Techniques d'ordonnancement et algorithmique parallèle en algèbre linéaire

Mounir Marrakchi

THESE

présentée par

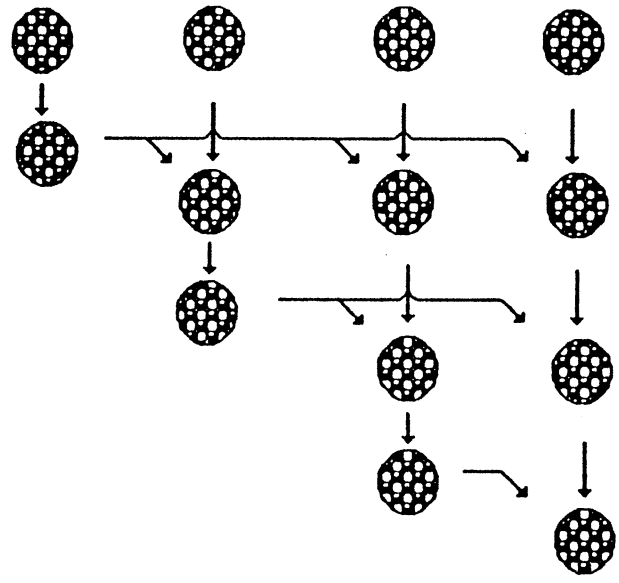
Mounir MARRAKCHI

pour obtenir le titre de DOCTEUR de

l'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

(arrêté ministériel du 5 juillet 1984)

en Mathématiques Appliquées.



TECHNIQUES D'ORDONNANCEMENT ET ALGORITHMIQUE PARALLELE EN ALGEBRE LINEAIRE.

Date de soutenance: 6 Juillet 1988

Composition du Jury:

F. ROBERT	Président
M. COSNARD	
J. M. LABORDE	Examinateurs
Y. ROBERT	
P. SPITERI	

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

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Président : Georges LESPINARD

Année 1988

Professeurs des Universités

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

Professeur Université des Sciences Sociales (Grenoble II)

BOLLIET Louis

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

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

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

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

Directeurs de recherche 2ème Classe

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

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

E.N.S.E.E.G

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

E.N.S.E.R.G

BOREL Joseph

E.N.S.I.E.G

DESCHIZEAUX Pierre
GLANGEAUD François
PERARD Jacques
REINISCH Raymond

E.N.S.H.G

ROWE Alain

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

COURTIN Jacques

E.F.P.

CHARUEL Robert

C.E.N.G

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

Laboratoires extérieurs

C.N.E.T

DEVINE Rodericq
GERBER Roland
MERCKEL Gérard
PAULEAU Yves

UNIVERSITE Joseph FOURIER (GRENOBLE I)

Président de l'Université :
M. PAYAN Jean Jacques

Année Universitaire 1987 - 1988

MEMBRES DU CORPS ENSEIGNANT DE SCIENCES ET DE GEOGRAPHIE

PROFESSEURS DE 1ère Classe

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

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

PROFESSEURS de 2ème Classe

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

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

MEMBRES DU CORPS ENSEIGNANT DE L' IUT 1

PROFESSEURS de 1ère Classe

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

PROFESSEURS de 2ème classe

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

PROFESSEURS DE PHARMACIE

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

MEMBRES DU CORPS ENSEIGNANT DE MEDECINE

PROFESSEURS CLASSE EXEPTIONNELLE ET 1ère CLASSE

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

à ma femme,

à mes frères et à ma sœur.

Je tiens à exprimer toute ma reconnaissance à Monsieur Yves ROBERT qui a guidé mes premiers pas dans la recherche, pour son soutien, ses conseils et sa bienveillance durant l'élaboration de cette thèse.

Mes vifs remerciments vont également à:

Monsieur François ROBERT, Professeur de l'ENSIMAG, pour l'honneur qu'il me fait en présidant ce jury de cette thèse.

Monsieur Michel COSNARD, Professeur de l'ENSL, pour avoir accepté de faire partie de ce jury. Qu'il veuille bien agréer l'expression de mes sincères et respectueuses considérations.

Monsieur Jean Marie LABORDE, Directeur de recherches du laboratoire LSD pour avoir accepté de siéger dans le jury.

Monsieur le Professeur Pierre SPITERI pour avoir accepté de juger cette thèse et se déplacer pour participer au jury.

