotations qui permet d'exécuter l'algorithme en temps minimum. Nous supposons qu'on annule colonne par colonne, informellement, la méthode de Givens est accomplie par l'algorithme séquentiel suivant:

Algorithme 1

Pour k=1 **à** n-1 **faire**

-- Choisir (n-k) rotations parmi les R(i,j,k), $i \neq j$, $1 \le i,j \le m$ et $1 \le k < i$

Exécuter séquentiellement les (n-k) rotations choisies.

fpour

Remarque: Le temps d'exécution dépend essentiellement du choix des rotations. Le choix de R(i,i-1,k) pour annuler l'élément en position (i,k) permet d'exécuter l'algorithme de Sameh et Kuck. Par contre le choix qui permet d'exécuter l'algorithme glouton n'est pas explicite. Comme le montrent les exemples illustrés sur les figures 2 et 3, pour un algorithme glouton avec des choix différents on n'a pas le même temps d'exécution. Par conséquent le temps minimum pour exécuter l'algorithme glouton reste inconu.

Les figures 2 et 3 illustrent deux graphes de précédence de l'algorithme glouton dans le cas où n=6. Les sommets du graphe, notés par (i,j,k), représentent les tâches. Les rotations sont effectuées dans le même ordre que celui de la figure 1.

*					
3	*				
2	5	*			
1	4	6	*		
1	3	5	7	*	
1	2	4	6	8	*

Figure 1: Ordre d'élimination de l'algorithme glouton

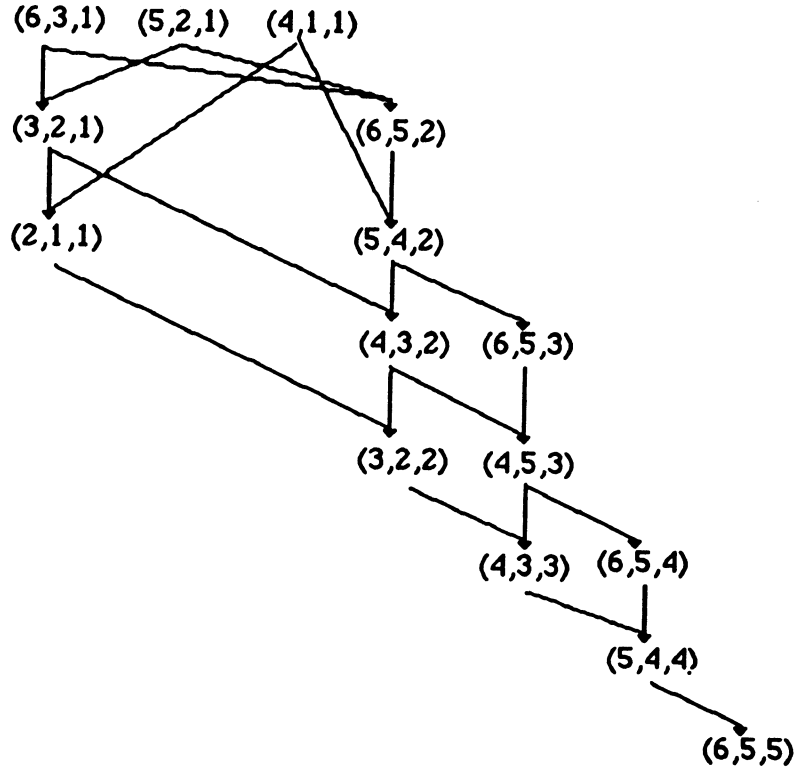


Figure 2: Un graphe de précédence de l'algorithme glouton

Le plus long chemin S_1 dans le graphe est composé des tâches:

$$T^1_{6,3}\ ,\ T^2_{6,5}\ ,\ T^2_{5,4}\ ,\ T^2_{4,3}\ ,\ T^2_{3,2}\ ,\ T^3_{4,3}\ ,\ T^4_{5,4}\ ,\ T^5_{6,5}$$

Le temps d'exécution est égal à la somme des temps d'exécutions des tâches éléments de S_1 donc égal à $L(S_1)= 27a+8b$.

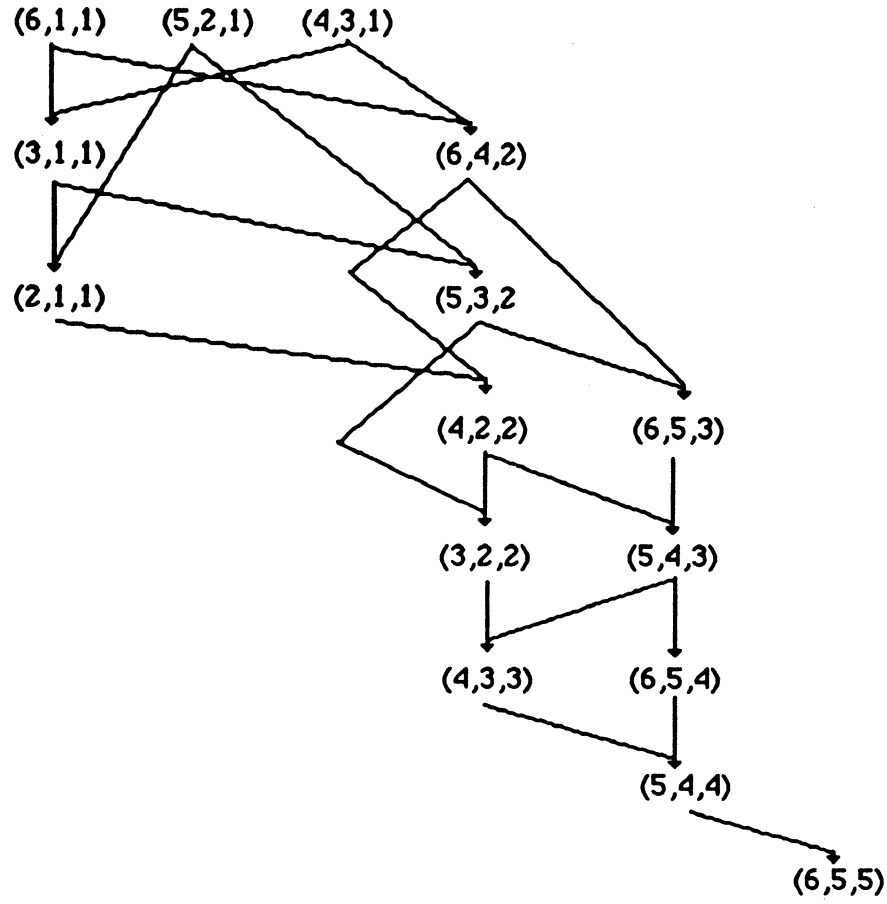


Figure 3: Un autre graphe de précédence de l'algorithme glouton

Le plus long chemin S_1 dans le graphe est composé des tâches:

$$T^1_{6,1}\ ,\ T^1_{3,1}\ ,\ T^1_{2,1}\ ,\ T^2_{4,2}\ ,\ T^2_{3,2}\ ,\ T^3_{4,3}\ ,\ T^4_{5,4}\ ,\ T^5_{6,5}$$

Le temps d'exécution est la somme des temps d'exécution des tâches éléments de s_1 donc égal à $L(S_1)=29a+8b$.

3.2. Un algorithme pour $\frac{n}{3} \leq p \leq \frac{n}{2}$

Soient A une matrice carrée de taille n et p le nombre de processeurs avec $\frac{n}{3}\leq p\leq\frac{n}{2}$. Nous posons $n=2p+cp$ avec $0\leq c \leq 1$.

Proposition 1: Le temps d'exécution T(p) pour calculer la décomposition de Givens de A avec p processeurs, $\frac{n}{3}\leq p\leq\frac{n}{2}$, est égal à:

(i) $T(p)=3(3n^2-p^2)+o(n^2)$ pour la méthode standard de Givens

(ii) $T(p)=2(3n^2-p^2)+o(n^2)$ pour la méthode de Givens sans racine carrée.

Le processus d'élimination consiste à:

a. Décomposer la matrice en deux blocs: le premier est composé des p premières lignes. Le deuxième bloc est composé des (1+c)p dernières lignes.

b. Eliminer dans chaque bloc par colonne et de bas en haut. Pour chaque colonne nous commençons par annuler les éléments situés dans le premier bloc puis on termine par ceux dans le deuxième bloc. Pour annuler l'élément en position (i,k), i≠p+1, nous utilisons la rotation R(i,i-1,k), et pour annuler l'élément en position (p+1,k) nous utilisons la rotation R(p+1,k,k).

Algorithme 2

```
Pour k = 1 à (n-1) faire
    pour i=p à (k+1) faire          * Annuler la colonne k du 1er bloc*
        R(i,i-1,k)
    fpour
    pour i=n à max(p+1,k)+1 faire   * Annuler la colonne k du 2-ème bloc*
        R(i,i-1,k)
    fpour
    Si (p+1)> k alors R(p+1,k,k)
fpour
```

Pour simplifier notons par $T_{i,k}$ la tâche qui effectue la rotation R(i,i-1,k) et par T^k_{p+1} la tâche qui effectue la rotation R(p+1,k,k). Donc l'ensemble J des tâches qui définissent l'algorithme séquentiel est égal à:

$$J=\{T_{i,k} / i\neq p+1,\ 1 \le i \le n,\ k \le \min(i-1,n)\} \cup \{T^k_{p+1} / 1 \le k < p+1\}$$

Les contraintes de précédence imposées par l'algorithme séquentiel sont définies:

<u>* Dans le premier bloc par</u>:

(A) $T_{i,k} \rightarrow T_{i-1,k}$ pour $2 \le i \le p$ et $k < i-1$

(B) $T_{k+1,k} \rightarrow T_{p,k+1}$

<u>* Dans le deuxième bloc par:</u>

(C) $T_{i,k} \rightarrow T_{i-1,k}$ pour $p+3 \le i \le n$ et $k < i-1$

$T_{p+2,k} \rightarrow T^k_{p+1}$

(D) $T_{k_1,k} \rightarrow T_{n,k+1}$ pour $k_1=\max(p+1,k)$

* Pour annuler l'élément en position (p+1,k) on utilise la rotation R(p+1,k,k)

(E) $T_{k+1,k} \rightarrow T^k_{p+1}$ pour $1 \le k \le p$

- (A) et (C) sont imposées par le choix de R(i,i-1,k) pour annuler en position (i,k) et par le choix de R(p+1,k,k) pour annuler en position(p+1,k).
- (B) et (D) sont imposées par le choix d'annuler colonne par colonne et de bas en haut.
- (E) est imposée par le choix de R(p+1,k,k) pour annuler en position (p+1,k)

Remarque: L'ordre d'élimination dans chaque bloc est le même que celui de l'algorithme de Sameh et Kuck.

Nous déduisons donc à partir de la méthode séquentielle décrite par l'algorithme 2 un schéma d'élimination parallèle basé sur le même principe d'élimination que l'algorithme de Sameh et Kuck. Les rotations sont effectuées dans le même ordre que montre la figure 4 dans le cas où n=12 et p=4.

```
*
3                 *
2                 4    *
1                 3    5    *
--------------------
8                10   12   14    *
7                 9   11   13   15    *
6                 8   10   12   14   16    *
5                 7    9   11   13   15   17    *
4                 6    8   10   12   14   16   18    *
3                 5    7    9   11   13   15   17   19    *
2                 4    6    8   10   12   14   16   18   20    *
1                 3    5    7    9   11   13   15   17   19   21    *
```

Figure 4: Ordre d'élimination parallèle des éléments

Remarque: Le nombre de pas que nécessite l'algorithme parallèle est égal à 2n-3 qui est le même que celui de l'algorithme de Sameh et Kuck.

* Dans le premier bloc nous utilisons l'algorithme de Sameh et Kuck donc les tâches $\{T_{i,k} / p+2k-i-1=r, 1\leq i\leq p, k<i \text{ et } 1\leq r\leq 2p-3\}$ sont indépendantes.

* Dans le deuxième bloc on annule les éléments en position (p+1,k), $1\leq k<p+1$, en utilisant la rotation R(p+1,k,k), et pour les autres éléments on annule avec le même principe que l'algorithme de Sameh et Kuck donc les tâches
$\{T_{i,k} / n+2k-i-1=r, p+1<i\leq n, k< i, 1\leq r\leq 2n-3\} \cup$
$\{T^{k}_{p+1} / n+2k-p-2=r, 1 \leq k < p+1, n-p\leq r\leq n+p-2\}$ sont indépendantes.

Soit d_r l'ensemble des tâches indépendantes au pas r, on a donc:

$$d_r=\{T_{i,k} / n+2k-i-1=r,\ 1\le i\le p,\ k< i\} \cup \{T_{i,k} / n+2k-i-1=r,\ p+1<i\le n,\ k\le i-1\} \cup \{T^k_{p+1} / n+2k-p-2=r,\ 1 \le k < p+1\}$$

Les tâches éléments de d_r sont indépendantes donc peuvent être exécutées simultanément. On définit donc le système des tâches qui représentent l'algorithme parallèle par C=(J,<<). Les contraintes de précédence sont définies par:

(A)' $T_{i,k} << T_{i-1,k}$ pour $2\le i\le n$ et $k\le i-1$, $i\neq p+2$ et $i\neq p+1$
$T_{p+2,k} << T^k_{p+1}$

(B)' $T_{i,k} << T_{i+1,k+1}$ $i\neq p+1$ et $i\neq p$
$T^k_{p+1} << T_{p+2,k+1}$

(C)' $T_{k,k+1} << T^k_{p+1}$ pour $1 \le k < p+1$

- (A)' sont dues au choix de R(i,i-1,k) pour annuler en position (i,k)
- (B)' sont dues au choix d'annuler colonne par colonne
- (C)' est imposée par le choix de R(p+1,k,k) pour annuler en position (p+1,k)

On obtient donc le graphe des tâches représenté sur la figure 5 dans le cas où n=9 et p =3. Pour simplifier, nous avons représenté les tâches $T_{i,k}$ et T^k_{p+1} par (i,k) et par (p+1,k). Les sommets du graphe sont les tâches et les arcs représentent la relation de précédence.

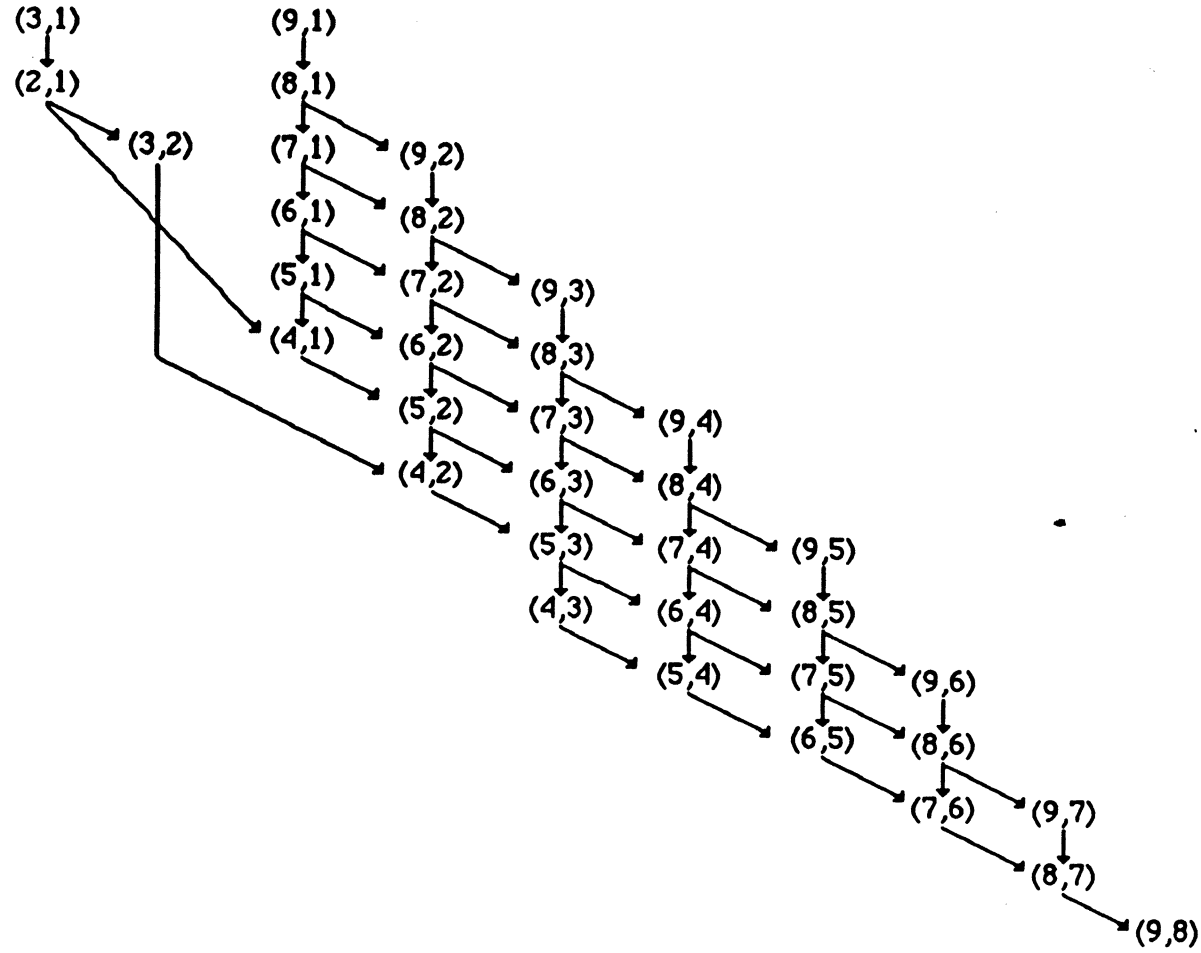


Figure 5: Graphe des tâches pour n=9 et p=3

Le plus long chemin est composé des tâches éléments de S_1

$S_1=\{T_{i,1} / p+2 < i \leq n\} \cup \{T^k_{p+1}, T_{p+2,k} / 1 \leq k \leq p\} \cup \{T_{k+1,k} / p+1 \leq k \leq n-1\}$

Le temps d'exécution T(p) est égal à la somme des temps d'exécutions des tâches qui composent S_1 on obtient donc:

$$T(p) = \sum_{k=1}^{n-p-2} [\, a(n-1) + b \,] + 2\sum_{k=1}^{p} [\, a(n-k) + b \,] + \sum_{k=p+1}^{n-1} [\, a(n-k) + b \,]$$

$$= \frac{a}{2}(3n^2 - p^2) + o(n^2)$$

Posons $p=\alpha n$ avec $\frac{1}{3} \leq \alpha \leq \frac{1}{2}$ on obtient donc:

$T(p)= \frac{an^2}{2}(3 - \alpha^2)+o(n^2)$ avec une efficacité égale à $E(\alpha)= \frac{2}{3\alpha(3 - \alpha^2)}$

Remarques: Pour ce graphe on a:

(i) T(p) est minimale pour $p=\frac{n}{2}$, dans ce cas $T(\frac{n}{2})= \frac{11an^2}{8} + o(n^2)$ et $E(\frac{n}{2})=0.484$

(ii) E(p) est maximale pour $p=\frac{n}{3}$, dans ce cas $T(\frac{n}{3})=\frac{13an^2}{9}+o(n^2)$ et $E(\frac{n}{3})=0.692$

Affectation des tâches aux processeurs

Pour parvenir à exécuter l'algorithme présenté ci dessus avec $\frac{n}{3}\leq p\leq\frac{n}{2}$ processeurs, nous avons distribué les tâches aux processeurs comme le montre la figure 6 dans le cas où n=12 et p=4. D'une manière générale chaque processeur exécute les tâches qui annulent sur deux lignes consécutives. On désigne par P_i le processeur de numéro i. Chaque processeurs P_i exécute les tâches suivantes (Pour simplifier nous avons aussi noté la tâche T^k_{p+1} par $T_{p+1,k}$):

$i\leq \frac{p}{2}$ $\{T_{n-2i+2,k}\ ,\ T_{n-2i+1,k}\ /\ k \leq n-2(i-1)+1\}$

$\frac{p}{2}\leq i$ $\{T_{2i-p,k},\ T_{2i-p-1,k}\ /\ 1\leq k\leq 2i-p-1\}\cup\{T_{n-2i+2,k},\ T_{n-2i+1,k}/\ k\leq n-2i+1,\ p<n-2i+1\}$

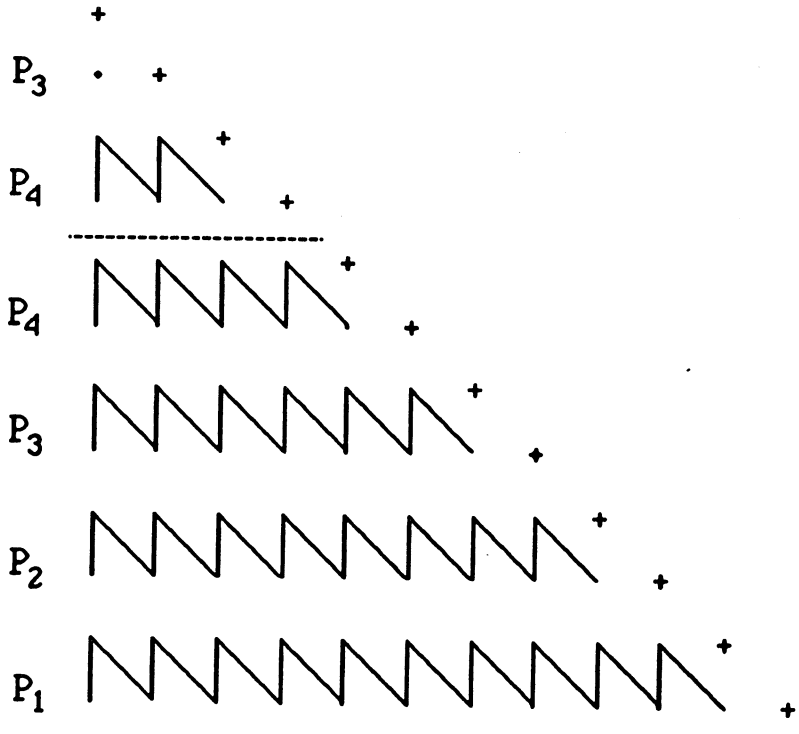


Figure 6: Schéma d'affectation des processeurs pour n=12 et p=4

3.3. Un algorithme pour p quelconque

Nous proposons un ordonnancement pour calculer la décomposition de Givens d'une matrice carrée de taille n avec p quelconque. Posons n=2kp+cp avec k∈ IN* et 0≤c<2. Deux cas seront étudiés séparément.

Premier cas: 0≤c≤1

Pour annuler nous procèdons de la façon suivante:

a. Nous décomposons la matrice en (k+1) blocs. Le premier bloc est composé des cp premières lignes. Les k blocs suivants sont composés chacun de 2p lignes. Nous annulons dans chaque bloc colonne par colonne et de bas en haut.

b. Pour annuler un élément en position (i,k), i≠(2j-2+c)p+1 avec 1≤j≤k, nous utilisons la rotation R(i,i-1,k) et pour annuler l'élément en position ((2j-2+c)p+1,k) nous utilisons la rotation R((2j-2+c)p+1,k,k).