Merci également à tous les membres du laboratoire TIM3 et en particulier les membres de l'équipe Algorithmique Parallèle pour la contribution que chacun d'eux a pu m'apporter.

De même je remercie tous mes enseignants pour leur participation dans ma formation scientifique et intellectuelle.

Que tous mes amis et collègues qui ont rendus mon séjour en France agréable soient vivement remerciés.

Je remercie toute l'équipe de service de reprographie pour l'excellente qualité de leur travail.

TABLE DES MATIERES

INTRODUCTION

1. INTRODUCTION

Les architectures parallèles peuvent reposer sur deux principes [46]:
- la concurrence
- le traitement à la chaîne, aussi appelé pipe-line.

La concurrence intervient lorsque plusieurs opérations ou séquences d'opérations peuvent être effectuées simultanément par des processeurs différents. Le pipe-line correspond lui au cas où des séries de données subissent, en circulant d'une unité à une autre, une même succession de traitements élémentaires. Les deux niveaux de parallélisation peuvent être utilisés à la fois.

Le concept de parallélisme rompt avec l'approche classique qui consiste à gagner de la vitesse en effectuant plus vite chaque opération. En calcul parallèle, le gain de vitesse provient de la réalisation simultanée de plusieurs opérations.

Un grand nombre de papiers concernant la programmation parallèle des algorithmes d'algèbre linéaire et non linéaire [59] a été publié. Certains algorithmes parallèles exploitent les propriétés de la machine sur laquelle l'algorithme parallèle est implémenté. Par exemple : Hypercube, Denelcor HEP, Illiac IV, IBM 3090, Réseaux Systoliques, CDC Star-100, Cray 1. Heller [26], Sameh [56], Dongarra et Coll. [15], Eisenbei et Erhel [16], Erhel et Coll. [17, 18], Hoffman [28] et enfin Robert et Sguazzero [49], ... se sont interéssés à la vectorisation sur des multiprocesseurs vectoriels. Les travaux de Cosnard et Coll. [8], Cosnard et Robert [11], Cosnard, Robert et Trystram [10], Kumar [32], Lord et Coll. [37], Marrakchi et Robert [39, 40, 41], Missirlis et Tajaferis [42], Polychronopoulus et Banerjee [44], Polychronopoulus et Kuck [45], ... sont dirigés vers la parallélisation à l'aide du formalisme du graphe de tâches sur des architectures à mémoire partagée. Enfin, Ahmed et Coll. [1], Gentleman et Kung [24], ... ont proposé des solutions systoliques.

A l'aide du formalisme du graphe des tâches, nous étudions dans cette thèse la conception d'algorithmes parallèles. Ce formalisme du graphe de tâches consiste à découper un algorithme séquentiel en tâches. Ces dernières sont reliées par une relation d'ordre partiel. Le graphe d'ordonnancement rend compte de ces contraintes temporelles [32].

Nous exécutons à la fois plusieurs programmes indépendants. Ces derniers sont des tâches indépendantes de l'algorithme séquentiel initial. L'ordre d'exécution de ces tâches est en fonction d'utilisation des données. Reste enfin, d'affecter les tâches aux processeurs en respectant les contraintes d'ordonnancement des tâches ainsi que les contraintes matérielles liées à l'architecture de la machine. En général, elle sont deux types : accès limité aux données et problèmes de synchronisation des processeurs [50]. Deux méthodes sont possibles pour la résolution d'un système d'équations linéaires Ax=b, où A est une matrice de taille nxn, x et b deux vecteurs de longueur n chacun:

(i) Décomposition de la matrice A : décomposition de Cholesky, décomposition LDM^t et décomposition LU qui est la méthode d'élimination de

Gauss.

(ii) Inversion de la matrice A.

La solution d'un système linéaire sur un ordinateur séquentiel nécessite $O(n^3)$ opérations arithmétiques. Une méthode parallèle pour inverser une matrice dense de taille nxn en $O(\log_2(n))$ opérations arithmétiques avec un nombre de processeurs supérieur à $n^4/2$, est développée par Csanky [13]; ce qui est excellent comme temps d'exécution mais l'algorithme présenté n'est pas assez stable. Sur les ordinateurs séquentiels la factorisation LU possède un de coût de $n^3/3 + O(n^2)$ opérations arithmétiques. Le parallélisme de cette dernière est étudié par plusieurs auteurs. Lord, Kowalik et Kumar [37] l'ont étudiée avec pivotage partiel sur un ordinateur MIMD en utilisant plus que $\lceil n/2 \rceil$ processeurs. Le même algorithme est étudié aussi par d'autres auteurs: Veldhorst [66], Marrakchi et Robert [41] et Robert et Trystram [48]. Cosnard et Coll. [8] ont discuté les six versions considérées en [15] en se servant du graphe d'ordonnancement. Ils ont présenté des algorithmes s'exécutant avec p processeurs $p=\alpha n$ où $\alpha \in [0,1]$.