La décomposition de Givens est accomplie par l'algorithme séquentiel suivant:

Algorithme 3

```
Pour col = 1 à (n-1) faire          * Boucle sur les colonnes*
    pour i=cp à col+1 faire              * Premier bloc*
        R(i,i-1,col)
    fpour
    pour j=1 à k faire                   * Boucles sur les blocs suivants*
        pour i=(2j+c)p à (2j-2+c)p +2 faire
            R(i,i-1,col)
        fpour
        R((2j-2+c)p+1,k,k)
    fpour
fpour
```

Soit $T_{i,k}$ la tâche qui effectue la rotation R(i,i-1,k), pour i≠(2j-2+c)p+1 avec 1≤j≤k. Pour simplifier notons aussi par $T_{(2j-2+c)p+1,k}$ la tâche qui effectue la rotation
R((2j-2+c)p+1,k,k)

Le principe d'élimination dans chaque bloc est le même que celui de Sameh et Kuck. Dans chaque bloc j, j>1, l'élimination évolue en 3 phases. Les rotations sont effectuées dans le même ordre que celui de la figure 7 dans le cas où n=20 et p=4.

```
*
3  *
2  4    *
1  3     5  *
---------------------
8  10   12  14  *
7   9   11  13  15  *
6   8   10  12  14  16  *
5   7    9  11  13  15  17  *
4   6    8  10  12  14  16  18  *
3   5    7   9  11  13  15  17  19  *
2   4    6   8  10  12  14  16  18  20  *
1   3    5   7   9  11  13  15  17  19  21  *
-------------------------------------------------------------
23  25  27  29  31  33  35  37  39  41  43  45  *
22  24  26  28  30  32  34  36  38  40  42  44  46  *
21  23  25  27  29  31  33  35  37  39  41  43  45  47  *
20  22  24  26  28  30  32  34  36  38  40  42  44  46  48  *
19  21  23  25  27  29  31  33  35  37  39  41  43  45  47  49  *
18  20  22  24  26  28  30  32  34  36  38  40  42  44  46  48  50  *
17  19  21  23  25  27  29  31  33  35  37  39  41  43  45  47  49  51  *
16  18  20  22  24  26  28  30  32  34  36  38  40  42  44  46  48  50  52  *
```

Figure 7: Ordre d'élimination de l'algorithme parallèle

* La première phase débute au temps d_j et se termine au temps $t_j=d_j+(2p-3)$ (temps d'exécution de la première phase de l'algorithme de Sameh et Kuck).

* La deuxième phase débute au temps t_j+1 et se termine au temps

$$f_j = t_j+4(j-1)p+2cp = d_j+4jp+2(c+1)p.$$

* La troisième phase débute au temps f_j+1 et se termine au temps f_j+2p, et consiste à annuler par la deuxième phase de l'algorithme de Sameh et Kuck. Par conséquent au temps $d_{j+1}=f_j+2$ on peut commencer la première phase du bloc (j+1).

A partir des relations $f_j = d_j+4jp+2(c+1)p$, $d_{j+1}=f_j+2$ et de $T(p) = f_k+2p$ nous en déduisons que le nombre de pas que nécessite l'algorithme parallèle est égal à $T(p)=2n+(k-1)(2kp-2(1-c)p-1)-3$.

Pour simplifier supposons que p divise n donc c=0 ou c=1. Dans ces deux cas on obtient:

* $T(p)=\frac{n^2}{2p} + 2p + o(n)$ pour c=0

* $T(p)=\frac{n^2}{2p} + \frac{3}{2}p + o(n)$ pour c=1

Affectation des tâches aux processeurs

Pour parvenir à exécuter cet algorithme nous avons distribué les tâches aux processeurs comme le montre la figure 8 dans le cas où n=20 et p=4:

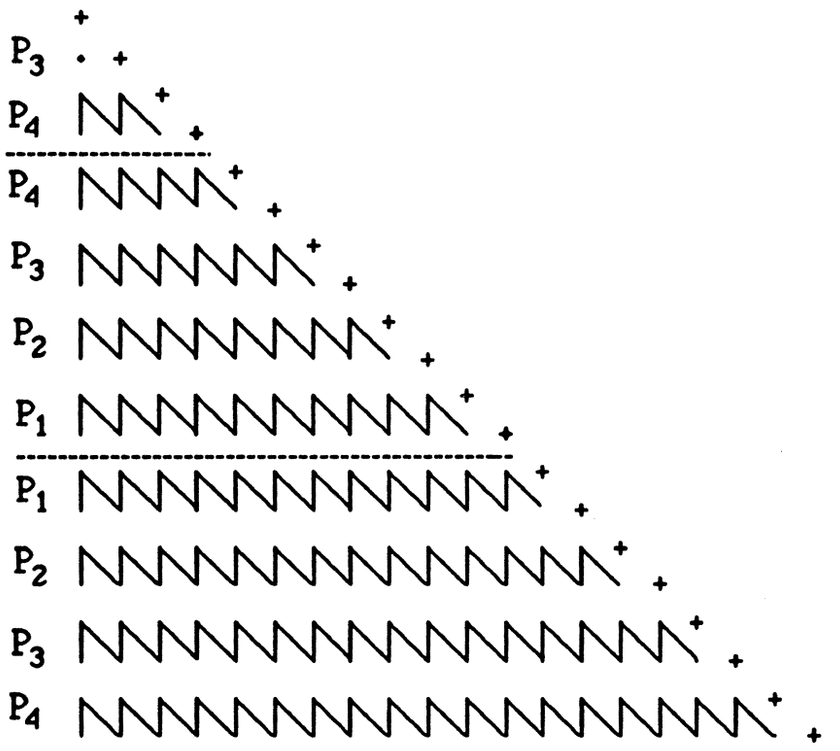


Figure 8: Schéma d'affectation des processeurs

D'une manière générale dans chaque bloc j le processeur P_i exécute les tâches suivantes:

* j pair

$$\{T_{2(j-1)p+cp+i+1,h}\ ,\ T_{2(j-1)p+cp+i+2,h} \,/\, h \le 2(j-1)p+cp+i+1,\ 1\le j\le k\}$$

* j impair

$$\{T_{2jp+cp-2i+2,h}\ ,\ T_{2jp+cp-2i+1,h} \,/\, h \le 2jp+cp-2i+1,\ 1\le j\le k\}$$

* Pour $i \geq \frac{p}{2}$, le processeur P_i exécute en plus les tâches suivantes (tâches qui annulent dans le premier bloc)

$\{T_{cp-2p+2i,h}, T_{cp-2p+2i-1,h}, / 1 \leq h \leq cp-2p+2i-1\}$

Suivant cette affectation le plus long chemin S_1 est composé des tâches suivantes:

$S_1=\{ C_j / 1 \leq j \leq k+1 \}$ où

$C_j= \{T_{i,1} / (2j+c)p \leq i \leq (2j-2+c)p+2 \} \cup \{T_{(2j-2+c)p+1,k} T_{(2j-2+c)p+2,k}\}$

$$\text{D'où } T(p)=\sum_{i=1}^{n-cp} (a(n-1)+b) + \sum_{j=1}^{k} \left(2 \sum_{i=1}^{(2j-2+c)p} (a(n-i)+b) \right) + \sum_{i=(2k-2+c)p+1}^{n-1} (a(n-i)+b)$$

$$T(p)=a(\frac{3}{2}n^2+2(k-1)(k+c-1)np- \frac{p^2}{6} (8k^3-12k^2(2-c)+6k(2-c)^2+4k-3(2-c)^2)$$

Supposons que p divise n et posons $p=\alpha n$ alors le temps d'exécution est égal à:

$T(p)=an^2(12\alpha^3-2\alpha^2+3\alpha+2)/6\alpha$ pour c=0

$T(p)=an^2(12\alpha^3-5\alpha^2+3\alpha+2)/6\alpha$ pour c=1

Deuxième cas:1<c<2

Une solution simple consiste à annuler tous d'abord dans le premier bloc et ne commencer à éliminer dans le deuxième bloc qu'au temps $d_1=cp$ (temps d'exécution de la première colonne du premier bloc). Pour le reste c'est le même principe que le cas étudié précédemment. L'ordre d'élimination est illustré sur la figure 9 pour n=20 et p=4. Dans ce cas, on a c=1.5.

De la même façon que le cas étudié précédemment nous déduisons que le temps d'exécution est égal à:

$$T(p)=a(\frac{3}{2}n^2+(2(k-1)(k+c-1)+c)np- \frac{p^2}{6} (8k^3-12k^2(2-c)+6k(2-c)^2+4k-3(2-c)^2)$$

```
*
5  *
4   6  *
3   5   7  *
2   4   6   8  *
1   3   5   7   9  *
---------------------------------
13  15  17  19  21  23  *
12  14  16  18  20  22  24  *
11  13  15  17  19  21  23  25  *
10  12  14  16  18  20  22  24  26  *
9   11  13  15  17  19  21  23  25  27  *
8   10  12  14  16  18  20  22  24  26  28  *
7   9   11  13  15  17  19  21  23  25  27  29  *
6   8   10  12  14  16  18  20  22  24  26  28  30  *
-------------------------------------------------------------------------
32  34  36  38  40  42  44  46  48  50  52  54  56  58  *
31  33  35  37  39  41  43  45  47  49  51  53  55  57  59  *
30  32  34  36  38  40  42  44  46  48  50  52  54  56  58  60  *
29  31  33  35  37  39  41  43  45  47  49  51  53  55  57  59  61  *
28  30  32  34  36  38  40  42  44  46  48  50  52  54  56  58  60  62  *
27  29  31  33  35  37  39  41  43  45  47  49  51  53  55  57  59  61  63  *
26  28  30  32  34  36  38  40  42  44  46  48  50  52  54  56  58  60  62  64  *
25  27  29  31  33  35  37  39  41  43  45  47  49  51  53  55  57  59  61  63  65 *
```

Figure 9: Ordre d'élimination dans le cas où n=15 et p=3

3.4. Un modèle synchrone

Dans ce paragraphe nous supposons que nous synchronisons tous les processeurs à la fin de chaque pas (ensemble de rotations disjointes). Donc on ne peut commencer l'exécution des tâches du pas suivant que lorsque l'exécution de toutes les tâches du pas en cours soit terminée. Soit $R(i_1,j_1,k_1)$, $R(i_2,j_2,k_2)$, ..., $R(i_q,j_q,k_q)$, avec $q \leq p$, l'ensemble des rotations disjointes au r-ème pas, donc le temps d'exécution de ce pas est égal au temps d'exécution de la tâche la plus longue donc égal à: $a(n-\min(k_i))+b$, pour $1 \leq i \leq q$.

Nous supposons que les éléments sont annulés de bas en haut et de gauche à droite. Soit r_k le pas auquel la colonne k est annulée (le pas auquel un zéro est introduit en position (k+1,k)). Le temps d'exécution T est celui pour annuler tous les éléments en positions (k+1,k) $1 \leq k \leq n-1$, donc égal à:

$$T=r_1(a(n-1)+b) + \sum_{k=2}^{n} (r_k - r_{k-1})(a(n-k)+b) = a(n-1)r_1 + br_n + a \sum_{k=2}^{n} (n-k)(r_k - r_{k-1})$$

Dans le corollaire ci dessous nous présentons les temps d'exécution des algorithmes présentés dans le chapitre II.

Corollaire [CDMR] :

1. L'algorithme de Sameh et Kuck: $r_1 = n-1$, $r_k - r_{k-1}=1$ et $T=3an^2/2 + o(n^2)$
2. L'algorithme de Fibonacci d'ordre 1: $r_1 = \sqrt{2n} + o(n)$, $r_k - r_{k-1}=2$ et $T=an^2 + o(n^2)$
3. L'algorithme glouton: $r_1 = \log_2(n)$, $r_k - r_{k-1}=2+\delta_k$ et $T=an^2 + o(n^2)$

Comme $T_{opt}=2n-o(n)$ on déduit que $\delta_2 + \ldots + \delta_n = o(n)$

IV.4. CONCLUSION

Nous avons analysé la complexité de la décomposition de Givens d'une matrice carrée de taille n sur deux types d'architectures à mémoire partagée composée de p processeurs.

Dans le premier type d'architecture nous avons supposé que: le nombre de processeurs est non limité donc une granularité trés fine, les processeurs travaillent en mode synhrone. Donc chaque rotation est exécutée par O(n) processeurs. Nous avons proposé un algorithme qui est en 8n -o(n) comparé à 12n-o(n) proposé dans [SK78]. Le calcul de la rotation semble nécessiter en parallèle au moins 4 opérations (nous supposons que l'évaluation de la racine carrée est équivalente à une opération élémentaire). Nous en déduisons la conjecture suivante: $T_{opt}=4r_{opt}$ où r_{opt} est le nombre optimal de pas. Le résultat obtenu s'étend au cas des matrices rectangulaires.

Dans le deuxième type d'architecture nous avons supposé que nombre de processeurs est limité, que les processeurs travaillent en mode asynchrone. A l'aide du formalisme du graphe des tâches nous avons proposé un ordonnancement qui améliore le temps d'exécution des algorithmes proposés dans la littérature. Dans le paragraphe 3.4 nous avons supposé que les processeurs se synchronisent à la fin de chaque pas. Dans ce cas le temps d'exécution de l'algorithme glouton est égal à $an^2+o(n^2)$. Plusieurs questions restent ouvertes:

(i) Quel est le temps optimal pour calculer la décomposition de Givens ? (Réponse $an^2+o(n^2)$?)

(ii) Etant donnée n et p quel est le temps optimal avec p processeurs ?

DEUXIEME PARTIE

COMPLEXITE DE LA DECOMPOSITION ORTHOGONALE SUR UNE ARCHITECTURE A MEMOIRE DISTRIBUEE

CHAPITRE V

INFLUENCE DES COÛTS DE COMMUNICATION

V.1. INTRODUCTION

Dans ce chapitre nous comparons la complexité de la méthode de Givens et de la méthode de Householder pour calculer la décomposition QR d'une matrice A carrée de taille n sur différents modèles d'architectures parallèles. L'approche que nous adoptons ici prend en compte le fait que les transferts des données entre processeurs ne sont pas négligeables devant le temps de calcul. L'unité de temps est prise égale au temps nécessaire à la lecture des données, le traitement local et l'écriture des nouvelles valeurs. Notre but et de montrer l'influence de l'architecture, en particulier des connexions entre processeurs, sur la complexité des algorithmes parallèles. Remarquons, de plus, que l'obtention des résultats de compléxité est assez rare, et nécessite en général des modèles simplifiés.

Sameh [Sam85, 82] propose plusieurs algorithmes pour résoudre des problèmes d'algèbre linéaire sur un réseau linéaire et sur un anneau de processeurs. Plusieurs méthodes de résolution de systèmes linéaires sont présentés dans [ISS] sur un anneau de processeurs, disposant d'un bus de communication. Dans ce modèle, le nombre de processeurs est petit devant la taille du problème. Dans [GVR], l'influence des communications sur la conception des algorithmes parallèles est étudiée dans le cas d'architecture à mémoire partagée et de réseau de processeurs à connexions locales. Y.Saad [Saad85], examine les possibilités de communication de plusieurs architectures possédant un grand nombre de processeurs par rapport à la taille n du problème, en prenant comme algorithme test la méthode de Gauss. Son résultat principal montre que la prise en compte des communications ne permet pas d'obtenir des temps d'exécution linéaire, car le temps de communication est de l'ordre de $O(n^2)$ quel que soit le nombre de processeurs. Dans [Saa86], il étudie l'implémentation de l'algorithme de Gauss sur un hypercube de processeurs. Il montre que les sous réseaux d'anneau et de grille bidimensionnelle permettent de résoudre efficacement le problème en dupliquant des lignes de la matrice (grille) ou en les pipelinant (anneau). C'est ce dernier principe que nous utilisons pour obtenir des algorithmes optimaux.

Nous présentons des résultats obtenus dans [CDT] qui comparent la complexité de plusieurs algorithmes d'algèbre linéaire sur un réseau linéaire et sur un anneau de processeurs. Le point commun de ces deux réseaux réside dans le fait que les communications inter-processeurs sont locales et les échanges avec l'extérieur se font par l'intermédiaire d'un seul processeur. Nous montrons que pour ces algorithmes le rôle joué par les communications entre les processeurs et l'hôte est plus important que les communications entre processeurs. Ensuite dans le paragraphe V.4.2 nous montrons comment varie la complexité des algorithmes proposés pour l'anneau lorsqu'on augmente dans l'anneau les moyens de communication.

V.2. RAPPELS

2.1. Méthode de Householder

Nous appelons opérateur d'élimination la transformation de Householder H_k qui permet d'annuler les éléments en position (k+1,k),...,(n,k) [ch.I]. H_k est complètement déterminée par la colonne k qu'on appelle colonne pivot. Par conséquent une même colonne pivot peut servir dans plusieurs éliminations. Mesuré en nombre d'éliminations e le coût de l'algorithme est égal à n(n-1)/2e (ce qui correspond au nombre d'éléments sous diagonaux de A).

L'idée de base pour paralléliser cet algorithme consiste à affecter les colonnes différentes à des processeurs différents, en dupliquant la colonne pivot à chaque étape.

Supposons que l'architecture sous jacente soit une machine SIMD/MIMD avec p processeurs, chaque processeur pouvant être un processeur vectoriel. Ces processeurs communiquent par partage de données grâce à une mémoire commune qu'ils accèdent par un réseau complet d'interconnexions. Rappelons que, pour simplifier l'étude, l'unité de temps correspond à l'exécution des opérations suivantes: lecture de deux colonnes de la matrice, calcul et application de la transformation de Householder et écriture des nouvelles colonnes en mémoire commune. En pratique, le temps d'exécution de toutes ces opérations dépend de la longueur des colonnes en question, mais nous ne prendrons pas ce paramètre en compte. Pour des résultats de l'implémentation de cet algorithme sur des architectures à mémoire partagée nous renvoyons à [LKK], [RTr], [MR] (extension des résultats obtenus pour la méthode de Gauss).

Prenons p=n-1 et considérons l'algorithme HouseholderMP suivant: au temps t=k, pour q variant de k à n-1, le processeur q lit les colonnes k et q+1, calcule et applique H_k à la colonne q puis les range en mémoire commune. Pour n=8, cet algorithme est décrit dans la figure 1. L'entier t est placé en position (i,k) si

l'élément correspondant a été annulé au temps t.

```
*
1  *
1  2  *
1  2  3  *
1  2  3  4  *
1  2  3  4  5  *
1  2  3  4  5  6  *
1  2  3  4  5  6  7  *
```

Figure 1: Ordonnancement des éliminations sur une architecture à mémoire partagée. Algorithme de HouseholderMP

Nous appellerons algorithme optimal un algorithme dont le temps d'exécution est optimal au sens du nombre d'opérations élémentaires définies par la granularité de la décomposition du problème. Le temps d'exécution sera égal à la longueur de la plus longue suite d'opérations élémentaires séquentielles nécessaires à la résolution du problème.

Théorème 1: Sur une architecture à mémoire partagée, l'algorithme HouseholderMP est optimal. Son temps d'exécution est égal à (n-1) e.

Le nombre de processeurs qui permet d'exécuter l'algorirhme HouseholderMP en temps optimal est égal à (n-1).

2.2. Méthode de Givens

Mesuré en nombre de rotations r le coût de la méthode de Givens est égal à n(n-1)/2r (ce qui correspond au nombre d'éléments sous diagonaux de A). Nous considérons la même architecture que dans le cas précédent. Une description détaillée des algorithmes parallèles est présentée dans les chapitres II et III. Rappelons que l'algorithme glouton est optimal son temps d'exécution est égal à (2n-o(n))r. Celui de l'algorithme de Sameh et Kuck est égal à (2n-3)r.

V.3. LES ALGORITHMES SUR UN RESEAU LINEAIRE

L'architecture à mémoire partagée considérée dans la partie précédente peut être vue comme un réseau complet de processeurs. Un tel réseau comporte $O(n^2)$ canaux de communication. De ce point de vue, ce réseau possède des possibilités maximales de communication. Par conséquent, un algorithme pour un réseau particulier de processeurs peut être exécuté sur une architecture à mémoire partagée. Bien entendu l'inverse n'est pas vrai. Le réseau de processeurs possédant

les possibilités minimales de communication est le réseau linéaire. Dans cette partie nous étudions l'implémentation des méthodes précédentes sur un tel réseau.

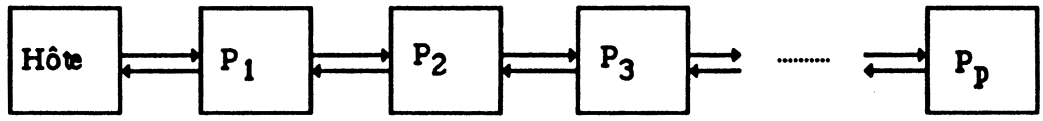


Figure 2 : Réseau linéaire de processeurs

Considérons une architecture multiprocesseur composée de p processeurs connectés entre eux comme le montre la figure 2. Chaque processeur P_i ($2 \leq i \leq p-1$) communique avec P_{i-1} et P_{i+1} par l'intermédiaire d'un canal de lecture et un canal d'écriture. Le processeur P_p communique seulement avec P_{p-1}. Le processeur P_1 communique avec l'ordinateur hôte aussi par un canal de lecture et un canal d'écriture. Chaque processeur possède une mémoire locale pouvant contenir 2 lignes (ou 2 colonnes) de A. Nous supposons que l'unité est le temps nécessaire pour lire une ligne (ou une colonne), effectuer une rotation (ou une transformation) et transmettre une ligne (ou une colonne) quelle que soit la position qu'occupe l'élément à annuler. Au début de l'algorithme, l'hôte contient la matrice A. A la fin, il doit contenir le résultat cherché: la matrice R.

3.1. Méthode de Givens

Nous allons tout d'abord étudier la méthode de Givens sur cette architecture. Nous présentons deux cas suivant le nombre de processeurs.

3.1.a. Cas $p=\lceil \frac{n}{2} \rceil$

Il est clair qu'on ne peut pas implémenter l'algorithme Glouton puisqu'il effectue plusieurs rotations au temps 1. Par contre l'algorithme de Sameh et Kuck s'implémente directement. Nous présentons deux algorithmes proposés dans [CDT].

L'algorithme GivensRL1 consiste en deux phases. Les lignes sont introduites dans le réseau les unes après les autres, de la ligne n à la ligne 1 de la façon suivante:

Première phase: De t=1 à t=n-1, les lignes sont "pipelinées" dans le réseau, chaque ligne restant deux unités de temps dans P_{i-1} avant d'être transférée dans P_i. A chaque instant, P_i lit une ligne à partir de P_{i-1} et effectue la rotation correspondante.

Deuxième phase: De t=n à t=2n-1, le sens de communication est renversé: P_i lit

une ligne à partir de P_{i+1} et effectue la rotation correspondante. Pendant cette phase, le processeur P_1 délivre à chaque unité de temps une nouvelle ligne de la matrice R.

	P_1		P_2		P_3		P_4	
t=1	8							
t=2	78	R(8,7,1)						
t=3	67	R(7,6,1)	8					
t=4	56	R(6,5,1)	78	R(8,7,2)				
t=5	45	R(5,4,1)	67	R(7,6,2)	8			
t=6	34	R(4,3,1)	56	R(6,5,2)	78	R(8,7,3)		
t=7	23	R(3,2,1)	45	R(5,4,2)	67	R(7,6,3)	8	
t=8	12	R(2,1,1)	34	R(4,3,2)	56	R(6,5,3)	78	R(8,7,4)
t=9	23	R(3,2,2)	45	R(5,4,3)	67	R(7,6,4)	8	
t=10	34	R(4,3,3)	56	R(6,5,4)	78	R(8,7,5)		
t=11	45	R(5,4,4)	67	R(7,6,5)	8			
t=12	56	R(6,5,5)	78	R(8,7,6)				
t=13	67	R(7,6,6)	8					
t=14	78	R(8,7,7)						
t=15	8							

Figure 3 : L'algorithme GivensRL1 sur une matrice de taille (8,8)

Remarquons que ce fonctionnement ne nécessite pas un contrôle global puisque le changement de sens de communication s'effectue lorsque le processeur P_i lit la ligne 2i-1. L'algorithme est décrit dans la figure 3. Le temps d'exécution de cet algorithme est égal à (2n-1)r. Les rotations sont effectuées dans le même ordre que dans l'algorithme de Sameh et Kuck. La différence entre les temps d'exécution des deux algorithmes (2r) provient du fait de la lecture et de l'écriture de la ligne n dans l'hôte aux instants t=1 et t=2n-1.