2. PLAN DE LA THESE

Nous commençons par une introduction au parallélisme à l'aide des graphes des tâches (chapitre I). Ensuite nous étudions le parallélisme de la décomposition d'une matrice en LU (chapitres II et III). L'inversion d'une matrice s'effectue soit par l'algorithme de Rote [51] qui possède trois phases:

- factorisation LU
- inversion de L et U
- multiplication de U^{-1} par L^{-1}

soit par la résolution de n systèmes linéaires.
Les deux versions sont étudiées dans le chapitre IV.

Le premier chapitre comprend l'étude de l'architecture d'une machine à mémoire partagée, du graphe des tâches et quelques exemples de calcul parallèle. Nous définissons ensuite un modèle théorique afin de simplifier le calcul des complexités des algorithmes définis.

Dans le chapitre II, nous analysons les performances de plusieurs algorithmes parallèles de triangularisation sur un ordinateur MIMD. Nous montrons que les algorithmes SAXPY, GAXPY et DOT de [15], ainsi que les versions parallèles des décompositions LDM^t, LDL^t, de Doolittle et de Cholesky peuvent être classifiées à l'aide de six types de graphes. Nous présentons des résultats de complexité et comparons les performances asymptotiques de ces algorithmes parallèles.

Au début du chapitre III, nous présentons et nous analysons le graphe 2-pas dans le cas général. Pour un problème de taille n, en utilisant $p = \alpha n$ processeurs,

nous cherchons la valeur minimale α_0 de α pour laquelle l'exécution du graphe 2-pas peut s'effectuer en temps T_{opt}. Nous proposons ensuite deux types d'algorithmes pour la décomposition LU. Le premier algorithme est facile à programmer. Des résultats de simulation pour ce dernier sont présentés. Le second algorithme est asymptotiquement optimal pour toute valeur de α. Pour $\alpha \leq \alpha_0 \simeq 0.347$, où α_0 est solution de l'équation $3\alpha-\alpha^3=1$, les algorithmes sont d'efficacité $e_\alpha = 1/(1+ \alpha^3) \geq 0.959$, et pour $\alpha \geq \alpha_0$, les algorithmes sont d'efficacité $e_\alpha = 1/(3\alpha)$. En particulier, nous atteignons la borne inférieure $p = \alpha_0 n$ établit par Lord, Kowalik et Kumar [37] pour le nombre minimal de processeurs nécessaires pour une exécution parallèle en temps optimal $T_{opt} = n^2 + O(n)$.

Nous étudions dans le chapitre IV la parallélisation de l'inversion d'une matrice dense à l'aide de l'algorithme de Rote [51], sans duplication et avec duplication temporaire et complète . Nous présentons aussi les différentes versions parallèles de l'inversion d'une matrice à l'aide de la méthode de Gauss, de Jordan et de Huard. Enfin, nous comparons les résultats des différentes versions.

Tout au long de cette thèse, notre travail est axé sur la recherche d'algorithmes optimaux et de résultats de complexité.

Chapitre I

INTRODUCTION AU PARALLELISME A L'AIDE DU GRAPHE DES TACHES

1. INTRODUCTION

Pour le calcul parallèle, le nombre de processeurs n'est pas le seul paramètre. Les contraintes imposées par la structure de la machine peuvent influencer notablement les performances. De plus le mode d'accès aux données prend une importance considérable et le rapport entre le temps de transmission et le temps de traitement devient une variable primordiale. Ces temps sont étroitement liés à la conception et à la fabrication de la machine parallèle sur laquelle est exécuté l'algorithme [50].

Dans ce chapitre, nous présentons d'une façon générale l'architecture d'un calculateur parallèle. Nous classifions les différentes architectures (classification de Flynn [21]) et nous donnons les définitions des systèmes à mémoire partagée et distribuée en précisant la différence entre les deux systèmes. Dans la section 5, nous citons quelques remarques sur les réseaux d'interconnexion qui possèdent un rôle très important dans l'échange des messages entre les processeurs et nous présentons ensuite quelques résultats sur les coûts des échanges de données entre processeurs et mémoire dans le cas où cette dernière est partagée [53]. La notion de système de tâches, telle que l'utilise Kumar [32] permet d'introduire un formalisme qui sous-tendra toute nos études de parallélisation d'algorithmes. La section 6 décrit cet outil principal. Enfin, nous présentons un modèle théorique d'ordinateur parallèle pour que nous puissions calculer les complexités des algorithmes parallèles définis.

2. CLASSIFICATION DES ARCHITECTUTRES

Nous présentons dans ce paragraphe la classification des architctures la plus utilisée: c'est celle de Flynn [21, 61]. Celle-ci a pour critère de sélection le mode de contrôle des suites d'opérations élémentaires effectuées par les différents processeurs. D'autres classifications ont été proposées : elles sont basées sur les flux d'instructions et d'exécutions Kuck [30], sur différents modèles de calcul et leur implémentation [64] ou sur des caractéristiques architecturales particulières Schwarz [58].

Un flux d'instructions (FI) est une suite d'instructions issues d'une partie de contrôle en direction d'un ou plusieurs processeurs. Un flux de données (FD) est une suite de données venant de la mémoire partagée (MP) vers l'unité de traitement (UT) ou inversement. L'unité de contrôle (UC) est un véritable chef d'orchestre de l'unité de traitement. La mémoire est partagée en modules mémoire (MM) pour accélérer la communication processeur-mémoire.

Pour l'obtention d'un calculateur parallèle, Flynn a donc observé la nécessité de la multiplicité des flux de données et des flux d'instructions. Ce qui donne naturellement les quatre catégories de calculateurs. Un seul flux d'instructions et de données (SISD) : c'est l'ordinateur classique de Von Neumann qui exécute les instructions séquentiellement. Ces dernières peuvent être pipe-linées. C'est le cas

de la plupart des machines d'aujourd'hui relevant de cette catégorie. Cette structure est rappelée à la figure 1.

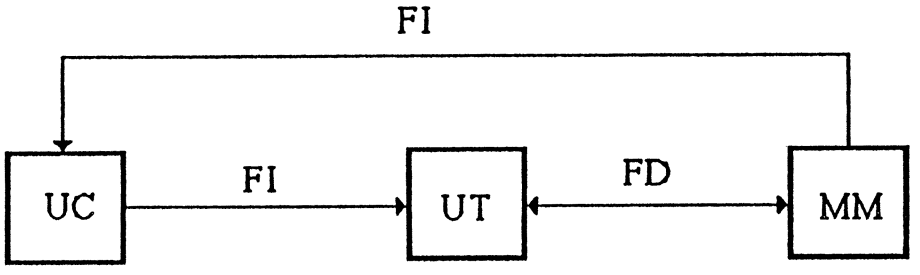


Figure 1 : Structure SISD.