L'algorithme GivensRL2 consiste lui aussi en deux phases. Les lignes sont introduites dans le réseau les unes après les autres, de la ligne 1 à la ligne n de la façon suivante:

Première phase: De t=1 à t=n-1, P_i lit une ligne à partir de P_{i-1} et effectue la rotation correspondante. Lorsque la ligne i arrive dans P_i, au temps (2i-1)r, celui-ci la stocke.

Deuxième phase: De t=n+1 à t=2n-1, le sens de communication est renversé: P_i

lit une ligne à partir de P_{i+1} et effectue la rotation correspondante. Pendant cette phase, le processeur P_1 délivre à chaque unité de temps une nouvelle ligne de la matrice R.

Remarquons que ce fonctionnement ne nécessite pas un contrôle global puisque le changement de sens de communication s'effectue lorsque le processeur P_i lit la ligne (n-i+1). L'algorithme est décrit dans la figure 4. Le temps d'exécution de cet algorithme est aussi égal à 2n r.

	P_1		P_2		P_3		P_4	
t=1	1							
t=2	21	R(2,1,1)						
t=3	31	R(3,1,1)	2					
t=4	41	R(4,1,1)	32	R(3,2,2)				
t=5	51	R(5,1,1)	42	R(4,2,2)	3			
t=6	61	R(6,1,1)	52	R(5,2,2)	43	R(4,3,3)		
t=7	71	R(7,1,1)	62	R(6,2,2)	53	R(5,3,3)	4	
t=8	81	R(8,1,1)	72	R(7,2,2)	63	R(6,3,3)	54	R(5,4,4)
t=9	82	R(8,2,2)	73	R(7,3,3)	64	R(6,4,4)	5	
t=10	83	R(8,3,3)	74	R(7,4,4)	65	R(6,5,5)		
t=11	84	R(8,4,4)	75	R(7,5,5)	6			
t=12	85	R(8,5,5)	76	R(7,6,6)				
t=13	86	R(8,6,6)	7					
t=14	87	R(8,7,7)						
t=15	8							

Figure 4 : L'algorithme GivensRL2 sur une matrice de taille (8,8)

La figure 5 montre l'ordre d'annulation des éléments de A dans les deux algorithmes.

*							
8	*						
7	9	*					
6	8	10	*				
5	7	9	11	*			
4	6	8	10	12	*		
3	5	7	9	11	13	*	
2	4	6	8	10	12	14	*

GivensRL1

*							
2	*						
3	4	*					
4	5	6	*				
5	6	7	8	*			
6	7	8	9	10	*		
7	8	9	10	11	12	*	
8	9	10	11	12	13	14	*

GivensRL2

Figure 5 : Ordre d'exécution des rotations dans les algorithmes GivensRL1 et GivensRL2

Théorème 2 : Sur un réseau linéaire, les algorithmes GivensRL1 et GivensRL2 sont optimaux pour calculer la décomposition QR d'une matrice carrée de taille n (n pair). Leur temps d'exécution est égal à (2n-1)r. Le nombre optimal de processeurs pour exécuter un algorithme en temps optimal est égal à $\lfloor \frac{n}{2} \rfloor$.

Démonstration

Il faut n unités de temps pour transférer les n lignes de la matrice A de l'hôte vers P_1 et il faut n unités de temps pour transférer les n lignes de R de P_1 vers l'hôte. Pour montrer l'optimalité des deux algorithmes, nous prouvons qu'il est impossible qu'une ligne soit sortie définitivement du réseau avant que toutes les lignes de A ne soient entrées. Remarquons que si un élément sort non définitivement, il devra réentrer, ce qui, avec l'hypothèse de non recouvrement des communications, augmente le temps total.
Pour l'étape 1 de l'élimination, la ligne 1 doit effectuer la dernière élimination car elle est la seule qui restera de longueur n. Lorsque cette (dernière) élimination est effectuée, toutes les lignes sont entrées et ont été éliminées. Comme une seule ligne peut rentrer à chaque top on est donc au moins au top n.
A chaque étape k, la ligne k doit effectuer la dernière élimination car elle est la seule qui restera de longueur n-k+1. A chaque étape k, les lignes qui peuvent être définitivement hors du réseau sont donc les lignes < k.
Comme une seule ligne peut sortir à chaque top et que les sorties débutent au top n (aprés la dernière élimination de l'étape 1), la derenière sortie aura lieu au plus tôt lors du top 2n-1.

Appelons ac(t) le nombre de rotations qu'exécute, au temps t, l'algorithme optimal considéré. Comme les lignes de A ne peuvent être introduites et délivrées que séquentiellement, nous déduisons que $ac(t) \leq \lfloor \frac{t}{2} \rfloor$ pour $1 \leq t \leq n$ et $ac(t) \leq \lfloor \frac{2n-t}{2} \rfloor$ pour $n+1 \leq t \leq 2n-1$. Comme le nombre total de rotations exécutées par l'algorithme est égal à n(n-1)/2, les inégalités précédentes sont en fait des égalités. Pour t=n, $\lfloor \frac{n}{2} \rfloor$ rotations doivent donc être exécutées simultanément, ce qui prouve que $\lfloor \frac{n}{2} \rfloor$ processeurs sont nécessaires. Dans le cas où n est pair cela clôt la démonstration.

Supposons maintenant que n est impair. Nous allons montrer qu'il faut un processeur supplémentaire qui servira au stockage d'une ligne. Nous supposons le contraire c'est à dire qu'il n'y a que $\lfloor \frac{n}{2} \rfloor$ processeurs. Par récurrence, on peut montrer qu'à l'issue des transformations du temps t=n-1, P_i contient une ligne avec i-1 zéros et une autre avec i zéros. La dernière ligne rentrant dans le réseau, une ligne du processeur P_1 doit sortir du réseau. Pour permettre une rotation la ligne

qui sort est celle qui possède un zéro. Il reste donc dans les processeurs $P_2,...,P_{\lfloor n/2 \rfloor}$ une seule ligne possédant un seul zéro, ce qui implique qu'il existe au moins un processeur inactif. Ceci contredit le fait que $ac(n)=\lfloor \frac{n}{2} \rfloor$.

Lorsque n est impair, le processeur $P_{\lceil n/2 \rceil}$ n'effectue aucune rotation. En fait, il ne sert qu'à stocker une ligne comme le montre la figure 6.

	P_1		P_2		P_3		P_4
t=1	1						
t=2	21	R(2,1,1)					
t=3	31	R(3,1,1)	2				
t=4	41	R(4,1,1)	32	R(3,2,2)			
t=5	51	R(5,1,1)	42	R(4,2,2)	3		
t=6	61	R(6,1,1)	52	R(5,2,2)	43	R(4,3,3)	
t=7	71	R(7,1,1)	62	R(6,2,2)	53	R(5,3,3)	4
t=9	72	R(7,2,2)	63	R(6,3,3)	54	R(5,4,4)	
t=10	73	R(7,3,3)	64	R(6,4,4)	5		
t=11	74	R(7,4,4)	65	R(6,5,5)			
t=12	75	R(7,5,5)	6				
t=13	76	R(7,6,6)					
t=14	7						

Figure 6 : L'algorithme GivensRL2 sur une matrice de taille (7,7)

3.1.b. Cas $p < \frac{n}{2}$

Théorème 3: Sur un réseau linéaire composé de p processeurs, le temps optimal $T_{opt}(p)$ pour calculer la décomposition de Givens vérifie la relation:

$$T_{opt}(p) \geq (\frac{n(n-1)}{2p} + 2p)r = R_{inf}(p)$$

Démonstration

Appellons ac(t) le nombre de rotations qu'exécute l'algorithme optimal au temps t.

(i) Comme les lignes de A ne peuvent être introduites que séquentiellement par P_1, nous déduisons que $ac(t) \leq \lfloor \frac{t}{2} \rfloor$ pour $1 \leq t \leq 2p$ par conséquent le nombre d'éléments annulés pendant les 2p premiers instants est inférieur ou égal à p^2.

(ii) Soit t_i le temps auquel l'élément en position (i+1,i) est annulé alors:

$$ac(t_i) \leq \min(p, \lfloor \frac{n-i+1}{2} \rfloor) \text{ donc pour } i > n-2p+1 \text{ on a } ac(t_i) \leq \lfloor \frac{n-i+1}{2} \rfloor$$

Par conséquent $\sum_{i=n-2p+2}^{n-1} ac(t_i) \leq p(p-1)$.

D'aprés la propriété 1 [Introduction et notations, première partie] nous déduisons que:

$$T_{opt}(p) \geq ((\frac{n(n-1)}{2} + p(p-1) + p^2)/p + 1)r$$

Dans la formule précédente nous avons rajouté 1 qui correspond au temps de sortie de la dernière ligne.

Remarque 2: La borne inférieure est atteinte pour $p=\frac{n}{2}$ (ce qui correspond au temps d'exécution de Givens RL1).

3.2. Méthode de Householder

Si nous remplaçons la rotation R(i,j,k) dans l'algorithme GivensRL2 par la transformation d'élimination H_kA [Sam85], nous obtenons donc un algorithme HouseholderRL2 qui consiste à pipeliner les colonnes de A. L'algorithme est décrit dans la figure 7 où A_k désigne la colonne k.

Première phase: De t=1 à t=n, P_i lit une colonne de P_{i-1} et effectue la transformation correspondante. Lorsque la colonne i arrive dans P_i, au temps (2i-1)e, celui-ci calcule la transformation de Householder H_i est mis à jours la colonne i.

Deuxième phase: De t=n à t=2n-1, le sens de communication est renversé: P_i lit la transformation de Householder à partir de P_{i+1} et effectue les modifications correspondantes. Lorsque le processeur P_i est en possession d'une seule colonne, celui-ci calcule la transformation de Householder et met à jour la colonne correspondante. Pendant cette phase, le processeur P_1 délivre à chaque unité de temps une nouvelle colonne de la matrice R.

	P_1		P_2		P_3		P_4	
t=1	A_1	H_1						
t=2	A_2	H_1A_2						
t=3	A_3	H_1A_3	A_2	H_2				
t=4	A_4	H_1A_4	A_3	H_2A_3				
t=5	A_5	H_1A_5	A_4	H_2A_4	A_3	H_3		
t=6	A_6	H_1A_6	A_5	H_2A_5	A_4	H_3A_4		
t=7	A_7	H_1A_7	A_6	H_2A_6	A_5	H_3A_5	A_4	H_4
t=9	A_8	H_1A_8	A_7	H_2A_7	A_6	H_3A_6	A_5	H_4A_5
t=10	H_2	H_2A_8	H_3	H_3A_7	H_4	H_4A_6	A_5	H_5
t=11	H_3	H_3A_8	H_4	H_4A_7	H_5	H_5A_6		
t=12	H_4	H_4A_8	H_5	H_5A_7	A_6	H_6		
t=13	H_5	H_5A_8	H_6	H_6A_7				
t=14	H_6	H_6A_8	A_6	H_7				
t=15	H_7	H_7A_8						
t=16	A_8							

Figure 7 : L'algorithme HousholderRL2 sur une matrice de taille (8,8)

Théorème 4: Sur un réseau linéaire, l'algorithme de HouseholderRL2 est optimal. Son temps d'exécution est égal à (2n-1)e. Le nombre optimal de processeurs pour exécuter un algorithme en temps optimal est égal à $\lceil n/2 \rceil$.
Dans le cas $p < \lfloor \frac{n}{2} \rfloor$ le temps optimal vérifie la relation:

$$T_{opt}(p) \geq (\frac{n(n-1)}{2p} + 2p)e = R_{inf}(p)$$

Remarque 1: Dans les algorithmes que nous avons présentés dans le cas $p=\lceil \frac{n}{2} \rceil$, les deux canaux de communication reliant P_i et P_{i+1} ne sont jamais utilisés simultanément. Les théorèmes sont donc valables sur un réseau linéaire où deux processeurs communiquent entre eux par un unique canal bidirectionnel. Un tel réseau est présenté dans la figure 8.

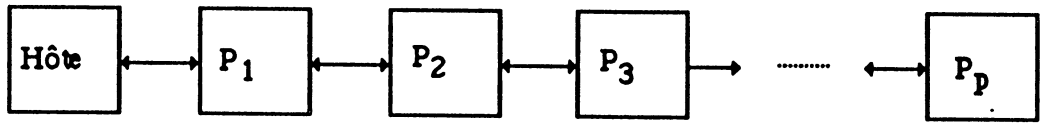


Figure 8: Réseau linéaire de processeurs communiquant par un canal bidirectionnel

V.4. LES ALGORITHMES SUR UN ANNEAU

4.1. L'hôte communique avec un seul processeur

Un anneau de processeurs est un réseau possédant des propriétés de communication supérieures à celles du réseau linéaire comme le montre la figure 9. Chaque processeur P_i ($2 \leq i \leq p$) communique avec P_{i-1} et P_{i+1} (modulo p) par l'intermédiaire d'un canal bidirectionnel. Le processeur P_1 communique avec l'ordinateur hôte aussi par un canal bidirectionnel. Nous supposerons qu'en une unité de temps P_i est capable de lire une ligne (ou une colonne), d'effectuer une rotation (ou une transformation) et une transmission quelle que soit la position qu'occupe l'élément à annuler. Au début de l'algorithme, l'hôte contient la matrice A. A la fin, il doit contenir le résultat cherché: la matrice R.

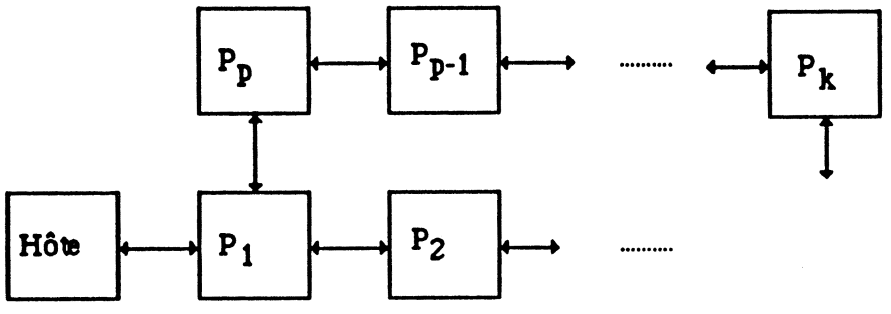


Figure 9: Anneau de processeurs.

Le réseau linéaire de la figure 8 est un sous réseau de l'anneau. Par conséquent, les algorithmes du paragraphe précédent s'appliquent à ce dernier type d'architecture. En reprenant la démonstration du théorème 3, il n'est pas difficile de montrer que les théorèmes 3 et 5 s'etendent à un anneau.

Théorème 5: Sur un anneau les algorithmes:
* GivensRL1 et GivensRL2 sont optimaux. Leur temps d'exécution est égal à (2n-1)r.
* HouseholderRL2 est optimal. Son temps d'exécution est égal à (2n-1)e.

Le nombre optimal de processeurs pour exécuter un algorithme en temps optimal est égal à $\lceil \frac{n}{2} \rceil$.

* Pour $p < \lfloor \frac{n}{2} \rfloor$, $T_{opt}(p) \geq (\frac{n(n-1)}{2p} + 2p)r$

Dans le paragraphe suivant nous montrons comment varie le temps d'exécution si nous augmentons les moyens de communication entre l'hôte et le réseau de processeurs.

4.2. L'hôte communique avec deux processeurs

Dans ce paragraphe nous montrons comment varie la complexité des algorithmes présentés pour l'anneau lorsqu'on augmente les moyens de communication dans ce dernier. L'architecture considérée possède donc des propriétés de communication supérieures à celles du cas précédent. L'ordinateur hôte communique avec chacun des processeurs P_1 et P_p par un canal bidirectionnel comme le montre la figure 10.

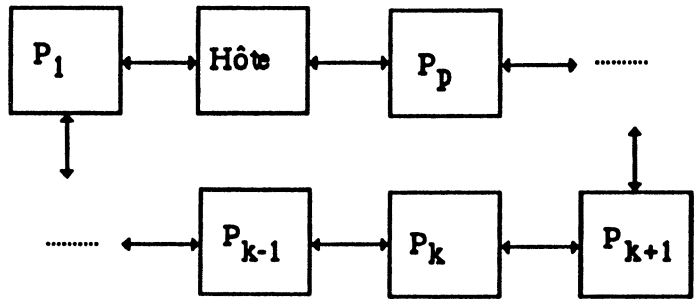


Figure 10: L'hôte communique avec P_1 et P_p

4.2.1. Une borne inférieure.

Dans ce paragraphe, nous montrons comment obtenir une borne inférieure de $T_{opt}(p)$.

Théorème 6: Sur un anneau composé de p processeurs qui communiquent avec l'ordinateur hôte par les processeurs P_1 et P_p, le temps optimal $T_{opt}(p)$ pour calculer la décomposition de Givens vérifie la relation:

$$T_{opt}(p) \geq \left(\frac{n(n-1)}{2p} + \frac{3p}{2} \right) r = A_{inf}(p)$$

Démonstration

Appelons ac(t) le nombre de rotations qu'exécute l'algorithme optimal au temps t.

(i) Comme les lignes de A ne peuvent être introduites que séquentiellement et des deux côtés (par P_1 et par P_p), nous déduisons que $ac(t) \leq 2\lfloor \frac{t}{2} \rfloor$ pour $1 \leq t \leq p$. Par conséquent le nombre d'éléments annulés pendant les p premiers instants est inférieur ou égal à $\frac{p^2}{2}$.

(ii) Soit t_i le temps auquel l'élément en position (i+1,i) est annulé. On a donc

$$ac(t_i) \leq \lfloor \frac{n-i+1}{2} \rfloor \text{ pour } i > n-2p+1$$

Par conséquent $\sum_{i=n-2p-2}^{n-1} ac(t_i) \leq p(p-1)$.

D'aprés la propriété 1 [Introduction et notations, première partie] nous déduisons que:

$$T_{opt}(p) \geq ((\frac{n(n-1)}{2} + \frac{p^2}{2} + p(p-1))/p+1)r = (\frac{n(n-1)}{2p} + \frac{p}{2} + p)r = A_{inf}(p)$$

Nous avons rajouté 1 dans la formule précédente, qui correspond au temps d'écriture de la dernière ligne dans l'hôte.

4.2.2. Un algorithme asymptotiquement optimal pour $p=\frac{n}{3}$.

Théorème 7: Sur un anneau composé de p processeurs, communiquant avec l'ordinateur hôte par les processeurs P_1 et P_p, le temps optimal pour calculer la décomposition de Givens est asymptotiquement égal à (2n-1)r. Le nombre optimal de processeurs est asymptotiquement égal à $\frac{n}{3}$.

Démonstration:
Supposons que 3 divise n. Nous montrons par construction que la borne inférieure du théorème 7 est atteinte pour $p=\frac{n}{3}$. Dans ce cas on a $T_{opt}(\frac{n}{3})=(2n-1)r$. Les rotations sont effectuées dans le même ordre sur la figure 11 qui est en (2n-3)r [ch. III]. La différence entre les temps d'exécutions des deux algorithmes (2r) provient de la lecture et de l'écriture de la ligne n dans l'hôte aux instants t=1 et t=2n-1.

```
*
4   *
3    5   *
2    4   6   *
-----------------------
9   11  13  15   *
8   10  12  14  16   *
7    9  11  13  15  17   *
6    8  10  12  14  16  18   *
5    7   9  11  13  15  17  19   *
4    6   8  10  12  14  16  18  20   *
3    5   7   9  11  13  15  17  19  21   *
2    4   6   8  10  12  14  16  18  20  22   *
```

Figure 11: Ordre d'élimination avec $\frac{n}{3}$ processeurs

L'algorithme que nous construisons est décrit dans la figure 12 dans le cas où n=12 et p=4. Il est construit en **quatre phases.** Les lignes sont introduites dans le réseau les unes après les autres.

Première phase: De t=1 à t=p, les lignes sont "pipelinées" dans le réseau de la façon suivante:

(i) De la ligne n à la ligne n-p+1 par le processeur P_1. Chaque ligne restant deux unités de temps dans P_{i-1} avant d'être transférée dans P_i, pour $2 \leq i \leq \frac{p}{2}$. A chaque instant, P_i lit une ligne à partir de P_{i-1} et effectue la rotation correspondante.

(ii) De la ligne p à la ligne 1 par le processeur P_p. Chaque ligne restant deux unités de temps dans P_{i+1} avant d'être transférée dans P_i, pour $\frac{p}{2} \leq i \leq p-1$. A chaque instant, P_i lit une ligne à partir de P_{i+1} et effectue la rotation correspondante.

Remarquons que tous se passe comme si nous avons 2 anneaux, chacun composé de $\frac{p}{2}$ processeurs. Le premier anneau comuunique avec l'hôte par P_1 et le deuxième anneau communique avec l'hôte par P_p. Sur chacun nous exécutons la première phase de l'algorithme Givens RL1.

Deuxième phase: De t=p+1 à t=2p les lignes sont "pipelinées" dans le réseau par le processeur P_1 de la ligne n-p à la ligne n-2p+2. Par contre le sens de communication dans les $\frac{p}{2}$ derniers processeurs est renversé: P_i lit une ligne à partir de P_{i-1} et effectue la rotation correspondante. Pendant cette phase, le processeur P_p écrit à chaque unité de temps une ligne de la matrice A dans l'hôte.

Troisième phase: De t=2p+1 à t=2n-2p-1, Le sens de communication change à chaque unité de temps. Au temps t=2p+2j+1, pour $0 \leq j \leq n-2p-1$, le processeur P_1 lit à partir du hôte la ligne j+1 et le processeur P_p écrit la ligne n dans l'hôte. Pour $2 \leq i \leq p-1$, P_i lit une ligne à partir de P_{i-1} et effectue la rotation correspondante. Au temps t=2p+2j+2 le processeur P_1 écrit une nouvelle ligne de R (la ligne j+1) dans l'hôte et le processeur P_p lit la ligne n à partir du hôte. Pour $2 \leq i \leq p-1$, P_{i-1} lit une ligne à partir de P_i et effectue la rotation correspondante.

Quatrième phase: De t=2n-2p à t=2n-1, P_i lit une ligne à partir de P_{i+1} et effectue la rotation correspondante. Pendant cette phase, le processeur P_1 délivre à chaque unité de temps une nouvelle ligne de la matrice R.

	P_1		P_2		P_3		P_4	
t=1	12							4
t=2	11.12	R(12,11,1)					R(4,3,1)	4.3
t=3	10.11	R(11,10,1)	12			4	R(3,2,1)	3.2
t=4	9.10	R(10,9,1)	11.12	R(12,11,2)	R(4,3,2)	4.3	R(2,1,1)	2.1
t=5	8.9	R(9,8,1)	10.11	R(11,10,2)	12	4	R(3,2,2)	3.2
t=6	7.8	R(8,7,1)	9.10	R(10,9,2)	11.12	R(12,11,3)	R(4,3,3)	4.3
t=7	6.7	R(7,6,1)	8.9	R(9,8,2)	10.11	R(11,10,3)	12	4
t=8	5.6	R(6,5,1)	7.8	R(8,7,2)	9.10	R(10,9,3)	11.12	R(12,11,4)
t=9	1.5	R(5,1,1)	6.7	R(7,6,2)	8.9	R(9,8,3)	10.11	R(11,10,4)
t=10	5.6	R(6,5,2)	7.8	R(8,7,3)	9.10	R(10,9,4)	11.12	R(12,11,5)
t=11	2.5	R(5,2,2)	6.7	R(7,6,3)	8.9	R(9,8,4)	10.11	R(11,10,5)
t=12	5.6	R(6,5,3)	7.8	R(8,7,4)	9.10	R(10,9,5)	11.12	R(12,11,6)
t=13	3.5	R(5,3,3)	6.7	R(7,6,4)	8.9	R(9,8,5)	10.11	R(11,10,6)
t=14	5.6	R(6,5,4)	7.8	R(8,7,5)	9.10	R(10,9,6)	11.12	R(12,11,7)
t=15	4.5	R(5,4,4)	6.7	R(7,6,5)	8.9	R(9,8,6)	10.11	R(11,10,7)
t=16	5.6	R(6,5,5)	7.8	R(8,7,6)	9.10	R(10,9,7)	11.12	R(12,11,8)
t=17	6.7	R(7,6,6)	8.9	R(9,8,7)	10.11	R(11,10,8)	12	
t=18	7.8	R(8,7,7)	9.10	R(10,9,8)	11.12	R(12,11,9)		
t=19	8.9	R(9,8,8)	10.11	R(11,10,9)	12			
t=20	9.10	R(10,9,9)	11.12	R(12,11,10)				
t=21	10.11	R(11,10,10)	12					
t=22	11.12	R(12,11,11)						
t=23	12							

Figure 12: Algorithme de Givens sur l'anneau communiquant avec l'hôte par 2 processeurs: cas $p = \frac{n}{3}$

De la relation $A_{inf}(p) \geq T_{opt}$ on trouve qu'asymptotiquement $p \leq \frac{n}{3} + o(n)$. Par conséquent le nombre optimal de processeurs pour calculer la décomposition de Givens en T_{opt} est asymptotiquement égal à $\frac{n}{3}$.