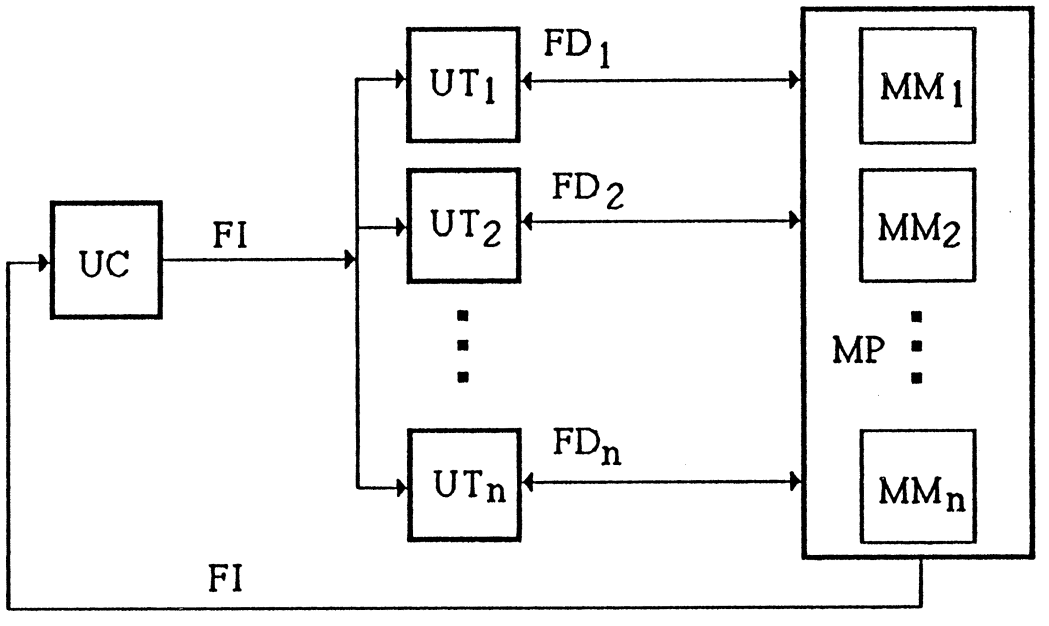


Figure 2 : Structure SIMD.

Le calculateur SIMD, un seul flux d'instructions, plusieurs flux de données, est obtenu en multipliant le flux de données d'un calculateur SISD; ce que réalise le calculateur de la figure 2. Une instruction sur un calculateur SIMD opère sur plusieurs données plutôt que d'opérer sur une seule donnée. L'unité de contrôle est unique et réalise la synchronisation des processeurs.

Comme on a multiplié les flux de données, il est logique de multiplier le flux des instructions. Ce qui réalise le calculateur: plusieurs flux d'instructions, un seul flux de données (MISD) (voir figure 3). Malheureusement, ce type de calculateurs n'est pas réaliste en calcul parallèle: il semble qu'il n'y ait guère d'applications possibles.

En combinant la multiplicité des deux flux d'instructions et de données, on obtient le calculateur MIMD: plusieurs flux d'instructions, plusieurs flux de données. Cet ordinateur est composé de n calculateurs SISD (voir figure 4). La différence entre cette dernière structure et la structure SIMD réside dans le fait

que la première structure possède plusieurs unités de contrôle tandis que la seconde a une seule unité de contrôle; ce qui établit respectivement le caractère synchrone ou asynchrone de l'architecture.

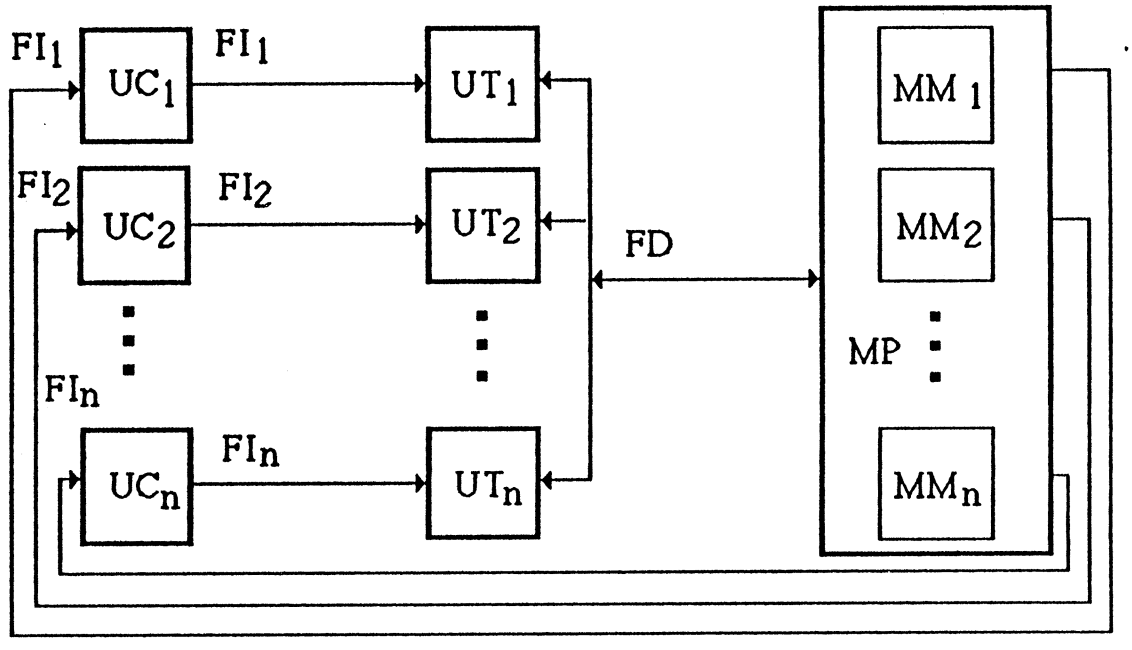


Figure 3 : Structure MISD.