Si 3 ne divise pas n alors on utilise $\lceil \frac{n}{3} \rceil$ processeurs pour exécuter l'algorithme en (2n-1)r de la façon suivante. On décompose la matrice en 2 blocs, le premier est composé des $q=(n-2\lceil \frac{n}{3} \rceil)$ premières lignes alors que le deuxième est composé des $2\lceil \frac{n}{3} \rceil$ dernières lignes. On pipeline les lignes dans le réseau par P_1 à partir de la ligne n alors que par P_p à partir de ligne q. Le principe reste le même que

l'algorithme décrit précédemment.

4.2.3. Un algorithme pour p quelconque

Remarquons tout d'abord que si le nombre de processeurs p est supérieur à $\frac{n}{3}$ il suffit d'utiliser $\frac{n}{3}$ processeurs pour calculer la décomposition de Givens asymptotiquement en temps optimal. Nous supposons donc que n=(2k+1)p+cp avec k$\in$ IN* et 0≤c<2, nous proposons un algorithme asymptotiquement optimal lorsque c=0.

Théorème 8: Si $T_{opt}(p)$ est le temps optimal pour calculer la décomposition de Givens d'une matrice carrée dense de taille n, avec n=(2k+1)p+cp, k$\in$ IN* et 0≤c<2, sur un anneau composé de p processeurs qui communiquent avec l'ordinateur hôte par les processeurs P_1 et P_p alors asymptotiquement on a:

(i) $T_{opt}(p) = (\frac{n^2}{2p} + \frac{3p}{2} - 2)r = A_{inf}(p) + o(p)$ si c=0

(ii) $(\frac{n^2}{2p} + \frac{3p}{2})r \leq T_{opt}(p) \leq (\frac{n^2}{2p} + \frac{3p}{2} + \frac{(2-c)(2n - p(c+2))}{2})r$ si c≠0

Démonstration

(i) Cas c=0

n=(2k+1)p avec k$\in$ IN*. Nous décomposons la matrice en k blocs. Le premier bloc est composé des 3p premières lignes. Les autres blocs sont composés chacun de 2p lignes. Les rotations de l'algorithme que nous proposons sont effectuées dans le même ordre et aux mêmes instants que sur la figure 13 dans le cas où n=20 et p=4. La différence de temps (kr) avec l'algorithme présenté dans le chapitre IV provient de la lecture des lignes (2j+1)p pour 0≤j≤k.

Le principe de cet algorithme est basé sur l'idée suivante. Pendant que l'une des deux extrémités de l'anneau délivre à chaque unité de temps une ligne du bloc (j-1) (j>1), par l'autre extrémité on introduit les lignes du bloc j, l'une aprés l'autre en commençant par la ligne (2j+1)p. Comme le premier bloc est composé de 3p lignes on lui applique l'algorithme 4.2.2 pour l'exécuter en temps optimal. L'algorithme est construit en k phases et est décrit sur la figure 14 dans le cas où n=20 et p=4.

```
*
4  *
3   5  *
2   4   6  *
----------------------
9  11  13  15  *
8  10  12  14  16  *
7   9  11  13  15  17  *
6   8  10  12  14  16  18  *
5   7   9  11  13  15  17  19  *
4   6   8  10  12  14  16  18  20  *
3   5   7   9  11  13  15  17  19  21  *
2   4   6   8  10  12  14  16  18  20  22  *
-------------------------------------------------------------
25 27 29 31 33 35 37 39 41 43 45 47 *
24 26 28 30 32 34 36 38 40 42 44 46 48 *
23 25 27 29 31 33 35 37 39 41 43 45 47 49 *
22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 *
21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 *
20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 *
19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 *
18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 *
```

Figure 13: Ordre d'élimination dans le cas où n=20 et p=4

Description des phases de l'algorithme

Première phase: Elle débute au temps $t_1=1$ et se termine au temps $f_1=t_1+4p-1$ et consiste à appliquer au premier bloc les trois premières phases de l'algorithme décrit précédemment.

Deuxième phase: Elle débute au temps $t_2=f_1+1$ et se termine au temps $f_2=t_2+8p-1$. Elle est décomposée en 2 étapes: **La première étape** consiste à annuler simultanément dans le premier et le deuxième bloc de la façon suivante. Pendant que le processeur P_1 délivre, à chaque unité de temps, une ligne du premier bloc, le processeur P_p lit une ligne à partir de l'hôte en commençant par lire la ligne 5p. Donc, pour le bloc 2, on exécute la première phase de l'algorithme GivensRL1 alors que pour le premier bloc on exécute la deuxième phase de l'algorithme GivensRL1. **La deuxième étape** débute au temps $t_2+2(p-1)$ et consiste à annuler dans le deuxième bloc de la même façon que la troisième phase de l'algorithme 4.2.2.

Troisième phase: Elle débute au temps $t_3=f_2+1$ et se termine au temps $f_3=t_3+12p-1$. Elle annule avec le même principe que la deuxième phase, la seule différence réside dans le fait que le rôle joué par les processeurs P_1 et P_p est inversé.

En général:

j-ème phase: Elle débute au temps $t_j=f_{j-1}+1$ et se termine au temps $f_j=t_j+4jp-2$. Elle annule avec le même principe que la deuxième phase si j est pair, si non avec le même principe que la troisième phase.

K-ème phase: Elle débute au temps $t_k=f_{k-1}+1$ et se termine au temps $f_k=t_k+2n-3$. Elle est décomposée en 3 étapes: Dans **la première et la deuxième étapes** on annule avec le même principe que la j ième phase mais en délivrant pendant la deuxième étapes à chaque 2 instants une nouvelle ligne de R. Pendant **la troisième étape** le processeur P_1 (si k est pair si non le processeur P_p) délivre à chaque instant une nouvelle ligne de R de la même façon que la troisième phase de l'algorithme GivensRL1.

Temps d'exécution: Le temps d'exécution de cet algorithme est celui de la dernière phase donc égal à $T(p)=f_k=t_k+2n-3$. A partir des relations $f_j=t_j+4jp-2$ et de $t_j=f_{j-1}+1$ pour $1\leq j\leq k$ nous en déduisons que $t_k=1+2k(k-1)p$ et par conséquent $T(p)=2k(k-1)p+2n-2$. Nous remplaçons dans la formule k par $(n-p)/2p$ alors asymptotiquement nous trouvons: $T_{opt}(p)=\frac{n^2}{2p}+\frac{3p}{2}-2$

	P_1		P_2		P_3		P_4	
...Cf figure 12								
t=16	R(6,5,5)	6.5	R(8,7,6)	8.7	R(10,9,7)	9.10	R(12,11,8)	11.12
t=17	R(7,6,6)	6.7	R(9,8,7)	9.8	R(11,10,8)	10.11	12	20
t=18	R(8,7,7)	7.8	R(10,9,8)	9.10	R(12,11,9)	11.12	R(20,19,1)	20.19
t=19	R(9,8,8)	8.9	R(11,10,9)	10.11	12	20	R(19,18,1)	19.18
t=20	R(10,9,9)	9.10	R(12,11,10)	11.12	R(20,19,2)	20.19	R(18,17,1)	18.17
t=21	R(11,10,10)	10.11	12	20	R(19,18,2)	19.18	R(17,16,1)	17.16
t=22	R(12,11,11)	11.12	R(20,19,3)	20.19	R(18,17,2)	18.17	R(16,15,1)	16.15
t=23	12	20	R(19,18,3)	19.18	R(17,16,2)	17.16	R(15,14,1)	15.14
t=24	R(20,19,4)	20.19	R(18,17,3)	18.17	R(16,15,2)	16.15	R(14,13,1)	14.13
t=25	R(19,18,4)	19.18	R(17,16,3)	17.16	R(15,14,2)	15.14	R(13,1,1)	13.1
t=26	R(20,19,5)	20.19	R(18,17,4)	18.17	R(16,15,3)	16.15	R(14,13,2)	14.13
t=27	R(19,18,5)	19.18	R(17,16,4)	17.16	R(15,14,3)	15.14	R(13,2,2)	13.2
t=28	R(20,19,6)	20.19	R(18,17,5)	18.17	R(16,15,4)	16.15	R(14,13,3)	14.13
t=29	R(19,18,6)	19.18	R(17,16,5)	17.16	R(15,14,4)	15.14	R(13,3,3)	13.3
t=30	R(20,19,7)	20.19	R(18,17,6)	18.17	R(16,15,5)	16.15	R(14,13,4)	14.13
t=31	R(19,18,7)	19.18	R(17,16,6)	17.16	R(15,14,5)	15.14	R(13,4,4)	13.4
t=32	R(20,19,8)	20.19	R(18,17,7)	18.17	R(16,15,6)	16.15	R(14,13,5)	14.13
t=33	R(19,18,8)	19.18	R(17,16,7)	17.16	R(15,14,6)	15.14	R(13,5,5)	13.5
t=34								
...								
t=46	R(20,19,15)	20.19	R(18,17,14)	18.17	R(16,15,13)	16.15	R(14,13,12)	14.13
t=47	R(19,18,15)	19.18	R(17,16,14)	17.16	R(15,14,13)	15.14	R(13,9,12)	13.12
t=48	R(20,19,16)	20.19	R(18,17,15)	18.17	R(16,15,14)	16.15	R(14,13,13)	14.13
t=49		20	R(19,18,16)	19.18	R(17,16,15)	17.16	R(15,14,14)	15.14
t=50			R(20,19,17)	20.19	R(18,17,16)	18.17	R(16,15,15)	16.15
t=51				20	R(19,18,17)	19.18	R(17,16,16)	17.16
t=52					R(20,19,18)	20.19	R(18,17,17)	18.17
t=53						20	R(19,18,18)	19.18
t=54							R(20,19,19)	20.19
t=55								20

Figure 14: Givens sur l'anneau cas n=20 et p=4

(ii) Cas c≠0

Posons n=(2k+1)p+cp avecke IN* et 0<c<2. Nous décomposons la matrice en (k+1) blocs. Les k premiers blocs sont composés de la même façon que le cas précédent. Le dernier bloc est composé des cp dernières lignes. Les rotations de l'algorithme que nous proposons sont effectuées dans le même ordre que montre la figure 15 pour n=18 et p=4 (cp=6).

```
*
4  *
3   5  *
2   4   6  *
----------------------
9  11  13  15  *
8  10  12  14  16  *
7   9  11  13  15  17  *
6   8  10  12  14  16  18  *
5   7   9  11  13  15  17  19  *
4   6   8  10  12  14  16  18  20  *
3   5   7   9  11  13  15  17  19  21  *
2   4   6   8  10  12  14  16  18  20  22  *
------------------------------------------------------------------
23  25  27  29  31  33  35  37  39  41  43  45  *
22  24  26  28  30  32  34  36  38  40  42  44  46  *
21  23  25  27  29  31  33  35  37  39  41  43  45  47  *
20  22  24  26  28  30  32  34  36  38  40  42  44  46  48  *
19  21  23  25  27  29  31  33  35  37  39  41  43  45  47  49  *
18  20  22  24  26  28  30  32  34  36  38  40  42  44  46  48  50  *
```

Figure 15: Ordre d'élimination dans le cas où n=18 et p=4

L'algorithme est construit en (k+1) phases. Il est décrit dans la figure 16 dans le cas où n=18, p=4 et $c=\frac{3}{2}$. Les k premières phases sont les mêmes que celles de l'algorithme décrit précédemment. La seule différence provient de l'exécution de la (k+1) ième phase. En effet comme le dernier bloc est composé de cp lignes alors nous n'utilisons que $\frac{cp}{2}$ processeurs quant au principe reste le même.

(k+1)-ème phase: Elle consiste à éliminer de la même façon que la k-ème phase du cas précédent. Supposons k pair. Pendant que le processeur P_1 délivre les lignes du bloc k, le processeur P_p lit les lignes du bloc (k+1) à partir de l'hôte en commençant par la ligne n. Comme le bloc (k+1) est composé de cp lignes alors

nousn'utilisons que cp/2 processeurs. Par conséquent dans la deuxième étape c'est le processeur $P_{p-cp/2}$ qui jouera le rôle de l'hôte. La troisième étape consiste à délivrer à chaque unité de temps une ligne de la matrice R.

Temps d'exécution: Pour le temps d'exécution il suffit de remplacer k par (k+1) dans la formule précédente. On obtient donc T(p)=2k(k+1)p+2n-2. Par la suite nous remplaçons dans T(p) la valeur de k par (n-p-cp)/2p.

	P_1		P_2		P_3		P_4	
... Cf figure 14								
t=17	R(7,6,6)	6.7	R(9,8,7)	9.8	R(11,10,8)	10.11	12	18
t=18	R(8,7,7)	7.8	R(10,9,8)	9.10	R(12,11,9)	11.12	R(18,16,1)	18.17
t=19	R(9,8,8)	8.9	R(11,10,9)	10.11	12	18	R(17,15,1)	17.16
t=20	R(10,9,9)	9.10	R(12,11,10)	11.12	R(18,17,2)	18.17	R(16,15,1)	16.15
t=21	R(11,10,10)	10.11	12	18	R(17,16,2)	17.16	R(15,14,1)	15.14
t=22	R(12,11,11)	11.12	R(18,17,3)	18.17	R(16,15,2)	16.15	R(14,13,1)	14.13
t=23	12	18	R(17,16,3)	17.16	R(15,14,2)	15.14	R(13,1,1)	13.1
t=24			R(18,17,4)	18.17	R(16,15,3)	16.15	R(14,13,2)	14.13
t=25		18	R(17,16,4)	17.16	R(15,14,3)	15.14	R(13,2,2)	13.2
t=26			R(18,17,5)	18.17	R(16,15,4)	16.15	R(14,13,3)	14.13
t=27		18	R(17,16,5)	17.16	R(15,14,4)	15.14	R(13,3,3)	13.3
t=28			R(18,17,6)	18.17	R(16,15,5)	16.15	R(14,13,4)	14.13
t=29		18	R(17,16,6)	17.16	R(15,14,5)	15.14	R(13,4,4)	13.4
t=30			R(18,17,7)	18.17	R(16,15,6)	16.15	R(14,13,5)	14.13
t=31		18	R(17,16,7)	17.16	R(15,14,6)	15.14	R(13,5,5)	13.5
...								
t=44			R(18,17,14)	18.17	R(16,15,13)	16.15	R(14,13,12)	14.13
t=45		18	R(17,16,14)	17.16	R(15,14,13)	15.14	R(13,12,12)	13.12
t=46			R(18,17,15)	18.17	R(16,15,14)	16.15	R(14,13,13)	14.13
t=47				18	R(17,16,15)	17.16	R(15,14,14)	15.14
t=48					R(18,17,16)	18.17	R(16,15,15)	16.15
t=49						18	R(17,16,16)	17.16
t=50							R(18,17,17)	18.17
t=51								18

Figure 16: Givens sur l'anneau cas p=4 et n=18

Remarque 3: Les resultats obtenus pour la méthode de Givens ne peuvent pas être étendus à la méthode de Householder. En effet dans cette dernière, la colonne k ne peut servir de colonne pivot, pour calculer la transformation H_k, que lorsque elle est modifiée par toutes les transformations H_i pour i<k. Par conséquent on ne peut pas charger le réseau par les deux extrémités. Pour étendre les résultats obtenus à la méthode de Householder, il faut faire des transformations locales (entre deux lignes pour annuler un seul élément). Par conséquent le problème revient à faire exactement la méthode de Givens, au lieu d'avoir des rotations nous aurons des symétries orthogonales.

V.5. CONCLUSION

Nous avons étudié les méthodes de Givens et de Householder pour calculer la décomposition orthogonale d'une matrice carrée de taille n sur 2 types d'architectures multiprocesseurs chacune composée de p processeurs: le réseau linéaire et l'anneau. Les résultats de complexités obtenus comparés entre eux ou avec ceux obtenus pour le réseau complet (mémoire globale avec fonctionnement par partage de données) montrent l'influence de l'architecture sur les algorithmes considérés. Deux cas ont été étudiés suivant le nombre p de processeurs:

Cas $p=\lceil \frac{n}{2} \rceil$**:** Dans le cas du réseau complet, les processeurs ont un accès en parallèle à l'ordinateur hôte (ici mémoire partagée), alors que dans les deux autres cas, l'accès est séquentiel. Ceci à peu d'influence sur la méthode de Givens puisque les performances dans chaque cas sont du même ordre: (2n-1)r pour le réseau linéaire et l'anneau, et (2n-o(n))r pour le réseau complet. Par contre la différence n'est pas négligeable pour la méthode de Householder (2n-1)e pour le réseau linéaire et l'anneau, et (n-1)e pour le réseau complet. Cela s'explique par le fait que la duplication d'une colonne permet d'accélérer l'algorithme dans cette méthode.

Les résultats précédents montrent que pour l'agorithme de Givens, les performances sur divers types de réseaux sont proches. Par contre pour l'algorithme de Householder ces differences sont non-négligeables, il est clair que l'augmentation du temps d'exécution est liée aux moyens de communications entre l'hôte (mémoire commune) et les processeurs. Nous en déduisons que le rôle joué par les communications entre les processeurs et l'hôte est plus important que les communications entre les processeurs. Par exemple, un réseau hypercube dont un seul des processeurs est connecté à l'hôte n'est utilisé que comme un réseau linéaire pour ces algorithmes.

Cas $p< \lfloor \frac{n}{2} \rfloor$**:** Dans ce cas la différence des performances sur les trois types d'architectures considérés n'est pas négligeable pour les deux algorithmes. En effet

puisque l'accès est séquentiel pour l'anneau1 (anneau communiquant par un seul processeur avec l'hôte) et pour le réseau linéaire on ne peut pas annuler avec une efficacité 1 pendant les 2p premiers instants (temps minimum pour charger le réseau).

L'augmentation des moyens de communications entre l'hôte et les processeurs influent considérablement sur les performances de chaque algorithme. Nous l'avons montré lorsque nous avons rajouté une seule connexion entre l'hôte et l'anneau en supposant que ce dernier communique avec l'hôte par deux processeurs (anneau2). Nous avons résumé dans le tableau 1 les résultats de complexité de la décomposition de Givens sur les différents types d'architectures.

Bien sûr notre modèle d'architecture est très imparfait. Nous supposons que l'unité de temps d'exécution est grossière: une rotation ou une transformation indépendamment de la position de l'élément à annuler. De plus, pour les comparaisons, raisonner à la largeur de parallèlisme égal peut apparaître comme une restriction puisque l'avantage du calcul distribué sur la mémoire partagée est la possibilté de parallélisme massif.

GIVENS	T_{opt}	p_{opt}	$T_{opt}(p)$
Mémoire partagée	(2n-o(n))r	$\frac{n}{2+\sqrt{2}}$	$= \frac{n^2}{2p} + p + o(n)$
Réseau linéaire	2nr	$\frac{n}{2}$	$\geq \frac{n^2}{2p} + 2p + o(n)$
Anneau 1	2nr	$\frac{n}{2}$	$\geq \frac{n^2}{2p} + 2p + o(n)$
Anneau 2	2nr	$\frac{n}{3}$	$= \frac{n^2}{2p} + \frac{3}{2}p + o(n)$

Tableau 1: Résultats de la complexité de la méthode de Givens sur différents types d'architectures multiprocesseurs.

CHAPITRE VI

DECOMPOSITION DE HOUSEHOLDER SUR UNE ARCHITECTURE A MEMOIRE DISTRIBUEE

VI.1. INTRODUCTION

Dans ce chapitre nous étudions la complexité de la décomposition QR d'une matrice rectangulaire de taille (m,n) par la méthode de Householder sur une architecture multiprocesseur à mémoire distribuée, composée de p processeurs. On suppose que chaque processeur possède une mémoire locale et une unité de traitement. Les communications inter-processeurs sont accomplies par passage de message en utilisant les liens locaux qui relient physiquement les processeurs. Les processeurs ont un fonctionnement totalement indépendant (mode asynchrone). La seule synchronisation est due au protocole de communication envoyer/recevoir. Le processeur P_i peut envoyer un message au processeur P_j si et seulement si P_i et P_j sont voisins et que P_j est prêt à lire. On suppose qu'il n'y a pas de transfert de données par l'intermédiaire d'une mémoire partagée ou par un bus.

Concernant les communications, le temps nécessaire pour envoyer/recevoir q données consécutives est défini dans [Saa85] par $(\beta_c+q\tau_c)$ où β_c est le temps d'initialisation et τ_c est le temps de transfert d'une donnée.

Concernant l'arithmétique nous supposons que chaque processeur possède une unité vectorielle avec un additionneur/soustracteur et un multiplieur/diviseur. Les deux unités vectorielles peuvent être enchainées. Par conséquent, l'unité de temps pour réaliser une opération vectorielle entre deux vecteurs chacun de longueur q est égal à $(\beta_a+q\tau_a)$ où β_a est le temps d'initialisation et τ_a est le temps du cycle de l'unité vectorielle.

Nous avons implémenté la méthode de Householder sur trois types d'architectures multiprocesseurs à mémoire distribuée: l'anneau, l'hypercube et le réseau linéaire de processeurs. On appelle colonne pivot la colonne qui permet de calculer la

transformation de Householder H_k à la k ième étape (H_k est complètement déterminée par un vecteur de longueur (m-k+1)). La stratégie utilisée consiste à déterminer la transformation H_k dans le processeur qui contient la colonne pivot puis à l'envoyer aux processeurs voisins suivant le mode de communication choisi. Nous avons testé les algorithmes, que nous présentons, sur l'hypercube T20 de FPS [GHS] installé au laboratoire TIM3 Grenoble.

Les algorithmes sont écrits en pseudo OCCAM [Hoa], [Wil] où **Seq, Par, If** et **True** sont les opérateurs du langage OCCAM pour spécifier le séquentiel, le parallèle ou le conditionnel.

VI.2. METHODE SEQUENTIELLE DE HOUSEHOLDER

Rappelons que la transformation de Householder consiste à calculer une matrice carrée de taille m de la forme $H_k=I-2uu^t$ où u est un vecteur unitaire construit pour que le produit H_kA_k annule les (m-k) dernières composantes de A_k, A_k est la k ième colonne de A [ch. I].

Nous supposons que l'addition et la multiplication sont enchainées. Pour simplifier, nous supposons que le temps d'une racine carrée est égal au temps d'exécution d'une opération élémentaire qu'on suppose égal à $(\beta_a+\tau_a)$.

Temps de calcul

Soient : - T_{calH_k} le temps pour calculer H_k à la k-ème étape:

$$T_{calH_k} = 4(\beta_a+\tau_a)+(\beta_a+(m-k+1)\tau_a)=5(\beta_a+\tau_a)+(m-k)\tau_a$$

- T_{applH_k} le temps total pour appliquer H_k à la k-ème étape:

$$T_{applH_k} = \sum_{j=k+1}^{n} ((\beta_a+\tau_a) +2\beta_a+2(m-k+1)\tau_a)$$
$$=3(n-k)(\beta_a+\tau_a) +2(n-k)(m-k)\tau_a$$

- T_{calseq} est le temps d'exécution de la méthode séquentielle:

$$T_{calseq} = \sum_{k=1}^{n} (T_{calH_k} + T_{applH_k})$$

$$T_{calseq} =n^2(m-\frac{n}{3})\tau_a+\frac{3}{2}n^2\beta_a + n^2\tau_a+\frac{7}{2}n\beta_a +o(n^2)$$

Par conséquent, lorsque la multiplication et l'addition sont enchainées, théoriquement, le temps d'exécution est asymptotiquement divisé par 2 comparé au

nombre d'opérations arithmétiques.