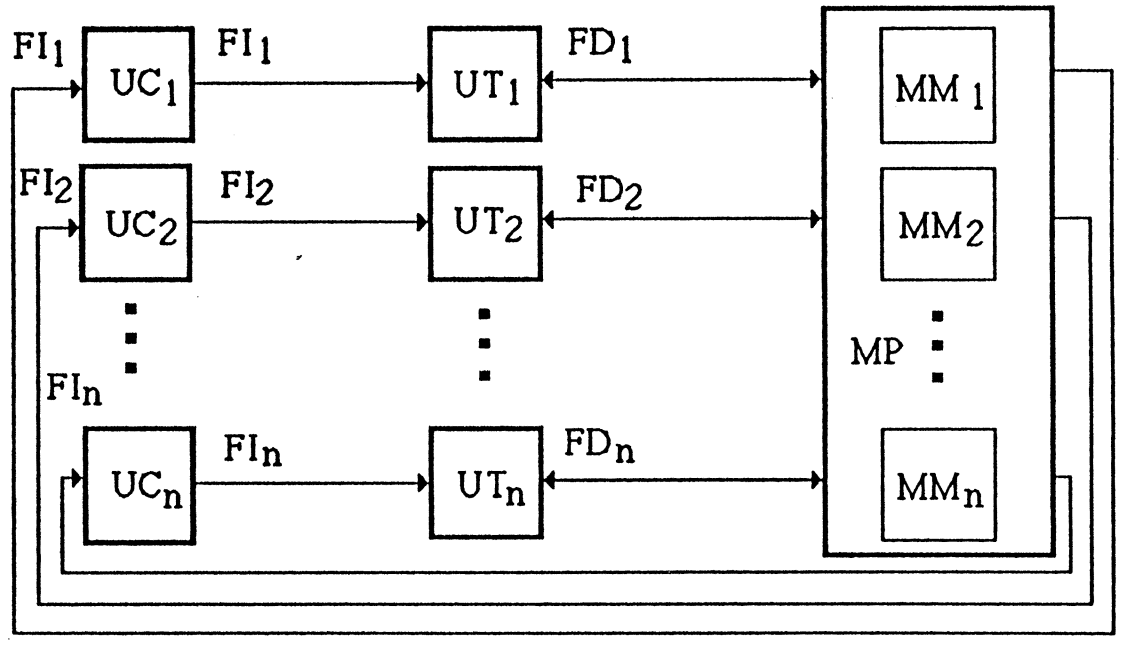


Figure 4 : Structure MIMD.