VI.3. IMPLEMENTATIONS PARALLELES

Nous étudions la complexité de la parallélisation de de la méthode de Householder sur trois types d'architectures multiprocesseurs à mémoire distribuée composée de p processeurs: l'anneau, l'hypercube et le réseau linéaire de processeurs. On décompose la matrice en p blocs, chacun formé de $\frac{n}{p}$ colonnes, puis on stocke un bloc dans chaque processeur. On suppose que les processeurs sont numérotés par P_i, pour i=0 à p-1. Plusieurs décompositions de A sont examinées dans [Saa86], [Tou] pour la méthode de Gauss. Pour tester nos algorithmes, nous avons choisi dans les paragraphes 3.1 et 3.2 la décomposition qui permet d'équilibrer le travail des processeurs. Au processeur P_i on affecte le bloc formé des colonnes i+1+kp pour k=0 à $\frac{n}{p}$ -1. Dans le paragraphe 3.3 on affecte au processeur P_i le bloc formé de $\frac{n}{p}$ colonnes consécutives $i\frac{n}{p}$ +k pour k=1 à $\frac{n}{p}$.

Pour résoudre le système linéaire Ax=b, on stocke le vecteur b dans P_{p-1}, puis on effectue sur lui les mêmes transformations que sur les autres colonnes de A. On se ramène donc à un système triangulaire supérieur qu'on sait résoudre efficacement sur les machines à mémoire distribuée. Dans notre travail nous ne nous intéressons qu'à l'étape de triangularisation, pour l'étape de résolution nous renvoyons à Ipsen et al. [ISS] et à Li et Coleman [LC].

Soient $T_{cal}(p)$ le temps de l'arithmétique et $T_{comm}(p)$ le temps des communications. $T_{comm}(p)$ correspond au temps des transferts des données $T_{tr}(p)$, plus le temps de synchronisation $T_{sy}(p)$, $T_{comm}(p)=T_{tr}(p)+T_{sy}(p)$. Soit $T_{exec}(p)$ le temps d'exécution de l'algorithme. Dans [SS2] on montre que le recouvrement de l'arithmétique et des communications permet d'avoir un gain d'un facteur au plus égal à 2 lorsque le temps d'attente est négligé.

3.1. Les algorithmes sur l'anneau

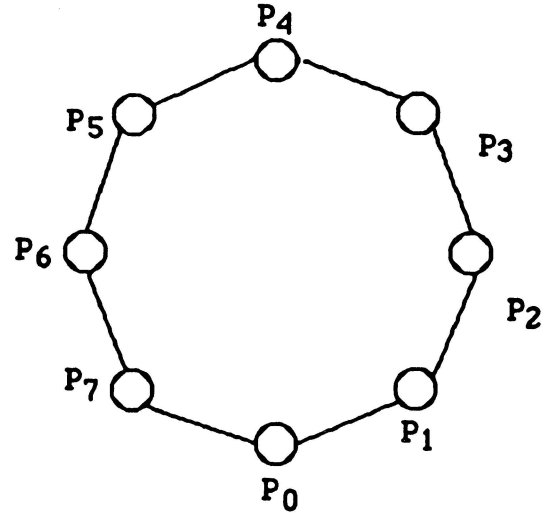


Figure 1: Anneau de 8 processeurs

Nous supposons que nous disposons d'un anneau composé de p processeurs comme le montre la figure 1. Les processeurs $P_{(i-1) \bmod p}$ et $P_{(i+1) \bmod p}$, pour i=0 à (p-1) sont les voisins du processeur P_i. On décompose la matrice en p blocs puis on affecte au processeur P_i, le bloc formé des colonnes i+1+kp pour k=0 à $\frac{n}{p}$ -1.

a. L'algorithme du pipeline sur l'anneau

L'algorithme du pipeline sur l'anneau "Pipline Ring Algorithm" introduit dans [Saa85] pour la méthode de Gauss s'adapte parfaitement à la méthode de Householder. Il s'exécute en n étapes. Dans chaque étape k, $1 \leq k \leq n$, on pipeline la transformation de Householder H_k calculée dans le processeur qui contient la colonne k. L'algorithme PR ci dessous décrit l'algorithme du pipeline sur l'anneau.

Algorithme du pipeline dans l'anneau: PR.

```
Seq k=[1 For n-1]
   If
      (Pi=((k-1) mod p ))                    * Processeur qui contient la colonne k *
         Seq
            calculer la transformation de Householder Hk
            Par
               envoyer Hk au processeur ( (Pi+1) mod p )
               appliquer Hk
      True
         Seq
            recevoir la transformation Hk
```

If
(P_i+1) mod p <> ((k-1) mod p)
Par
envoyer H_k au processeur ((P_i+1) mod p)
appliquer H_k
True
appliquer H_k

*** Temps de calcul**

Le temps de calcul de l'algorithme PR est donné par le processeur P_{p-1}. C'est le processeur qui contient la dernière colonne. A chaque étape k, le processeur P_{p-1} applique la transformation de Householder H_k au $\lceil \frac{n-k}{p} \rceil$ colonnes qu'il contient. De plus il calcule H_k lorsque la colonne pivot lui appartient (lorsque la colonne k est un multiple de p). On obtient donc:

$$T_{cal}(p)= \sum_{k/k=ip}^{n} \left[5(\beta_a+\tau_a)+(m-k)\tau_a\right] + \sum_{k=1}^{n-1} \lceil \frac{n-k}{p} \rceil \left[3(\beta_a+\tau_a)+2(m-k)\tau_a\right]$$

$$=\frac{n^2}{p}(m-\frac{n}{3})\tau_a + \frac{3n^2}{2p}\beta_a + \left[(1-\frac{1}{p})mn +(\frac{3}{2p}-\frac{1}{2})n^2\right]\tau_a + \left[\frac{3}{2}+\frac{2}{p}\right]n\beta_a + o(n^2m)$$

*** Temps de communication**

A l'étape k la transformation de Householder H_k est déterminée par un vecteur v de longueur (m-k+1) et par sa norme n(v) [ch. I]. Par conséquent H_k est déterminée par un vecteur de longueur (m-k+2). Le temps de communication est donné par:

1) Le temps d'initialisation du pipeline, correspond au temps pour envoyer H_1 au processeur P_{p-1}, qui est égal à $(p-1)((m+1)\tau_c+\beta_c)$.
2) Si la colonne pivot, supposée de longueur q, appartient au processeur P_{p-1} alors elle doit être envoyée à tous les processeurs. Ceci demande $(p-1)((q+1)\tau_c+\beta_c)$.
3) Si la colonne pivot, supposée de longueur q, n'appartient pas au processeur P_{p-1} alors celui ci la reçoit à partir du processeur P_{p-2} et puis l'envoie au processeur P_0. Ceci est accomplie en $2((q+1)\tau_c+\beta_c)$.
4) Pendant les p dernières étapes, le processeur P_{p-1} recoit la colonne pivot, supposée de longueur q, sans l'envoyer au processeurs P_0. Ceci est accompli en $((q+1)\tau_c+\beta_c)$

1) et 2) représentent le temps de synchronisation donc égal à:

$$T_{sy}(p)= (p-1)((m+1)\tau_c + \beta_c) + (p-1)\sum_{j=1}^{\frac{n}{p}-1} [\beta_c+(m-jp+2)\tau_c]$$

$$= \frac{(p-1)}{p} [\, n\beta_c + n(m - \frac{n}{2})\tau_c \,] + o(nm)$$

3) et 4) représentent le temps de transfert donc égal à:

$$T_{tr}(p)= 2\sum_{j=1}^{n-p-1} [\beta_c+(m-j+2)\tau_c] \; -2\sum_{j=1}^{\frac{n}{p}-1} [\beta_c+(m-jp+2)\tau_c] + \sum_{j=n-p}^{n-1} [\beta_c+(m-j+2)\tau_c]$$

$$= \frac{(p-1)}{p} [\, 2n\beta_c + n(2m-n)\tau_c \,] + o(mn)$$

Par conséquent le temps total de communication $T_{comm}(p)$ est égal à:

$$T_{comm}(p)= \frac{(p-1)}{p}\left[3n\beta_c +3\, n(m- \frac{n}{2})\tau_c\right] + o(mn)$$

b. L'algorithme de diffusion sur l'anneau (BR)

Le principe de cet algorithme consiste, à chaque étape k, à pipeliner la transformation de Householder H_k dans les deux directions de l'anneau. Supposons que le processeur P_0 contient H_k alors celui ci l'envoie simultanément à P_{p-1} et à P_1. Ces derniers chacun à son tour l'envoie au processeur voisin. Par conséquent le temps pour que H_k arrive à tous les processeurs est divisé par deux. Le temps de communication de l'algorithme est au plus divisé par 2 ([ISS]) lorsque le temps du retard est négligé. Quant au temps de calcul, il est le même que celui de l'algorithme du pipeline.

3.2. Les algorithmes de diffusion sur l'hypercube

On suppose que nous disposons d'un d-cube composé de $p=2^d$ processeurs. Donc chaque processeur à $d=\log_2(p)$ voisins. On décompose la matrice en p blocs, chacun contient $\frac{n}{p}$ colonnes. Au processeur P_i on affecte le bloc formé des colonnes i+1+kp pour k=0 à $\frac{n}{p}$ -1. Le principe de l'algorithme consiste, à chaque étape k, à utiliser tous les liens pour envoyer H_k à partir du processeur racine (processeur qui contient la colonne k) à tous les processeurs dans le minimum de temps, qui est égal à $(\beta_c +(m-k+2)\tau_c)\log_2(p)$. La structure de l'arbre de communication [SS], [SS1] est décrite sur la figure 2 pour un 3-cube en supposant que la racine est le processeur 0.

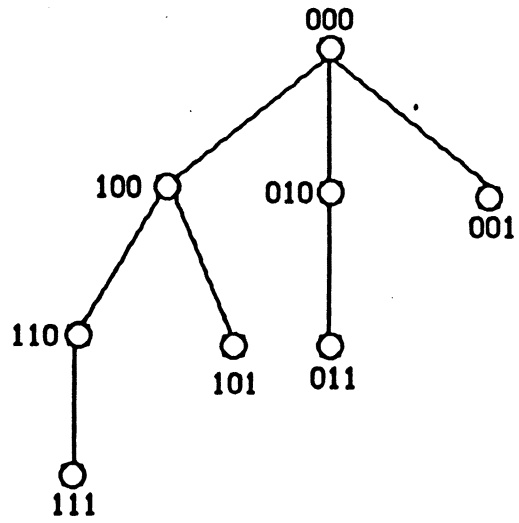


Figure 2: Structure d'arbre de communication

Algorithme de diffusion sur l'hypercube: BH

```
Seq k=[1 For n ]
  If
    Pi = ((k-1) mod p)        * Pi est la racine dans l'arbre ( Pi contient la colonne k)
      Seq
        calculer la transformation de Householder Hk
        Par
          diffuser Hk
          appliquer Hk
    True
      Seq
        recevoir Hk
        Par
          diffuser Hk
          appliquer Hk
```

Le temps de calcul est le même que celui obtenu pour l'algorithme du pipeline sur l'anneau.

Temps de communication: Le temps de communication est asymptotiquement égal à:

$$T_{comm}(p) = \sum_{k=1}^{n} [\beta_c + (m-k+2)\tau_c]\log_2(p)$$
$$= n\log_2(p)\beta_c + n(m - \frac{n}{2})\log_2(p)\tau_c + o(mn)$$

*** Diffusion avec découpage en paquets de taille optimale (BHdp)**

Pour diminuer le temps de communication, il faut diviser les données en paquets de longueur égale [ISS], puis pipeliner les données. Supposons qu'on veut envoyer q données partagées en μ paquets à P_p à partir des processeurs $P_1,..,P_{p-1}$. On pipeline les paquets de la façon suivante: à l'étape 1, le processeur P_1 envoie le premier paquet à P_2. A l'étape 2 pendant qu'il recoit le deuxième paquet à partir de P_1 le processeur P_2 envoie le premier paquet à P_3 et ainsi de suite. En général le processeur P_{j+1} recoit le premier paquet au temps j pendant que le processeur P_1 envoie le j-ème paquet. Le temps pour que le processeur P_p reçoive le dernier paquet est égal à :

$$t(\mu) = (\mu + p - 1)(\beta_c + \frac{m}{\mu}\tau_c) \text{ [ISS]}$$

$t(\mu)$ est minimal pour $\mu_{opt} = \sqrt{(p-1)m\frac{\tau_c}{\beta_c}}$

Dans ce cas on a $t_{min} = \left(\sqrt{(p-1)\beta_c} + \sqrt{m\tau_c}\right)^2$

En utilisant cette stratégie on peut éliminer le facteur $\log_2(p)$ dans la formule précédente. Dans ce cas le temps de communication est asymptotiquement égal au temps présenté par Cosnard et al [CTV] pour la méthode de Gauss, pour des matrices carrées, qui est égal à:

$$T_{comm}(p) = \gamma_{n,p}\left[\frac{(n+2)(n-1)}{2}\tau_c + \left(\left(n - \frac{p}{2}\right)(\log_2(p)-1) + \log_2\left(\left(\frac{p}{2}-1\right)!\right) + \alpha_{n,p}\left(\frac{p}{2}-1\right)\right)\beta_c\right]$$

avec $1 < \gamma_{n,p} < 2$) et $0 \le \alpha_{n,p} \le 1/2$:

Dans le cas des matrices carrées on obtient un majorant du temps de communication.

$$T_{comm}(p) = \sum_{k=1}^{n}\left(\sqrt{(\log_2(p)-1)\beta_c} + \sqrt{(m-k)\tau_c}\right)^2 \le 2n(\log_2(p)-1)\beta_c + 2n(m - \frac{n}{2})\tau_c + o(mn)$$

3.3. Algorithme du pipeline sur le réseau linéaire

Le réseau linéaire de processeurs possède des moyens de communication inférieurs à ceux de l'anneau comme le montre la figure 3. Chaque processeur a deux voisins sauf les processeurs P_0 et P_{p-1}. On décompose la matrice en p blocs, chacun formé de $\frac{n}{p}$ colonnes consécutives. Au processur P_i on affecte le bloc formé des colonnes $i\frac{n}{p}+k$ pour k=1 à $\frac{n}{p}$.

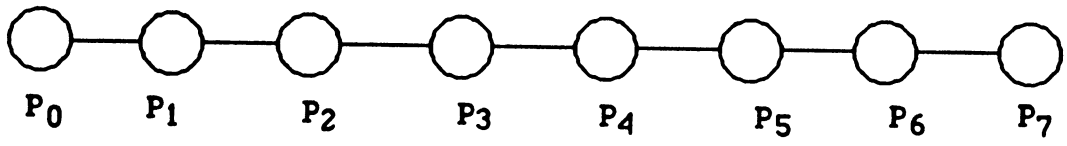


Figure 3: Réseau linéaire de processeurs

L'algorithme consiste à pipeliner, à chaque étape k la transformation H_k à partir du processeur qui contient la colonne k. Contrairement au cas de l'anneau, quand le processeur P_{p-1} reçoit H_k, il ne l'envoie pas au processeur P_0. Comme la matrice est distribuée en bloc de colonnes consécutives, chaque processeur termine son travail après $(i+1)\frac{n}{p}$ étapes.

Algorithme du pipeline sur le réseau linéaire: RL

```
Seq k=[ 1 For (i+1)n/p ]
   Seq
      If
         (k-1)< i*n/p
            Seq
               recevoir Hk à partir de Pi-1
               Par
                  envoyer Hk au processeur Pi+1
                  appliquer Hk
         True
            Seq
               calculer Hk
               Par
                  envoyer Hk au processeur Pi+1
                  appliquer Hk
```

* **Temps de calcul**: le temps de calcul est donné par P_{p-1}, le processeur qui

contient la dernière colonne. Il y a deux phases:

(i) Pendant les $n - \frac{n}{p}$ premières étapes, le processeur P_{p-1} applique la transformation de Householder au $\frac{n}{p}$ colonnes qu'il contient. Ceci est accompli en:

$$T_1(p)=\sum_{k=1}^{n-\frac{n}{p}} \frac{n}{p}[\ 3(\beta_a+\tau_a)+2(m-k)\tau_a\ ]$$

(ii) Pendant les $(\frac{n}{p}-1)$ dernières étapes, le processeur P_{p-1} calcule la décomposition de Householder de la sous matrice formée des $\frac{n}{p}$ dernières colonnes et de $(m-n+\frac{n}{p})$ dernières lignes. En utilisant le résultat du paragraphe VI.2, le temps de calcul des $(\frac{n}{p}-1)$ dernières étapes est égal à:

$$T_2(p)=\left(\frac{n}{p}\right)^2\left[m-n+\frac{2n}{3p}\right]\tau_a+\frac{3}{2}\left(\frac{n}{p}\right)^2\beta_a+\left(\frac{n}{p}\right)^2\tau_a+\frac{7n}{2p}\beta_a+o(n^2)$$

Donc le temps de calcul de l'algorithme est égal à la somme des temps de calcul des deux phases:

$$T_{cal}(p)=\frac{n^2}{p}\left[(2-\frac{1}{p})m-(1-\frac{1}{p}+\frac{1}{3p^2})n\right]\tau_a+\frac{3n^2}{2p}\left[2-\frac{1}{p}\right]\beta_a+(2-\frac{1}{p})\frac{n^2}{p}\tau_a+\frac{7}{2p}n\beta_a+o(n^2)$$

Remarque: Dans le cas des matrices carrées, le temps de calcul sur le réseau linéaire est 1.5 fois plus grand que celui sur l'anneau et sur l'hypercube (pour p assez grand).

* **Temps de communication**: le processeur P_{p-1} communique seulement pendant les $n-\frac{n}{p}$ premières étapes. Le temps de communication est donné par:

1. Le temps d'initialisation du pipeline qui prend $(p-1)((m+1)\tau_c+\beta_c)$, correspond au temps nécessaire pour envoyer la première transformation de Householder au processeur P_{p-1}.

2. Pendant les $n-\frac{n}{p}$ premières étapes le processeur P_{p-1} reçoit à partir du processeur P_{p-2} la colonne pivot supposée de longueur q. En utilisant la technique utilisée par Cosnard et al [CTV] ceci nécessite $2(q+1)\tau_c+2\beta_c$.

Donc le temps total de communication est égal à:

$$T_{comm}(p)=(1-\frac{1}{p})[\ 2n\beta_c+(2m-n+\frac{n}{p})n\tau_c\ ]+o(n^2)$$

Remarque: Dans le cas des matrices carrée, le temps de communication sur le réseau linéaire est 1.5 fois plus petit que celui de l'anneau (pour p grand).

VI.4. COMPARAISONS ET RESULTATS EXPERIMENTAUX

Le tableau 1 suivant résume les résultats de l'analyse théoriques obtenus dans les paragraphes précédents.

Algorithmes	T_{cal}	T_{comm}
PR	$\frac{n^2}{p}(m-\frac{n}{3})\tau_a+\frac{3n^2}{2p}\beta_a$	$\frac{(p-1)}{p}\left(3n(m-\frac{n}{2})\tau_c+3n\beta_c\right)$
BH	$\frac{n^2}{p}(m-\frac{n}{3})\tau_a+\frac{3n^2}{2p}\beta_a$	$\log_2(p)\left(n(m-\frac{n}{2})\tau_c+n\beta_c\right)$
RL	$\frac{n^2}{p}\left((2-\frac{1}{p})m-(1-\frac{1}{p}+\frac{1}{3p^2})n\right)\tau_a+\frac{3n^2}{2p}(2-\frac{1}{p})\beta_a$	$\frac{(p-1)}{p}\left(n(2m-n+\frac{n}{p})\tau_c+2n\beta_c\right)$

Tableau 1: Résultats de l'analyse théorique

Cas des matrices carrées (m=n)

L'analyse théorique montre que l'algorithme de diffusion sur l'hypercube avec découpage en paquets de taille optimale est meilleur que l'algorithme du pipeline sur l'anneau. Il est important de noter que le temps de communication est plus important sur l'anneau et l'hypercube que sur le réseau linéaire, avec en outre un contrôle plus simple dans ce dernier. Cependant le temps de calcul est très important: proche de 1.5 fois plus grand que l'anneau et l'hypercube. Ceci est dû au fait que sur le réseau linéaire le processeur P_i termine aprés $(i+1)\frac{n}{p}$ étapes. Par contre sur l'anneau et sur l'hypercube les processeurs travaillent tout le temps, sauf pendant les p dernières étapes. Puisque le temps de calcul est de l'ordre de n^3, asymptotiquement, le temps de calcul sur le réseau linéaire l'emporte sur le temps de communication.

Nous avons implémenté et comparé ces algorithmes sur l'hypercube T20 de FPS [GHS] installé au laboratoire TIM3 Grenoble, composé de 16 noeuds. Chaque noeud à une unité vectorielle et une mémoire de taille 1Mbytes. Sur chaque noeud on peut stocker une matrice carrée de réels (64 bits) de taille 256. Par conséquent, on peut résoudre un problème de taille 1024 sur les 16 processeurs, pour plus de detail nous renvoyons à [TV], [KT].

Dans les tableaux 2.a et 2.b, nous présentons les résultats expérimentaux que nous avons obtenu pour une matrice de taille 256 avec un nombre de processeurs variant de 1 à 16. Ces résultats correspondent bien à l'analyse théorique. Les algorithmes sur l'anneau sont meilleurs pour p=2, 4, 8, 16. Cependant pour 16 processeurs le réseau linéaire est proche de l'anneau (sans diffusion) (3.5 contre 3.2 secondes).

Le tableau 2.c compare les temps d'exécution de tous les algorithmes présentés dans les paragraphes précédent en utilisant 16 processeurs. Théoriquement l'algorithme de diffusion sur l'hypercube avec découpage en paquet de taille optimal est meilleur que l'algorithme de pipeline dans l'anneau. Les résultas expérimentaux montrent le contraire. Puisque le nombre de paquets de taille optimale obtenu pour la taille maximale (1024) est égal à 4, l'amélioration qu'apporte le découpage ne couvre pas le temps de contrôl plus élevé. Pour des tailles petite($n \leq 384$), le temps d'exécution dans le réseau linéaire est inférieur a clui de diffusion dans l'hypercube avec découpage en paquet de taille optimal.

Processurs	BR	PR	BH	RL
1	22.48	22.48	22.48	22.48
2	12.16	12.28	12.30	16.24
4	6.81	7.25	7.47	10.08
8	4.19	4.57	5.14	5.84
16	2.84	3.24	4.12	3.55

Tableau 2.a: Temps d'exécution en seconde pour une matrice de taille 256

Processurs	BR		PR		BH		RL	
	Acc	Eff	Acc	Eff	Acc	Eff	Acc	Eff
1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
2	1.85	0.92	1.83	0.92	1.83	0.91	1.38	0.69
4	3.30	0.82	3.10	0.78	3.01	0.75	2.23	0.56
8	5.36	0.67	4.92	0.62	4.37	0.55	3.85	0.48
16	7.91	0.49	6.93	0.43	5.46	0.34	6.33	0.40

Tableau 2.b: Accélération et efficacite obtenues pour une matrice de taille 256

Taille	BR	PR	BH	BHdp	RL
256	2.84	3.24	4.12	4.33	3.55
384	5.96	6.71	8.19	8.56	7.88
512	10.42	11.63	13.86	14.22	14.25
768	24.03	26.54	30.65	30.77	34.11
1024	44.97	49.26	55.81	54.86	65.15

Tableau 2.c Temps d'exécution en seconde avec 16 processeurs

Dans le tableau 2.d nous résumons les résultats obtenus pour calculer la décomposition de Householder pour des matrices carrées de taille $n_{max}(p)$ où $n_{max}(p)$ est la taille maximale du problème qu'on peut résoudre avec p processeurs. Les résultats expérimentaux montrent que les performances obtenues pour l'anneau sont très bonnes: le temps d'exécution de l'algorithme du pipeline sur l'anneau pour des matrices de taille 1024 est égal à 49 secondes comparé à 55.8 et à 54.8 secondes pour les algorithmes de diffusion sur l'hypercube et à 65 secondes pour l'algorithme du pipeline sur le réseau linéaire. En utilisant la stratégie de diffusion sur l'anneau on améliore le temps d'exécution: pour des matrices de taille 1024 le temps d'exécution est égal à 45 secondes seulement. Dans ce cas le nombre de Mégaflops est égal à 31.8. Ceci s'explique par la diminution du temps de communication. Comme Saad [Saa86] l'a conclut pour la méthode de Gauss, l'augmentation de connexions n'améliore pas l'efficacité. Les résultats expérimentaux montrent que l'anneau est la meilleure topologie pour la méthode de Householder.