3. STRUCTURES SIMD & MIMD

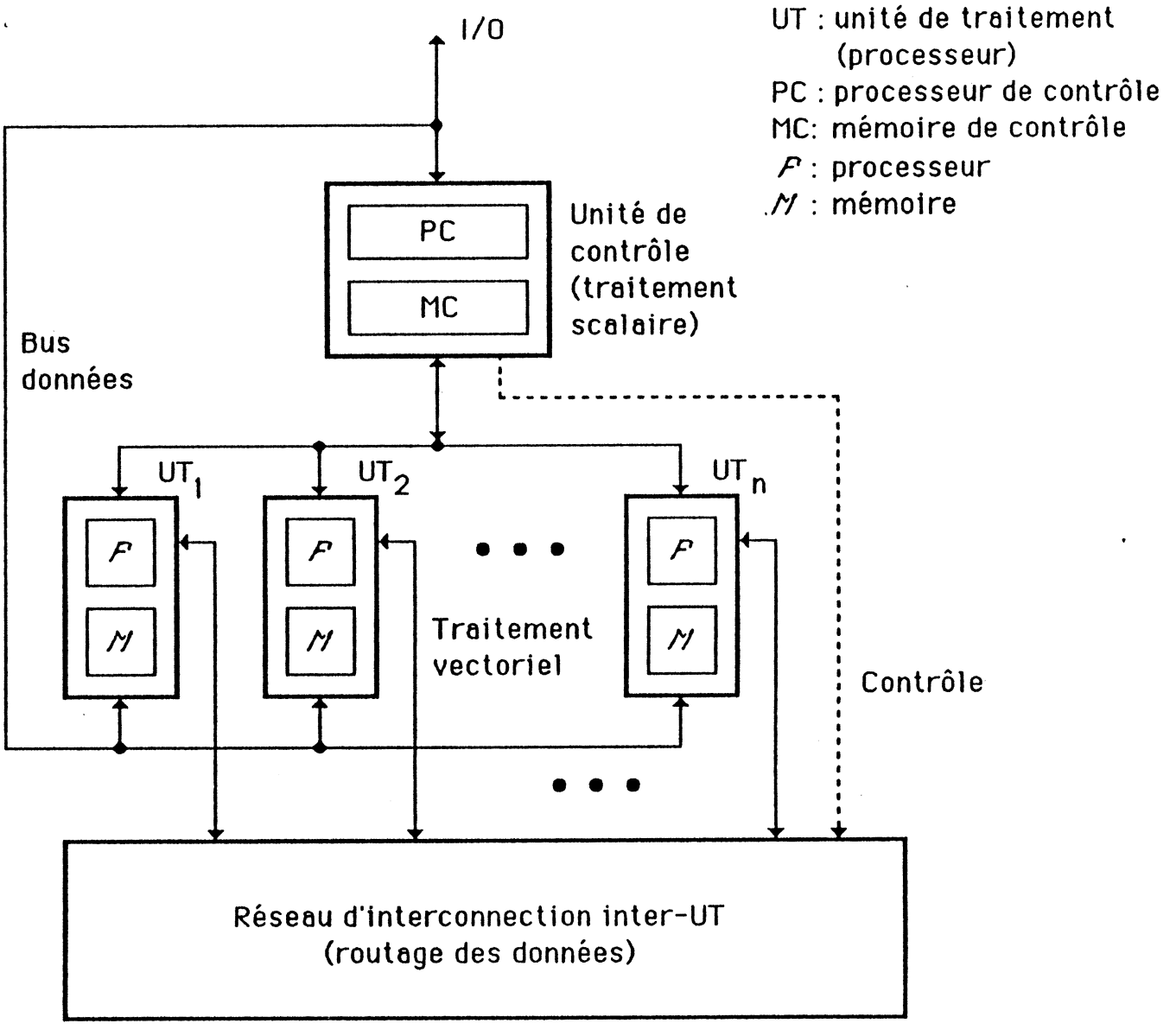
Dans ce paragraphe, nous illustrons les formes générales des calculateurs SIMD et MIMD et leurs modes de fonctionnement dans le cas où la mémoire est partagée.

STRUCTURE SIMD :

Ce type de calculateur est constitué de n processeurs liés aux m bancs mémoire indépendants par un réseau d'interconnexion facilitant la communication des données (voir figure 5). L'unité de contrôle est un véritable chef d'orchestre du calculateur donnant aux processeurs le signal de l'exécution de chaque instruction et ordonnant l'exécution des instructions. Les unités de traitement exécutent donc la même instruction au même instant. On obtient un fonctionnement synchrone des processeurs. Comme exemples d'ordinateurs SIMD Illiacs IV, BSP, STARAN, MPP, DAP. Par contre, OPSILA, conçu par AUGUIN, BOERI et Coll. [4] est une machine SIMD/SPMD.

STRUCTURE MIMD :