Processeurs	Taille	BR	PR	BH	RL
1	256	22.48	22.48	22.48	22.48
2	384	28.67	28.78	28.84	39.06
4	512	29.04	30.32	30.79	44.34
8	768	40.04	42.41	45.16	61.02
16	1024	44.97	49.26	55.81	65.15

Tableau 2.d: Comparaison des temps d'exécution (le temps est exprimé en seconde)

Les résultats complets sont présentés dans les figures 4.a, 4.b, 4.c, 4.d et 4.e

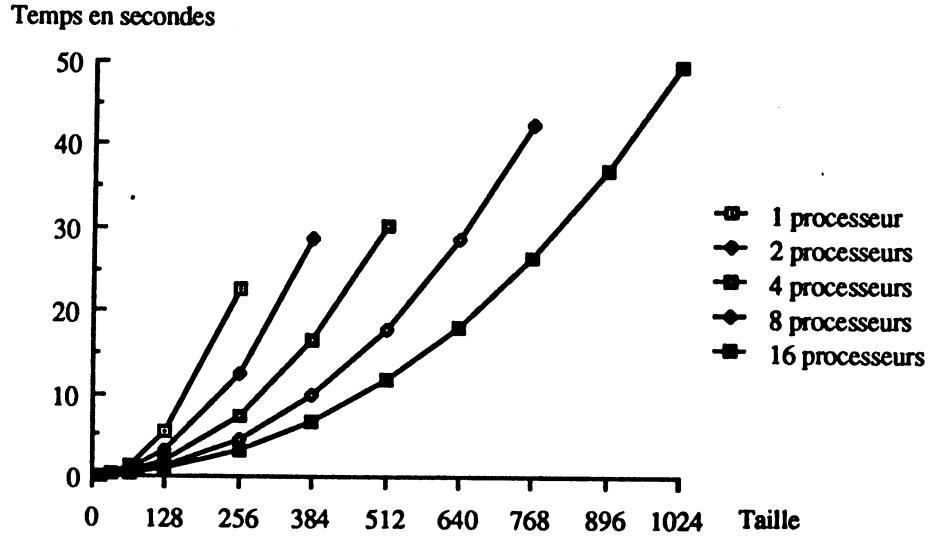


Figure 4.a: Performances sur l'anneau
Algorithme du pipeline

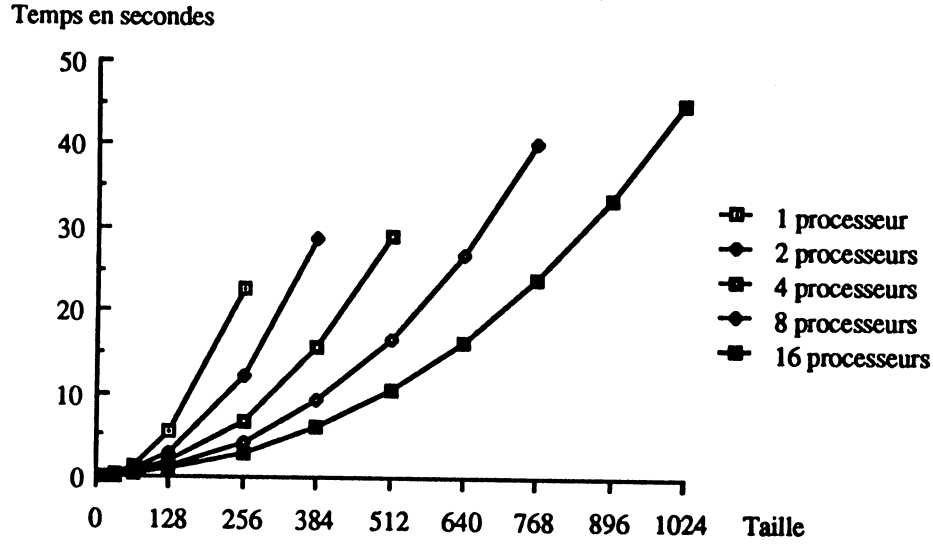


Figure 4.b: Performances sur l'anneau
Algorithme avec diffusion

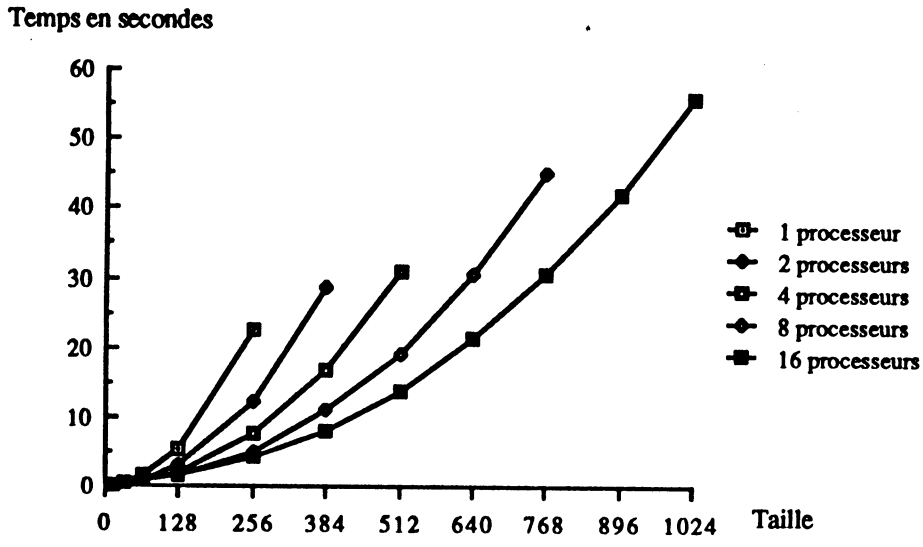


Figure 4.c: Performances de l'algorithme de diffusion sur l'hypercube

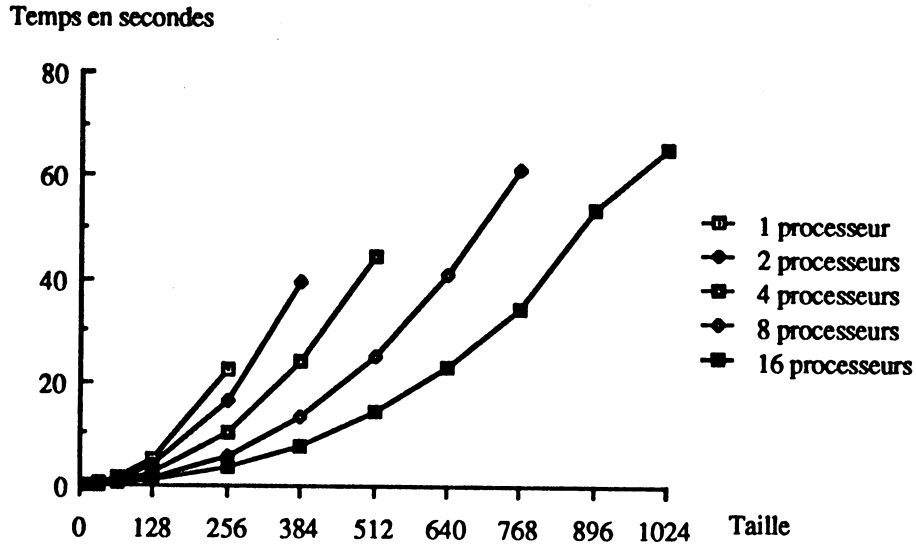


Figure 4.d: Performances de l'algorithme du pipeline sur le réseau linéaire

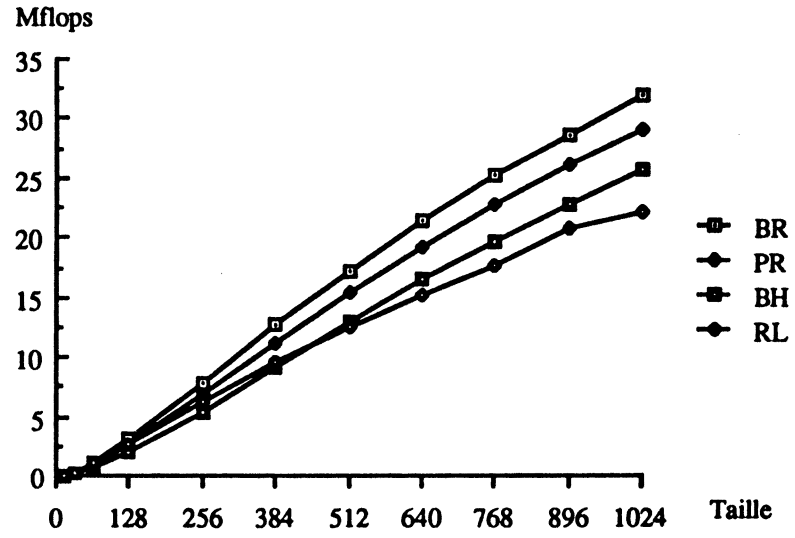


Figure 4.e: Performances en Mflops

Cas des matrices rectangulaires (m≥n)

Dans le tableau 3 nous présentons les résultats expérimentaux obtenus pour p=16. Ces résultats montrent que les performances de l'algorithme de diffusion dans l'anneau sont meilleures. Le temps d'exécution de l'algorithme de diffusion sur l'hypercube avec découpage en paquets de taille optimale est meilleur que celui de l'algorithme du pipeline sur l'anneau pour des matrices de taille suffisamment grandes (dans ce cas le nombre de paquets de taille optimale augmente) mais reste inférieur à l'algorithme de diffusion dans l'anneau. Ce dernier necéssite moins de temps de communication que l'agorithme du pipeline. Les performnaces sur le réseau linéaires sont moins bonnes lorsque les dimensions de la matrice augmentent. Celà est dû certainement au temps de calcul plus élevé dans le réseau linéaire. Les résultas théoriques et expérimentaux montrent que les performences de l'algorithme du pipeline dans l'anneau sont meilleures que celles de diffusion dans l'hypercube sans découpage en paquet de taille optimale (demande plus de temps de communication).

n	m	BR	PR	BH	BHdp	RL
16	128	0.094	0.135	0.171	0.173	0.133
	256	0.132	0.201	0.234	0.272	0.196
	512	0.205	0.335	0.371	0.378	0.325
	1024	0.350	0.599	0.644	0.577	0.577
	2048	0.645	1.145	1.205	0.878	1.099
	8192	2.414	4.424	4.572	2.700	4.239
32	128	0.195	0.279	0.347	0.351	0.230
	256	0.271	0.414	0.475	0.551	0.332
	512	0.421	0.685	0.747	0.747	0.535
	1024	0.721	1.234	1.301	1.169	0.968
	2048	1.321	2.371	2.435	1.781	1.855
	8192	4.903	9.195	9.222	5.491	7.170
64	128	0.415	0.541	0.717	0.725	0.437
	256	0.775	0.799	0.964	1.121	0.566
	512	0.881	1.350	1.516	1.505	0.950
	1024	1.500	2.483	2.570	2.351	1.742
	2048	2.737	4.795	4.796	3.665	3.353
	8192	10.169	18.696	18.170	11.320	13.039
128	256	1.252	1.561	1.977	2.164	1.313
	512	1.907	2.597	3.112	3.107	1.907
	1024	3.223	4.860	5.375	4.825	3.273
	2048	5.864	9.532	10.064	7.713	6.358
	8192	21.721	37.580	38.216	24.045	24.898
256	512	4.337	5.280	6.510	6.645	5.025
	1024	7.298	9.642	11.424	10.372	7.958
	2048	13..218	18.792	21.498	16.935	13.581
	4096	25.130	37.387	41.675	29.634	25.597
512	1024	17.664	21.015	24.975	23.641	22.638
	2048	32.111	40.513	47.740	39.578	39.390

Tableau 3: Resultats expérimentaux pour p=16

VI.5. EVALUATION DE L'ACCELERATION

Pour simplifier nous nous limitons dans ce paragraphe au cas des matrices carrées. La façon habituelle de mesurer l'amélioration due à l'utilisation de p processeurs (le facteur d'accélération) d'un algorithme est de fixer la taille du problème et de le résoudre avec un nombre variable de processeurs. Sur les architectures à mémoire distribuée le problème est différent. En effet chaque processeur possède sa propre mémoire, donc la taille du problème qu'on peut résoudre croit avec le nombre de processeurs. Par conséquent cela rend impossible sa résolution sur un seul processeur.

Recemment Gustafson [Gus] a proposé un nouveau moyen pour calculer l'accéleration sur les architectures à mémoire distribuée. Le but est de mesurer le temps d'exécution du problème de taille maximale avec p processeurs, puis de calculer le temps du même problème avec un seul processeur. Le rapport des deux temps donne l'accélération de Gustafson.

En utilisant cette définition, Cosnard et al [CRT] ont appliqué ce moyen pour estimer l'accélération pour la méthode de Gauss sur l'hypercube T20 de FPS. Leur technique consiste à calculer $\tau_{max}(p)$, le temps moyen d'une opération arithmétique que nécessite la résolution du problème en utilisant la taille maximale, qui est égal à l'inverse du nombre de Mflops. Le rapport $\tau_{max}(1)$ sur $\tau_{max}(p)$ donne l'accélération de Gustafson.

Soit M la taille de la mémoire d'un seul processeur. Donc la taille maximale d'un problème matriciel qu'on peut résoudre avec p processeurs est égale à $n_{max}(p)=(pM)^{1/2}$. Dans les tableau 4.a, 4.b, 4.c et 4.d nous mesurons l'accélération de Gustafson des algorithmes proposés dans les paragraphes précédentes. Sur l'hypercube T20 de FPS, la valeur de M=65536 mots donne $n_{max}(1)=256$. Puisqu'on tient compte de la puissance de la machine, le résultat obtenu est plus élevé que l'accélération habituelle.

NB: Le temps est exprimé en seconde et τ_{max} en microseconde

Proc	Taille	Temps	τ_{max}	Accélération	Efficacité
1	256	22.48	1.005	1.0	1.0
2	384	28.782	0.381	2.64	1.32
4	512	30.318	0.169	5.93	1.48
8	768	42.414	0.070	14.32	1.79
16	1024	49.262	0.034	29.21	1.82

Tableau 4.a: Mesures de l'accélérartion de Gustafson
Algorithme du pipeline sur l'anneau

Proc	Taille	Temps	τ_{max}	Accélération	Efficacité
1	256	22.48	1.005	1.0	1.0
2	384	28.84	0.382	2.63	1.31
4	512	30.79	0.172	5.84	1.46
8	768	45.16	0.075	13.43	1.69
16	1024	55.81	0.039	25.77	1.61

Table 4.b: Mesures de l'accélérartion de Gustafson
Algorithme du diffusion sur l'hypercube

Proc	Taille	Temps	τ_{max}	Accélération	Efficacité
1	256	22.48	1.005	1.0	1.0
2	384	39.06	0.517	1.943	0.97
4	512	44.336	0.248	4.057	1.01
8	768	61.017	0.101	9.949	1.24
16	1024	65.148	0.046	22.06	1.38

Tableau 4.c: Mesures de l'accélérartion de Gustafson
Algorithme du pipeline sur le réseau linéaire

En combinant ces idées et les résultats théoriques obtenus dans le paragraphe VI.3 on obtient une formule de $\tau_{max}(p)$ [CRT]. Soit T(n,p) le temps d'exécution d'une matrice de taille n en utilisant p processeurs. Supposons que l'arithmétique et les communications ne se chevauchent pas, alors on a $T(n,p)=T_{cal}(n,p)+T_{comm}(n,p)$ où $T_{cal}(n,p)$ est le temps de l'arithmétique et $T_{comm}(n,p)$ et le temps de

communication. On obtient l'évaluation suivante de $\tau_{max}(p)$ [CRT]:

$$\tau_{max}(p)=\frac{T(n_{max}(p),p)}{\frac{4}{3}n^3_{max}(p)} \text{ avec } n_{max}(p)=(pM)^{1/2}$$

Pour les algorithmes présentés dans le paragraphe VI.3. on obtient une évaluation théorique de $\tau_{max}(p)$:

Algorithme du pipeline sur l'anneau: PR

$$\tau_{max}(p)=(0.375\tau_a+1.125\tau_c)M^{-0.5}p^{-0.5}+(0.5\tau_a+1.125M^{-1}\beta_a+2.25M^{-1}\beta_c)p^{-1}+$$
$$(0.375\tau_a+1.125\beta_a - 1.125\tau_c)M^{-0.5}p^{-1.5}\ (1.5\beta_a -2.25\beta_c)M^{-1}p^{-2}$$

Algorithme de diffusion sur l'hypercube: BH

$$\tau_{max}(p)=0.375\tau_cM^{-0.5}p^{-0.5}\log_2 p + 0.75\beta_cM^{-1}p^{-1}\log_2 p + 0.375\tau_aM^{-0.5}p^{-0.5}+$$
$$(0.5\tau_a+1.125\beta_aM^{-1})p^{-1}+(0.375\tau_a+1.125\beta_a)M^{-0.5}p^{-1.5} +1.5\beta_aM^{-1}p^{-2}$$

Algorithme de pipeline sur le réseau linéaire: RL

$$\tau_{max}(p)=(1.5\tau_a+0.75\tau_c)M^{-0.5}p^{-0.5}+(0.75\tau_a+1.5\beta_c)M^{-1}p^{-1}+2.25.\beta_a\ M^{-0.5}p^{-1.5} +$$
$$(2.625\beta_a - 1.5\beta_c)M^{-1}p^{-2}-(0.75\tau_a+1.25.\beta_a + 0.75\tau_c)M^{-0.5}p^{-2.5}- 2.25.\tau_ap^{-3}$$

Les mesures expérimentales sur l'hypercube T20 de FPS donnent les valeurs suivantes:

τ_a=1.93 e-7 β_a=2.06 e-4 τ_c=1.15e-5 β_c=8.0 e-4.

Algorithme du pipeline sur l'anneau :

$$\tau_{max}(p)=0.0508p^{-0.5} + 0.1275p^{-1} + 0.8581p^{-1.5} - 0.0227p^{-2}$$

Algorithme de diffusion sur l'hypercube :

$$\tau_{max}=0.0168(\log_2 p).p^{-0.5}+0.0003p^{-0.5}+0.00915(\log_2 p).p^{-1}+0.100p^{-1}+$$
$$0.9086p^{-1.5} +0.0005p^{-2}$$

Algorithme du pipeline sur le réseau linéaire :

$$\tau_{max} = 0.0348p^{-0.5} + 0.1631p^{-1} + 1.8167p^{-1.5} - 0.0100p^{-2} - 0.9426p^{-2.5} - 0.04825p^{-3}$$

Le tableau 5 ci dessous montre que l'adéquation entre les résultats expérimentaux et théoriques est trés bonne. Par conséquent on peut utiliser l'estimation théorique pour prévoir l'accélération possible pour des architectures massivement parallèles. La figure 5 montre l'accélération estimée pour l'algorithme du pipeline sur l'anneau pour p variable entre 32 (S(p)=57) à 2048 (S(p)=848).

		PR		BH		RL	
Proc	Taille	τ_{exp}	τ_{theo}	τ_{exp}	τ_{the}	τ_{ex}	τ_{the}
1	256	1.005	1.014	1.005	1.014	1.005	1.014
2	384	0.381	0.397	0.382	0.389	0.517	0.573
4	512	0.169	0.163	0.172	0.160	0.248	0.254
8	768	0.070	0.071	0.075	0.074	0.101	0.107
16	1024	0.034	0.040	0.039	0.040	0.046	0.046

Tableau 5: Comparaison entre τ_{max} théorique et τ_{max} expérimentale

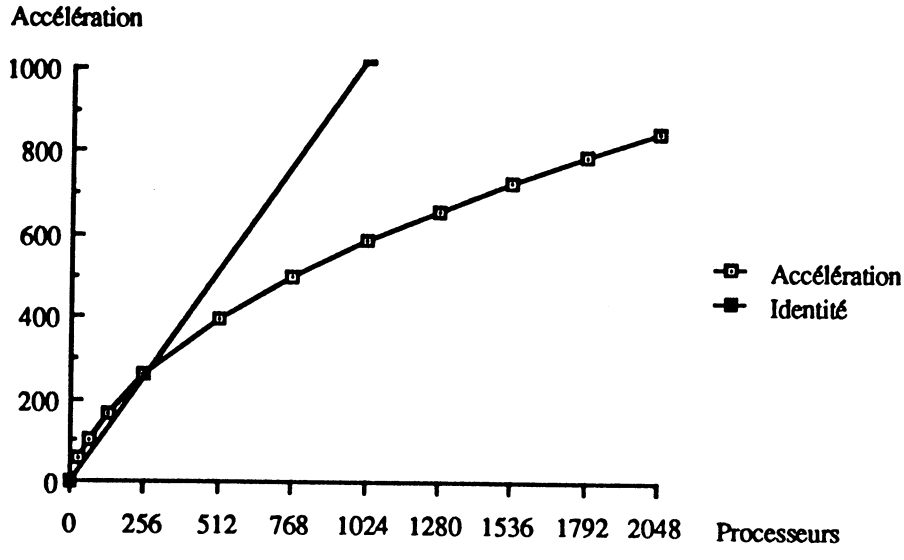


Figure 5: L'accélération estimée pour l'algorithme du pipeline sur l'anneau

VI.6. CONCLUSION

Nous avons analysé divers algorithmes pour calculer la décomposition QR d'une matrice rectangulaire de taille (m,n) ($m \geq n$) par la méthode de Householder. L'anneau de processeurs est suffisant pour atteindre des performances trés élevées. Ceci est essentiellement dû à la grande efficacité du pipeline des données à travers les canaux. Par conséquent les communications se chevauchent avec l'arithmétique. La diffusion sur l'hypercube n'est pas puissante sans découpage des données en paquets de taille optimale. Le réseau linéaire est performant en ce qui concerne les communications. Le modèle a une bonne adéquation avec les résultats expérimentaux sur l'hypercube T20 de FPS. Nous avons appliqué une nouvelle méthode de calcul de l'accélération sur les architectures à mémoire distribuée pour les algorithmes présentés dans le cas des matrices carrées. On peut alors prévoir que l'algorithme du pipeline sur l'anneau peut être utilisé avec une grande efficacité sur des machines massivement parallèles, pour des matrices de taille suffisamment grande.

CHAPITRE VII

DECOMPOSITION DE GIVENS SUR UNE ARCHITECTURE A MEMOIRE DISTRIBUEE

VII.I. INTRODUCTION

Dans ce chapitre nous présentons deux algorithmes qui ont été proposés par Pothen et al [PJV] pour calculer la décomposition de Givens d'une matrice rectangulaire de taille (m,n) sur une architecture multiprocesseur à mémoire distribuée. L'ordre d'élimination dans le premier algorithme est le même que celui dans l'algorithme de Sameh et Kuck. Dans le deuxième algorithme, le principe d'élimination est basé sur l'algorithme glouton. Nous supposons que le temps de communication et le temps de l'arithmetique sont modélisés de la même façon que dans le chapitre précédent. Les algorithmes présentés sont implémentés sur l'hypercube T20 de FPS [GHS].

VII.2. L'ALGORITHME SEQUENTIEL

Rappellons que la décomposition de Givens est accomplie en n étapes. Chaque étape k consiste à annuler les (m-k) éléments de la colonne k qui sont situés sous la diagonale principale en utilisant des rotations planes. Soit Q_k le produit de ces rotations [Ch. I].

Temps de calcul: On suppose que l'addition et la multiplication sont enchainées. Pour simplifier, nous supposons que l'évaluation d'une racine carrée est équivalente au temps d'exécution d'une opération élémentaire qu'on suppose égal à $(\beta_a+\tau_a)$.

Soient : - $T_{R(k)}$ le temps d'exécution de la rotation de Givens R(i,j,k)

$$T_{R(k)} = b(\tau_a+\beta_a)+[\ a\beta_a+a(n-k)\tau_a\ ]$$

Où $b(\beta_a+\tau_a)$ est le temps pour déterminer la rotation et $a\beta_a+a(n-k)\tau_a$ est le temps pour appliquer la rotation aux deux lignes qui la composent, avec

$\left\{\begin{array}{l} a=4 \text{ et } b=5 \text{ pour la méthode standard de Givens} \\ a=2 \text{ et } b=8 \text{ pour la méthode de Givens sans racine carrée} \end{array}\right.$

- T_{calQ_k} le temps d'exécution de Q_k à la k-ème étape

$$T_{calQ_k}=\sum_{i=k+1}^{m} T_{R(k)}$$
$$=b(m-k)(\tau_a+\beta_a)+a(m-k)\beta_a+a(m-k)(n-k)\tau_a$$
$$=(m-k)\left[(b+a(n-k))\tau_a+(a+b)\beta_a\right]$$

- T_{calseq} le temps d'exécution de la méthode séquentielle

$$T_{calseq}=\sum_{k=1}^{n}(T_{calQ_k})$$
$$T_{calseq}=\frac{a}{2}n^2\left[m-\frac{n}{3}\right]\tau_a+(a+b)\left[m-\frac{n}{2}\right]n\beta_a+\left[b(m-\frac{n}{2})-a\frac{m}{2}\right]n\tau_a+o(mn)$$

Lorsque l'addition et la multiplication sont enchainées, théoriquement, le temps d'exécution, comparé au nombre d'opérations élémentaires, est asymptotiquement divisé par 2 dans la version Givens sans racine carrée et par 3/2 dans la version standard de Givens.

VII.3. IMPLEMENTATIONS PARALLELES

Pour paralléliser la méthode de Givens, on décompose la matrice en p blocs de lignes, contrairement à la méthode de Householder, chacun de taille m/p, puis on affecte à chaque processeur un bloc. En effet pour annuler un élément, les deux lignes qui définissent la rotation de Givens sont toutes les deux modifiées.

En cas de résolution du système linéaire Ax=b on borde la matice A par le vecteur b puis on effectue sur b les mêmes transformations que sur les lignes de A. On se ramène donc à un système triangulaire supérieur, décomposé en blocs de lignes, qu'on sait résoudre par des algorithmes existants [ISS], [LC]. Dans se travail nous ne nous intéressons qu'à la parite de triangularisation.

Soient $T_{cal}(p)$ le temps de l'arithmétique, $T_{comm}(p)$ le temps de communication et $T_{exec}(p)$ le temps d'exécution d'un algorithme utilisant p processeurs.

3.1. Algorithme sur l'anneau

L'algorithme que nous présentons a été proposé par Pothen et al [PJV]. Il consiste à annuler par le même principe que l'algorithme de Sameh et Kuck [SK78], pour annuler en position (i,k) on utilise la rotation R(i,i-1,k), $1\leq i\leq m$ et $1\leq k<i$. Il annule

de bas en haut et de gauche à droite. On suppose que nous disposons d'un anneau composé de p processeurs numérotés par $P_0,\ldots,P_{p-1}$ [cf figure 1 ch. VI]. On décompose la matrice en p blocs chacun de taille m/p lignes puis on affecte à chaque processeur un bloc.
Au processeur P_0 on affecte le bloc formé des lignes m, m-p, m-2p,...
Au processeur P_1 on affecte le bloc formé des lignes m-1, m-p-1, m-2p-1,...
D'une manière général, au processeur P_i, on affecte le bloc formé des lignes m-i-kp pour $0 \leq k \leq m/p-1$.

Le principe de l'algorithme, décrit ci dessous par "algorithme SK", consiste à éliminer de la façon suivante: pour annuler un élément sur la ligne i, le processeur qui contient cette ligne recoit la ligne (i-1) à partir de son successeur, calcule la rotation puis effectue les modifications sur la ligne i. Simultanément son successeur reçoit la ligne i, calcule la rotation et modifie la ligne (i-1).

A chaque pas r, on calcule le nombre de rotations disjointes "nbrot" et on détermine la première colonne "j_1" sur laquelle on annule. Les lignes i sur lesquelles on annule sont déterminées par la relation $m+2j-(r+1)=i$ avec $j_1 \leq j \leq nbrot+j_1-1$ et $1 \leq r \leq m+n-2$ [SK78].

Algorithme SK

```
Seq r=[1 For m+n-2]
   Seq
      If
         (r≤m-1)
            Seq
               j1=1
               nbrot=min(n,⌈r/2⌉)
         True
            Seq
               j1=r-m+2
               nbrot=min(n,⌈(m-j1)/2⌉
      Seq j=[j1 For nbrot]
         i=m+2j-(r+1)
         If
            (Pk=(m-i) mod p)          * processeur qui contient la ligne i *
               Seq
                  Par
                     Envoyer la ligne i à son successeur
                     Recevoir la ligne (i-1) à partir de son successeur
                  Calculer et appliquer la rotation à la ligne i
```

(P_k=(m-i+1) mod p) * processeur qui contient la ligne i-1 *
Seq
Par
Recevoir la ligne i à partir de son prédécesseur
Envoyer la ligne (i-1) à son prédécesseur
Calculer et appliquer la rotation à la ligne (i-1)

Complexité de l'algorithme

Le temps d'exécution de l'algorithme est donné par le processeur P_0, c'est le processeur qui contient la ligne m.

Temps de calcul: Il est dificile d'évaluer le temps de calcul de l'algorithme puisque plusieurs rotations appartenant à des groupes (groupe de rotations disjointes) différents peuvent être exécutées simultanément. Nous présentons une borne supérieure du temps de calcul obtenue dans [PJV].

Le nombre maximum de groupes de rotations est égal à n. Par conséquent le nombre maximum de lignes que peut modifier un processeur est égal à $X=\lceil\frac{2n}{p}\rceil$.

Les rotations correspondantes annulent sur les colonnes $\frac{jp}{2}$ pour j=0 à X-1.

Le temps pour calculer la rotation et modifier une seule ligne de longueur $(n-\frac{jp}{2})$ est égal à $T_j=b(\tau_a+\beta_a)+\frac{a}{2}\beta_a+\frac{a}{2}(n-\frac{jp}{2})\tau_a$. Par conséquent le temps pour modifier les X lignes est égal à $\sum_{j=0}^{X-1} T_j$ qui est borné par $\frac{an^2}{2p}\tau_a+\frac{2n}{p}(b\tau_a+(\frac{a}{2}+b)\beta_a)$

Par conséquent le temps de calcul $T_{cal}(p)$ vérifie:

$$T_{cal}(p)\leq[\frac{an^2}{2p}\tau_a+\frac{2n}{p}(b\tau_a+(\frac{a}{2}+b)\beta_a)](m+n-2)$$

$$\leq\frac{an^2}{2p}(m+n)\tau_a+(a+2b)\frac{n}{p}(m+n)\beta_a+o(mn^2)$$

Temps de communication: Puisque le temps pour envoyer ou recevoire une ligne de longueur $(n-\frac{jp}{2})$ est égal à $\beta_c+(n-\frac{jp}{2})\tau_c$, le temps de communication $T_{comm}(p)$ vérifie:

$$T_{comm}(p)\leq\frac{2n^2}{p}(m+n-2)\tau_c+\frac{2n}{p}(m+n-2)\beta_c$$

3.2. Algorithme sur l'hypercube

Cet algorithme a été proposé par Chamberlain et Powel [CP], et par Pothen et al [PJH]. Une légère différence entre les deux algorithmes proposés réside dans la phase de communication. On décompose la matrice en p blocs de lignes, chacun de taille m/p. On suppose que les processeurs sont numérotés par $P_0,\ldots P_{p-1}$. Au processeur P_i on affecte le bloc composé des lignes i+1+kp, pour $0\leq k\leq m/p-1$. Par conséquent chaque colonne est décomposée en p sous colonnes, chacune de taille m/p stockée dans un processeur.

Le principe de cet algorithme consiste à annuler colonne par colonne en exploitant le caractère local de la méthode de Givens et la liberté de choix des rotations. On suppose que l'élimination dans la colonne (k+1) ne peut commencer que lorsque les éléments de la colonne k, situés sous la diagonale principale, soient tous annulés. L'algorithme est accomplie en n étapes. Dans chaque étape k, l'élimination évolue en deux phases.

* Dans la première phase dite interne " internal rotations phase (IP) " [PJH], chaque processeur utilise ses propres lignes pour annuler dans la sous colonne qu'il contient. Par conséquent à la fin de la première phase, dans chaque processeur il reste un élément à annuler, sauf dans le processeur qui contient la ligne k (la ligne qui contient l'élément diagonal).

* Dans la deuxième phase dite recursive " recursive elimination phase (RP) " [PJH], les processeurs communiquent entre eux pour annuler les (p-1) éléments restants. Donc au plus (p-1) communications sont nécessaires. Le coût de communication est donc borné par $(\beta_c+(n-k)\tau_c)(p-1)$ qui est indépendant du nombre de lignes de A, par conséquent, il devient négligeable devant le temps de calcul si le nombre de lignes est suffisamment grand par rapport au nombre de colonnes (m>>n).

a. Description des deux phases de l'étape k

Première phase: Chaque processeur effectue son travail interne indépendamment des autres processeurs. Il élimine de bas en haut dans la sous colonne qu'il contient en utilisant des rotations locales, composées par ses propres lignes. Remarquons que l'élimination dans cette phase s'effectue indépendamment du type de l'architecure disponible.

Temps de calcul de la première phase: Le temps de calcul est donné par le processeur P_{p-1} . Celui ci annule séquentiellement $(\lceil\frac{m-k}{p}\rceil-1)$ éléments sur la colonne k donc le temps de calcul est égal à:

$$T_{1,k}(p)=(\lceil \frac{m-k}{p} \rceil -1)T_{R(k)}$$
$$=(\lceil \frac{m-k}{p} \rceil -1)[(a+b)\beta_a + (b+a(n-k))\tau_a]$$

Deuxième phase: A la fin d'exécution de la première phase, il reste un élément à annuler dans chaque processeur, sauf l'élément diagonal. Par conséquent les processeurs doivent communiquer entre eux pour annuler les (p-1) éléments restants dans la colonne k. Supposons que l'architecture que nous disposons est un hypercube, donc on peut éliminer les (p-1) éléments en temps minimum qui est égal à $d=\log_2(p)$ éliminations.

Pendant chaque élimination i, les processeurs qui diffèrent du (d-i)-ème bit dans leur représentation binaire, se communiquent pour annuler l'élément situé dans le processeur le plus grand (le processeur qui a le (d-i)-ème bit égal à 1). Donc on élimine la moitié des éléments. Pendant l'élimination (i+1) on ne communique que la moitié des processeurs qui contiennent les éléments non encore annulés. Supposons que les cannaux de communication sont bidirectionnels. Par conséquent, pendant chaque élimination, chaque processeur envoie et recoit simultanément une ligne de longueur (n-k).

La figure 1 montre la structure du schéma de communication sur un hypercube composé de 8 processeurs (d=3) en supposant que la ligne k est stockée dans le processeur P_0. Sur la figure 1, deux processeurs qui communiquent pendant la ième élimination sont reliés par un trait .

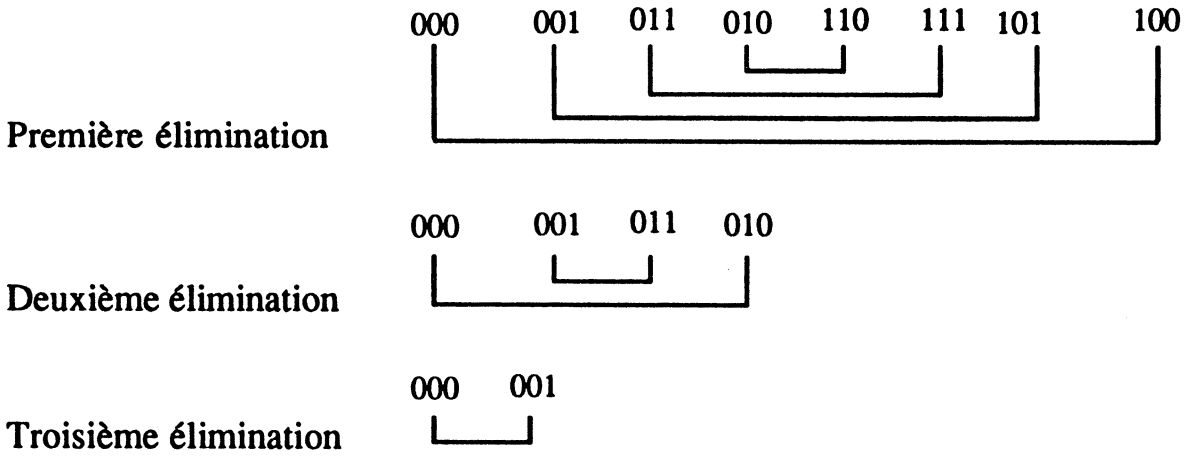


Figure 1: Structure du schéma de communication

Temps de communication de deuxième phase: Pendant chaque élimination, le processeur qui contient la ligne k envoie et reçoit simultanément une ligne de longueur (n-k). Le nombre d'éliminations étant égal à $\log_2(p)$, donc le temps de

communication est égal à: $T_{c,k}=[\ \beta_c+(n-k)\tau_c\ ]\log_2(p)$

Temps de calcul de la deuxième phase: Pendant chaque élimination, le processeur qui contient la ligne k détermine la rotation et l'applique à sa propre ligne. Le nombre d'éliminations étant égal à $\log_2(p)$, donc le temps de calcul est égal à:

$$T_{2,k}=[\ b(\tau_a+\beta_a)+\frac{a}{2}\beta_a+\frac{a}{2}(n-k)\tau_a\ ]\log_2(p)$$
$$=[\ (b+\frac{a}{2})\beta_a+(b+\frac{a}{2}(n-k))\tau_a\ ]\log_2(p)$$

Remarque: Pendant les p dernières étapes chaque processeur n'exécute que la deuxième phase qui nécessite un traitement un peu différent sans modifier les résultats asymptotiques.

b. Complexité de l'algorithme

Temps de communication: Pendant chaque étape k, les processeurs ne communiquent entre eux que pendant la deuxième phase. Par conséquent le temps de communication est égal à:

$$T_{comm}(p)=\sum_{k=1}^{n} T_{c,k}=[\ n\beta_c+\frac{n(n-1)}{2}\tau_c\ ]\log_2(p)$$
$$=n\log_2(p)\beta_c+\frac{n(n-1)}{2}\log_2(p)\tau_c$$

Temps de calcul: Le temps de calcul de l'algorithme est égal à la somme des temps de calcul dans les deux phases.

Le temps de calcul nécessaire dans la première phase est égal à:

$$T_1(p)=\sum_{k=1}^{n} T_{1,k}$$
$$=\frac{a}{2p}n^2(m-\frac{n}{3})\tau_a+\frac{a+b}{p}n(m-\frac{n}{2})\beta_a+\frac{b}{p}n(m-\frac{n}{2})\tau_a-(a+b)\left(\frac{1}{p}+\frac{1}{2}\right)n\beta_a-an^2\left(\frac{5}{4p}+\frac{1}{4}\right)\tau_a+$$
$$amn\left(\frac{1}{2p}+\frac{1}{2}\right)\tau_a+o(mn)$$
$$=\frac{a}{2p}n^2(m-\frac{n}{3})\tau_a+\frac{a+b}{p}n(m-\frac{n}{2})\beta_a+o(mn^2)$$

Le temps de calcul nécessaire par la deuxième phase est égal à:

$$T_2(p)=\sum_{k=1}^{n} T_{2,k}$$
$$=(\frac{a}{2}+b)\log_2(p)n\beta_a+(b+\frac{(n-1)}{4}a)\log_2(p)n\tau_a$$

Par conséquent le temps de calcul de l'algorithme est égal à la somme des temps de calcul obtenus dans les deux phases donc égal à:

$$T_{cal}(p)=T_1(p)+T_2(p)= \frac{a}{2p}n^2(m-\frac{n}{3})\tau_a+\frac{a+b}{p}n(m-\frac{n}{2})\beta_a+o(mn^2)$$

VII.4. COMPARAISONS ET RESULTATS EXPERIMENTAUX

Dans le tableau 1 nous résumons les résultats de l'analyse théorique pour les algorithmes présentés sur l'anneau et sur l'hypercube. Dans le cas de l'anneau, nous donnons une borne supérieure du temps de communication et du temps de calcul.

Algorithmes	T_{cal}	T_{comm}
Anneau	$\frac{an^2}{2p}(m+n)\tau_a+(a+2b)\frac{n}{p}(m+n)\beta_a$	$\frac{2n^2}{p}(m+n)\tau_c+\frac{2n}{p}(m+n)\beta_c$
Hypercube	$\frac{an^2}{2p}(m-\frac{n}{3})\tau_a+\frac{a+b}{p}(m-\frac{n}{2})n\beta_a$	$\log_2(p)(\frac{n^2}{2}\tau_c+n\beta_c)$

Tableau 1: Résultats de complexité de l'analyse théorique

L'analyse théorique montre que l'algorithme sur l'hypercube est meilleur que l'algorithme sur l'anneau. Le temps de communication dans l'anneau a le même ordre que le temps de calcul, alors que dans l'hypercube, le temps de communication est indépendant de m. Par conséquent, il peut devenir négligeable devant le temps de calcul quand m devient grande devant n.

Nous avons implémenté et testé ces deux algorithmes, pour calculer la décomposition standard de Givens d'une matrice carrée, sur l'hypercube T20 de FPS [GHS]. Les résultats expérimentaux sont en bonne adéquation avec les résultats théoriques. Les performances sur l'hypercube sont meilleures que celles sur l'anneau pour p=2, 4, 8 et 16. Dans les tableaux 2.a et 2.b nous présentons les temps d'exécution et l'évaluation de l'accélération et l'efficacité pour une matrice de taille 256.

Sur la figure 2 nous comparons les temps d'exécution en utilisant p=1, p=2, p=4, p=8 et p=16 processeurs pour l'algorithme dans l'anneau pour une matrice réel (64 bits) de taille $\leq$ 256 (taille maximale qu'on peut stocker dans un seul processeur). Comme le temps de communication est de l'ordre du temps de calcul, il influt sur le temps d'exécution de l'algorithme. Les résultats expérimentaux montrent que pour une matrice de taille inférieure à 256, les performances en utilisant un seul processeur sont meilleures que celles en utilisant 2 ou 4 processeurs. Lorsque p=8

ou 16, en utilise les p processeurs avec une trés faible efficacité (pour p=16, efficacité=0.17). Ceci montre la grande influence du temps de communication sur le temps d'exécution.

Processeurs	Taille	Temps	Accélération	Efficacité
1	256	97.67	1.0	1.0
2	256	50.68	1.92	0.96
4	256	27.79	3.51	0.88
8	256	17.17	5.69	0.71
16	256	12.86	7.59	0.47

Tableau 2.a: Résultats expérimentaux sur l'hypercube

Processeurs	Taille	Temps	Accélération	Efficacité
1	256	97.67	1.0	1.0
2	256	263.50	0.37	0.185
4	256	133.56	0.73	0.183
8	256	68.60	1.42	0.178
16	256	36.04	2.71	0.169

Tableau 2.b: Résultats expérimentaux sur l'anneau

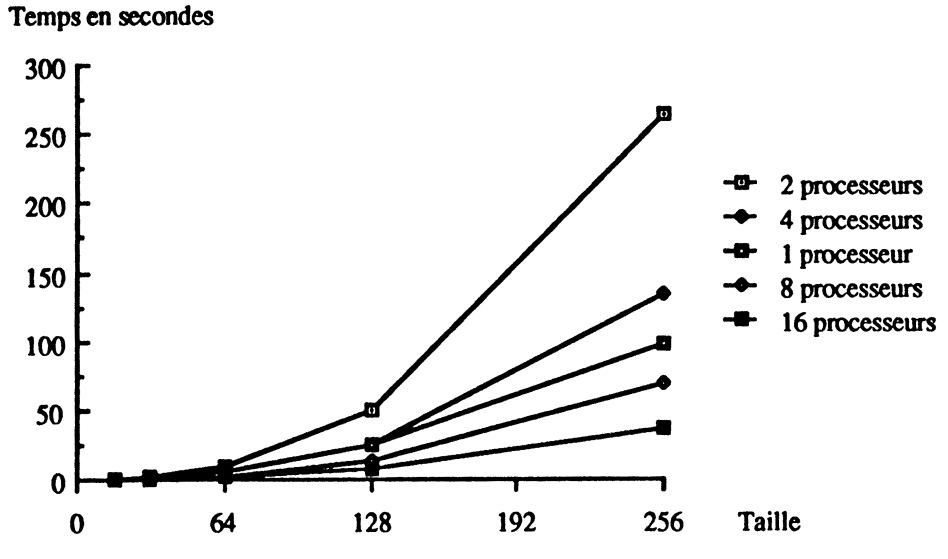


Figure 2: Comparaison des temps d'exécution sur l'anneau pour une matrice de taille inférieure à 256

Dans le tableau 3 nous résumons les résultats obtenus pour calculer la décomposition standard de Givens pour des matrices carrées de taille $n_{max}(p)$ où $n_{max}(p)$ est la taille maximale du problème qu'on peut résoudre avec p processeurs. Les résultats expérimentaux confirment les résultats théoriques. Les performances obtenues pour l'hypercube sont meilleures, dans tous les cas, à celles obtenues pour l'anneau. Pour une matrice de taille 1024 le temps d'exécution sur l'hypercube est égal à 178 secondes comparé à 1383 secondes sur l'anneau. Dans les figures 3.a et 3.b nous présentons les résultats complets des temps d'exécutions pour p=1, 2, 4, 8 et 16.

Processeurs	Taille	Anneau	Hypercube
1	256	97.67	97.67
2	384	743.50	115.52
4	512	800.37	112.05
8	768	1223.21	149.70
16	1024	1383.14	178.41

Tableau 3: Comparaison des temps d'exécution des deux algorithmes (temps en seconde)

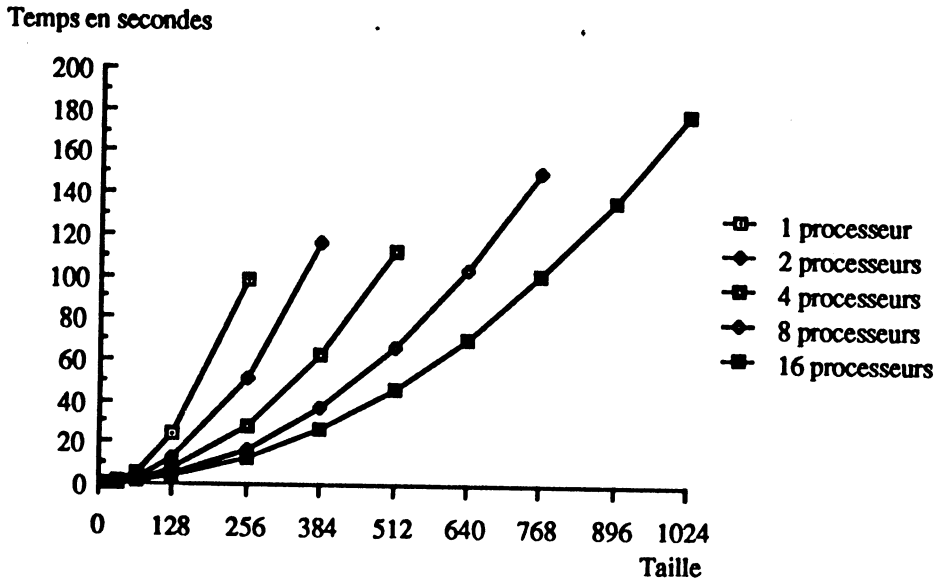


Figure 3.a: Perfomances sur l'hypercube
Givens avec racine carrée

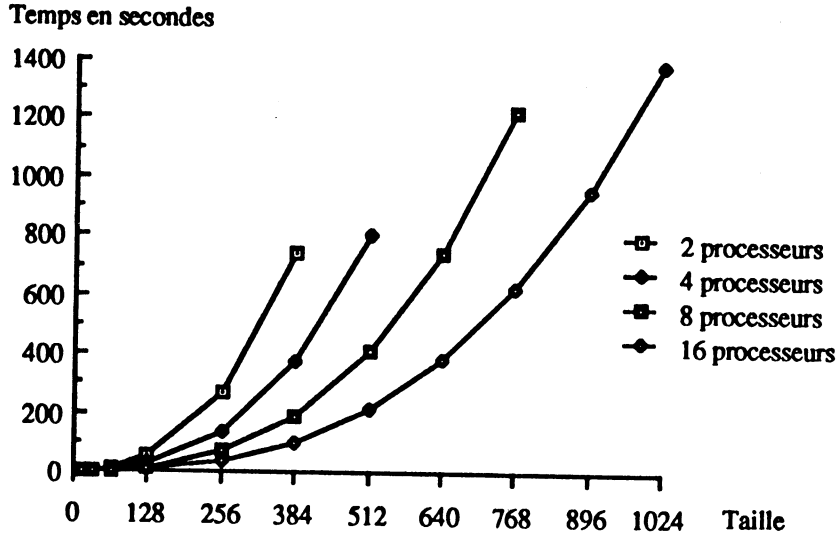


Figure 3.b: Perfomances sur l'anneau
Givens avec racine carrée

VII.5. CONCLUSION

Nous avons implémenté est testé deux algorithmes pour calculer la décomposition standard de Givens d'une matrice carrée. Les résultats théoriques est expérimentaux montrent que les performances de l'algorithme sur l'hypercube sont meilleures que celles obtenues pour l'anneau. Le temps de communication dans l'algorithme proposé pour l'hypercube est d'ordre inférieur que le temps de calcul, alors que le temps de communication dans l'algorithme proposé pour l'anneau est très important (du même ordre que le temps de calcul).

Comparées aux implémentations parallèles de la méthode de Householder présentées dans le chapitre précédent, l'analyse théorique montre que les performances des algorithmes proposés pour la méthode de Givens sont moins bonnes. Dans le tableau 4 nous comparons les résultats expérimentaux obtenus pour l'algorithme de diffusion dans l'anneau pour la méthode de Householder et pour l'algorithme sur l'hypercube pour la méthode standard de Givens en utilisant 1, 2, 4, 8 et 16 processeurs. Pour une matrice de taille 1024, le temps d'exécution de la méthode de Householder est égal à 45 secondes comparé à 178 secondes pour la méthode standard de Givens.

Processeurs	Taille	Householder	Givens
1	256	22.48	97.67
2	384	28.67	115.52
4	512	29.04	112.05
8	768	40.04	149.70
16	1024	44.97	178.41

Tableau 4: Comparaison des temps d'exécution
(le temps est exprimé en seconde)

Pour une matrice de taille 256 le temps d'exécution expérimental de la méthode séquentielle de Givens est 4.34 fois plus grand que le temps d'exécution de la méthode séquentielle de Householder alors que le rapport théorique est égal à 2. En effet, dans la pratique les opérations vectorielles n'ont pas le même temps d'exéctution (pour les mésures expérimentales des opérations vectorielles sur l'hypecube T20 de FPS nous renvoyons à [KT]). Pour la méthode de Givens on effectue deux opérations "scalaire vecteur" et deux "saxpy" alors que dans la méthode Hoseholder on effectue un "produit scalaire" et un "saxpy". D'autre part, théoriquement, le temps de calcul de la partie séquentielle de la méthode de Givens (la partie qui génére les rotations) est du même ordre que celui de la partie séquentielle de la méthode de Householder (la partie qui détermine la

transformation de Householder). En pratique c'est différent, car la partie séquentielle de la méthode de Givens est scalaire est donc réalisée sur le transputer par conséquent demande plus de temps que la partie séquentielle de la méthode de Householder qui consiste à calculer un produit scalaire. Celui ci s'effectue rapidement sur l'unité vectorielle.

CONCLUSION

Nous avons étudié la complexité de la décomposition orthogonale en parallèles d'une matrice rectangulaire de taille (m,n) par les méthodes de Givens et de Householder sur différents types d'architectures multiprocesseurs composée de p processeurs.

Dans la première partie, différentes analyses de la complexité de la parallélisation de la méthode de Givens sur une architecture à mémoire partagée sont examinées.

Dans les chapitres II et III, l'unité est prise égale au temps d'exécution d'une rotation de Givens indépendamment de la longueur des vecteurs qui la composent. Nous avons étudié le temps d'exécution minimum de ce problème dans le cas où p est quelconque (inférieur à $\lfloor m/2 \rfloor$) et lorsque $p=\lfloor m/2 \rfloor$ (parallélisme maximal).

Dans le chapitre II, consacré au cas $p=\lfloor m/2 \rfloor$, nous avons présenté les algorithmes parallèles existants avec les resultats de complexité. Nous avons introduit les algorithmes de Fibonacci modifié d'ordre i et des résultats concernant le contrôle d'activité des processeurs pour les algorithmes de Fibonacci et de Fibonacci modifié d'ordre 1. Ces resultats représentent une étape intermédiaire pour résoudre le problème dans le cas où p est inférieur à $\lfloor m/2 \rfloor$. L'algorithme glouton introduit indépendamment par Cosnard et Robert [CR] et par Modi et Clarcke [MC] est optimal pour tout m et n. Dans le cas $m>n^2$, l'algorithme glouton est optimal mais on ne connait pas son temps d'exécution.

Le chapitre III résoud entièrement le problème pour des matrices carrées dans le cas $p<\lfloor m/2 \rfloor$. Nous avons montré que le nombre minimum de processeurs p_{opt} qui permet de calculer la décomposition de Givens d'une matrice carrée de taille n en temps optimal T_{opt} est égal à $n/(2+\sqrt{2})$. Lorsque le nombre de processeurs p vérifie $1 \leq p \leq p_{opt}$, nous avons montré comment construire un algorithme asymptotiquememnt optimal en $T_{opt}(p)$. Dans le cas où $p \geq p_{opt}$ il suffit d'utiliser p_{opt} processeurs pour exécuter l'algorithme, asymptotiquement, en T_{opt}. Le problème d'optimalité reste ouvert dans le cas des matrices rectangulaires: construction et temps d'exécution de l'algorithme optimal.

Dans le chapitre IV, nous supposons que l'unité est égale au temps d'une opération élémentaire. Par conséquent le temps d'exécution de la rotation dépend de la position de l'élément annulé. Dans la première partie de ce chapitre nous supposons que le nombre de processeurs est non limité et travaillent en mode synhrone. Chaque processeur exécute une opération élémentaire. Nous avons proposé une version parallèle pour calculer la décomposition standard de Givens qui permet d'améliorer les résultats proposés dans la littérature [SK78] (8n-o(n) comparé à 12n-o(n)) en utilisnat $O(n^2)$ processeurs (dans le cas des matrices carrées). Nous

déduisons la conjecture suivante: $T_{opt}=4r_{opt}$ où r_{opt} est le nombre optimal de groupes de rotations disjointes. Dans la deuxième partie du chapitre nous supposons que le nombre de processeurs est limité à O(n) et fonctionnent en mode asynchrone pour calculer la décomposition de Givens d'une matrice carrée de taille n. A l'aide du formalisme du graphe des tâches nous avons proposé un ordonnancement qui améliore les résultats proposés dans la littérature [LKK]. Une autre approche consiste à synchroniser les processeurs à la fin de chaque pas. Dans ce cas le temps d'exécution que nous avons obtenu pour l'algorithme glouton est égal à $an^2+o(n^2)$ avec a=4 si la racine carrée est évité, sinon a=6. Plusieurs questions restent ouvertes:(i) Quel est le temps optimal pour calculer la décomposition de Givens ? (Réponse $an^2+o(n^2)$?)
(ii) Etant donnée n et p quel est le temps optimal avec p processeurs ?

La deuxième partie de la thèse est consacrée à l'analyse de la complexité des méthodes de Givens et de Householder sur une architecture multiprocesseur à mémoire distribuée.

Dans le chapitre V nous avons supposé qu'en une unité de temps chaque processeur lit une ligne, effectue son traitement local et transmet une nouvelle ligne. L'approche que nous adoptons prend en compte les coûts de communication. Nous avons comparé les résultats de complexité de ces deux méthodes, dans le cas des matrices carrées, sur deux types d'architectures à mémoire distribuée communiquant avec l'exétrieur par un canal: le réseau linéaire et l'anneau [CDT], [Sam85]. Les résultats obtenus montrent l'influence de l'architecture sur les algorithmes considérés. Ensuite nous avons montré comment varie la complexité lorsque'on augmente les moyens de communication avec l'exterieur dans le cas de l'anneau. Dans ce cas nous avons construit des algorithmes asymptotiquement optimaux pour un nombre de processeurs quelconque.

Dans le chapitre VI nous avons analysé et comparé plusieurs algorithmes pour calculer la décomposition de Householder [CDa]. Le modèle s'adapte bien avec les résultats expérimentaux sur l'hypercube T20 de FPS. A l'aide de la nouvelle méthode pour calculer l'accélération sur les architectures à mémoire distribuée, on peut prévoire que l'algorithme du pipeline sur l'anneau, pour des matrices carrées, peut être utilisé avec une grande efficacité sur des machines massivement parallèles pour des matrices de tailles suffisamment grandes.

Dans le chapitre VII nous avons implementé deux algorithmes proposés par Pothene et al [PJV] pour calculer la décomposition standard de Givens d'une matrice carrée. Les resultats expérimentaux sur l'hypercube T20 de FPS montrent que les performances de l'algorithme sur l'hypercube sont meilleures que celles sur l'anneau.

REFERENCES

[Amd] G.M. AMDAHL, "Validity of the single-processor approach to achieving large-scale computing capabilities", in AFIPS Conference Proceedings 30, AFIPS Press (1967)

[AB] M. AUGUIN, F. BOERI, "Réseau d'interconnexion et leurs commandes asynchrones", dans Parallélisme, communication et synchronisation, Ed. J.P. VERJUS et G. ROUCAIROL, 489-517

[BKK] A. BOJANCZYK, R.P. BRENT, H.T. KUNG, "Numerically stable solution of dense systems of linear equations using mesh-connected processors", Tech. Rep, Carnegie Mellon University (1981)

[BGH] M. BERRY, K. GALLIVAN, W. HARROD, W. JALBY, S. LO, U. MEIER, B. PHILIPPE, A.H. SAMEH, "Parallel algorithms on the CEDAR system", in CONPAR 86 (G. Goos and J. Hartmanis eds), Lecture Notes in Computer Sciences 237, Springer Verlag (1986), 25-39

[Cos] M. COSNARD, cours INRIA-CNRS Algorithmique distribuée, La Colle sur Loup (1987)

[Csa] L. CSANKY, "Fast parallel matrix inversion algorithms", SIAM Review 5,4, (1976), 618-623

[CDe] E.G. COFFMAN Jr., P.J. DENNING, Operating system theory, Prentice Hall, Englewood Cliffs, NJ, (1973)

[CDa] M. COSNARD, E.M. DAOUDI, "Householder factorization on distributed architectures", Proceedings of the International Meeting on Parallel Computing: Methods, Algorithms, Applications, Verona, 28-30 september 1988. Adam Hilger. eds. D.J. EVANS and C. SUTTI Editors

[CDMR] M. COSNARD, E.M.DAOUDI, J.M. MULLER, Y. ROBERT, "On parallel and systolic Givens factorisation of dense matrix", in Parallel Algorithms and Architectures, Eds. M. COSNARD et al., North-Holland (1986), 245-258

[CDR] M. COSNARD, E.M.DAOUDI, Y. ROBERT, "Complexity of parallel Givens factorization on shared memory architectures", Rapport de Recherche ENS Lyon (1989)

[CDT] M. COSNARD, E.M. DAOUDI, B. TOURANCHEAU, "Communication dans les réseaux de processeurs et complexité des algorithmes", Actes du

Colloque C3, Angoulème (1987), RR 636, IMAG Grenoble (1986).

[CMR] M. COSNARD, J.M. MULLER, Y. ROBERT, "Parallel QR decomposition of a rectangular matrix", Num. Math. 48, (1986), 239-249

[CMRT] M. COSNARD, M. MARRAKCHI, Y. ROBERT, D. TRYSTRAM, "Parallel Gaussian elimination on an MIMD Computer", Par. Comp. (1987)

[CP] R.M. CHAMBERLAIN, M.J. POWELL, "QR factorization for linear least-squars problems of hypercube multiprocessor", IMA J. Numerical Analysis 8, (1988), 401-413

[CR] M. COSNARD, Y. ROBERT, "Complexity of parallel QR factorization", J. ACM 33, 4, (1986), 712-723

[CR83] M. COSNARD, Y. ROBERT, "Complexité de la factorisation QR en parallèle", C. R. Acad. Sc. Paris, t. 297, série I, (1983), 137-139

[CR86] M. COSNARD, Y. ROBERT, "Systolic Givens factorization of dense rectangular matrices", apparu dans Intert. Journal computer Mathématics

[CR87] M. COSNARD, Y. ROBERT, "Algorithmique parallèle: une étude de complexité", Techniques et Science Informatique 6, 2, (1987), 115-125

[CRBS] P. CARNEVALI, G. RADICATI di BROZOLO, Y. ROBERT, P. SGUAZZERO, "Efficient FORTRAN implementation of the Gaussian elimination and Householder reduction algorithms on IBM 3090 vector multiprocessor", IBM, ICE-0012, Rome, (1987)

[CRT] M. COSNARD, Y. ROBERT, B. TOURANCHEAU, "Evaluating speedups on distributed memory architectures", Parallel Computing V 10, N 2, (1989) 247-253

[CTV] M. COSNARD, B. TOURANCHEAU, G. VILLARD, "Gaussian elimination on message passing architecture", in Supercomputing, E.N. Houstis et al. eds., Lecture Notes in Computer Science 297, Springer Verlag (1988), 611-628

[Dua] J. DUATO, "Parallel triangularization of a sparse matrix on distributed memory multiprocessor using fast Givens rotations", Preprint

[DI] J.M. DELOSME, I.C.F. IPSEN, "Systolic array synthesis", in Parallel Algorithms and Architectures, Eds. M. COSNARD et al., North-Holland (1986), 295-312

[Ens] P.H. ENSLOW, Multiprocessor and parallel processing, John Willey and Sons, New York (1984)

[Fen] T. FENG, "Some characteristics of associative parallel processing", Proc. of the 1972 Sagamore Comp. Conf., Syracuse University (1972), 5-16

[ED] D.J. EVANS, R.C. DUMBAR, "The parallel solution of triangular systems of equations", IEEE Tran. Computers, C-32 (1983), 201-204

[Flyn] M.J. FLYNN, "Very high-speed computing systems", Proc. IEEE 54, (1966), 1901-1909

[FH] C. FRABOUL, N. HIFDI, "LESTAP : expression et mise en oeuvre d'applications parallèles", R.R. 1/3192, ONERA-CERT, Toulouse (1984)

[Gen73] W.M. GENTLEMAN, "Least squares computation by Givens transformations without square roots", J. Inst. Math. Appl. 12 (1973), 329-336

[Gen75] W.M. GENTLEMAN, "Error analysis of QR decomposition by Givens tranformations", Linear Algebra and Appl. 10 (1975), 189-197

[Giv] W. GIVENS, "Computation of plane unitary rotations transforming a general matrix to triangular form", J. SIAM 6, 1 (1958), 26-50

[Gus] J.L. GUSTAFSON, "The scaled-sized model: a revision of Amdahl's law", in ICS Supercomputing'88, L.P. Kartachev and S.I. Kartachev eds., International Supercomputing Institute Inc., vol. II (1988), 130-133

[GHS] J.L. GUSTAFSON, S. HAWKINSON and K. SCOTT, "The architecture of a homogeneous vector supercomputer", Journal Par. Distr. Comp. 3 (1986) 297-304

[GP] G.H. GOLUB, R. PLEMMONS, "Large-scale geodetic least-squares adjustment by dissection and orthogonal decomposition", Linear Algebra and Appl. 38 (1980), 3-28

[GVL] G.H. GOLUB, C.F. VAN LOAN, Matrix Computations, The John Hopkins Univ. Press (1983)

[GVR] D. GANNON, J. VAN ROSENDALE, "On the impact of communication in the design of parallel algorithms", IEEE T.C. 33, 12 (1984), 1180-1194

[Hel] D. HELLER, "A survey of parallel algorithms in numerical linear algebra", Siam Review 20 (1978), 740-777

[Hoa] C.A.R. HOARE, Comunicating Sequential Processes, Prentice Hall, Series Computer Sciences (1985)

[Hous] A.S. HOUSEHOLDER, "Unitary triangularization of a non symetric matrix", J. ACM 5, (1958), 339-342

[HB] K. HWANG, F. BRIGGS, Parallel processing and computer architecture, Mc Graw Hill (1984)

[ISS] I.C.F. IPSEN, Y. SAAD, M.H. SCHULTZ, "Complexity of dense linear system solution on a multiprocessor ring", RR 349, Computer Science Dpt, yale University (1985)

[Kum] S.P. KUMARD, "Parallel algorithms for solving linear equations on MIMD computers", PhD. Thesis, Washington State University (1982)

[Kun] H.T. KUNG,"Why systolic architectures", IEEE Comp. 15 (1982),37-46.

[KAP] S. KIM, D.P. AGRAWAL, R. PLEMMONS, "Householder orthogonalization on a distributed memory multiprocessor", preprint

[KL] H.T. KUNG, C.E. LEISERSON, "Systolic arrays for (VLSI)", in Proc. of the Symposium on Sparse Matrices Computations, I.S. Duff and G.W. Stewart eds, Knoxville, Tenn. (1978), 256-282

[KT] S. KUPPASWAMI, B. TOURANCHEAU, " Evaluating the performances of a transputer based hypercube vector computer", Mars 1988, R. interne

[LC] G. LI, T.F. COLEMAN, "A parallel triangular solver for a hypercube multiprocessor", Hypercube Multiprocessors 1986, M.T. Heath ed., SIAM (1986), 539-551. TR 86-787, Dept. of Computer Science, Cornell University, Ithaca, NY

[LH] C. LAWSON, R. HANSON, Solving least squares problems, Prentice-Hall (1974)

[LKK] R.E. LORD, J.S. KOWALIK, S.P. KUMAR, "Solving linear algebraic equations on an MIMD computer", J. ACM 30 (1), 1983, p 103-117

[Mol] C. MOLER, "Matrix computations on distributed memory multiprocessors", Hypercube Multiprocessors 1986, M.T. Heath ed., SIAM (1986), 161-180

[MC] J.J. MODI, M.R.B. CLARKE, "An alternative Givens ordering", Num. Math. 43 (1984), 83-90

[MR] M. MARRAKCHI, Y. ROBERT, "Optimal scheduling algorithms for parallel iterative methods on multiprocessors systems", RR 693, (1988)

[PJV] A. POTHEN, S. JHA, U. VEMULAPATI, "Orthogonal factorization on a distributed memory multiprocessor", Hypercube Multiprocessor 1987,Eds. M.T. HEATH, (1987), 587-596

[Qui] P. QUINTON, "The systematic design of systolic arrays", R.R. 193 IRISA (1983)

[RTr] Y. ROBERT, D. TRYSTRAM, "Optimal schedeling algorithms for parallel Gaussian elimination", Proceedings of EUCOPE 87, North Holland à paraître

[RTo] Y. ROBERT, B. TOURANCHEAU, "LU and QR factorisation on the FPS T series hypercube", in Compar 88, Manchester, U.K., september (1988)

[Saa85] Y.SAAD, "Communication complexity of Gaussian elimination algorithm on multiprocessors", R R 348, Computer Science Dpt., Yale University (1985)

[Saa86] Y. SAAD, "Gaussian elimination on hypercube, in Parallel Algorithms and Architectures", Eds. M. COSNARD et al., North-Holland 1986, 5-18.

[Sam77] A. SAMEH, "Numerical parallel algorithms - a survey", in High Speed Computer and Algorithm Organization, D.Kuck, D.Lawrie and A.Sameh eds, p 207-228, Academic Press (1977)

[Sam82] A. SAMEH, "Solving the linear least squares problem on a linear array of processors", Proc. Purdue Workshop on algorithmically-specialized computer organizations, W. Lafayettte, Indiana, September 1982

[Sam83] A. SAMEH, "An overview of parallel algorithms", Bull. EDF, C1 (1983)

[Sam85] A SAMEH, "On some parallel algorithms on a ring processors", Computer Phys. Com. 37 (1985) 159-166

[Sch] U. SCHENDEL, Introduction to numerical methods for parallel computers, Ellis Horwood Series, J. Wiley & Sons, New York (1984)

[Sei] C.L. SEITZ, "The cosmic cube", comm. ACM 28,1 (1985), 22-33

[SK74] A. SAMEH, D.J. KUCK, "Linear system solvers for parallel computers", Rep. No. UIUCDCS-R-75-701, Dept. of Comp. Sci., U. of Illinois at Urbana-Champaign, Urbana, Ill. Dec. (1974)

[SK78] A. SAMEH, D.J. KUCK, "On stable parallel linear system solvers", J. ACM 25, 1 (1978), 81-91

[SS] Y. SAAD, M.H. SCHULTZ, "Topological properties of hypercubes", RR 389, Computer Science Dpt., Yale University (1985)

[SS1] Y. SAAD, M.H. SCHULTZ, "Data communication in hypercubes", J. Par. Distr. Comp. V 3, N 1, (1989) 115-135

[SS2] Y. SAAD, M.H. SCHULTZ, "Parallel direct methods for solving banded linear systems", RR 387, Computer Science Dpt., Yale University (1985)

[SVL] R. SCHREIBER, C. VAN LOAN, "A storage efficient WY representation for products of Householder transformations", TR 87-864, Dept. of Comp. Sc. Cornell University, Ithaca, New York (1987)

[Tou] B. TOURANCHEAU, "Algorithmique parallèle pour des machines à mémoire distribuée (applications aux algorithmes matriciels)", Thèse de l'INPG, Février (1989)

[TV] B. TOURANCHEAU, G. VILLARD, "Manuel d'utilisation de l'hypercube T20 de FPS", Mars 1988, Rap. interne, IMAG.

[VL] C. VAN LOAN, "A block QR factorization scheme for loosely coupled systems of array processors", TR 86-797, Dept. of Comp. Sc. Cornell University, Ithaca, New York (1986)

[Wil] P. WILSON, "OCCAM architecture eases system design-part 1", Computer design (1983), 107-115

AUTORISATION de SOUTENANCE

VU les dispositions de l'Arrêté du 23 novembre 1988 relatif aux Etudes doctorales

VU les rapports de présentation de Messieurs

- ROBERT Yves, Professeur
- TCHUENTE Maurice, Professeur

Monsieur DAOUDI El Mostafa

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 25 avril 1989

Pour le Président de l'I.N.P.G.
et par délégation,
le Vice-Président
P. VENNEREAU