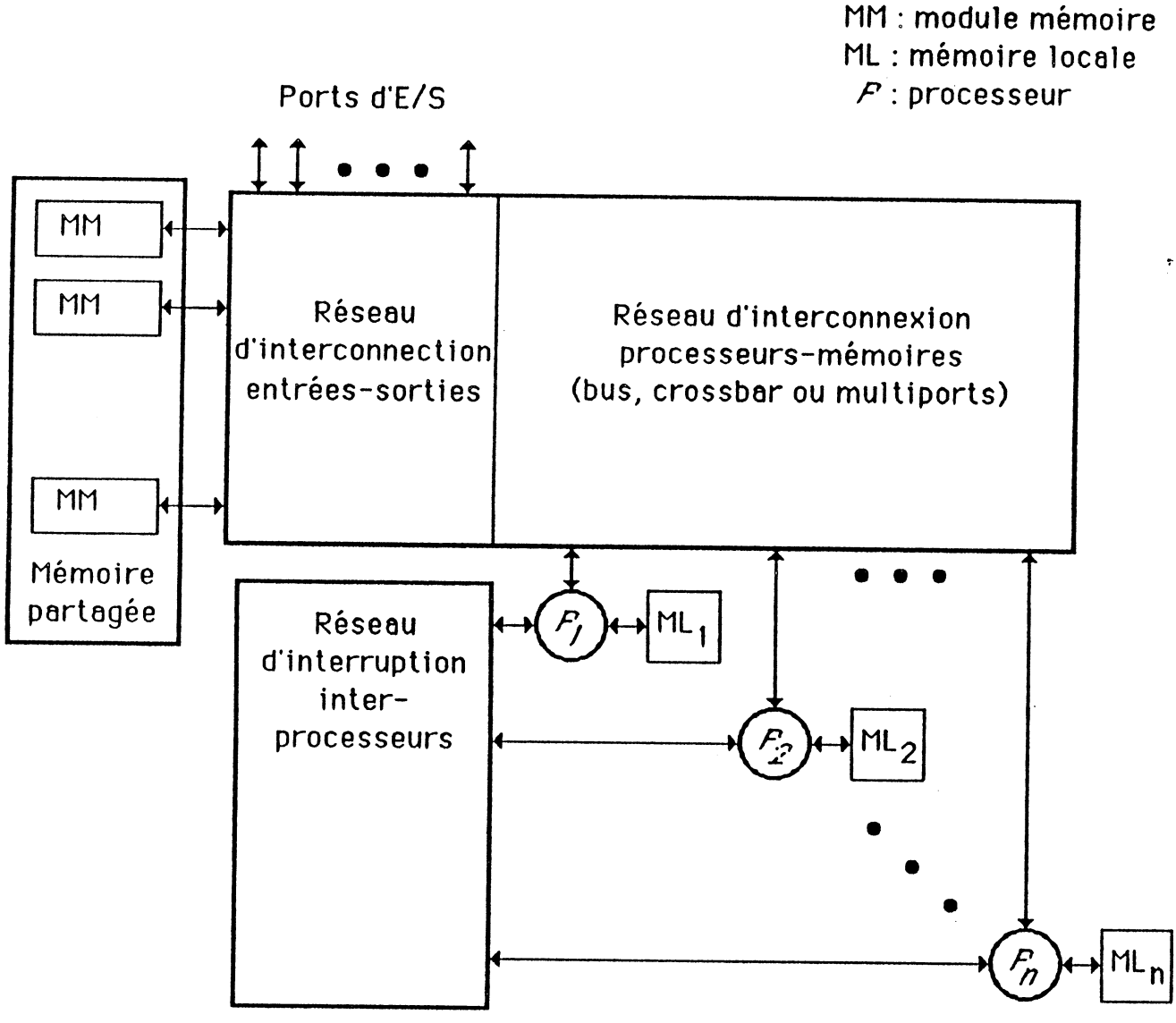
Le calculateur MIMD est plus flexible que le calculateur SIMD. Il est adapté à une plus large classe d'algorithmes. Dans un ordinateur MIMD, chaque processeur possède sa propre unité de contrôle. Les processeurs ont donc un fonctionnement indépendant (en particulier asynchrone) et exécutent des programmes différents (voir figure 6). Une structure MIMD "intrinsèque" implique des interactions entre les n processeurs (car tous les flux de mémoires sont issues d'une même unité allouée aux données). Si les n flux de données étaient issus de sous-unité disjointes de la mémoire partagée, nous aurions une structure dite "multiple-SISD", qui n'est rien d'autre que la juxtaposition de n systèmes uniprocesseurs SISD indépendants. Un ordinateur MIMD est fortement couplé si les interactions entre les processeurs sont importantes. La plupart des architectures MIMD commerciales sont faiblement couplées (IBM 370/168MP, Univac 1100/80, IBM 3081/3084, Cm*, ...). Parmi les MIMD fortement couplés, citons Cray-2, Cray X-MP, C.mmpp, ALLIANT FX/8 et IBM 3090.



Structure fonctionnelle d'un ordinateur vectoriel SIMD
avec traitement scalaire simultané dans l'unité de contrôle

d'après Hwang & Briggs, Computer Architecture & Parallel Processing,
Mac Graw Hill 1984

Figure 5



Structure fonctionnelle d'un système multiprocesseur MIMD

d'après Hwang & Briggs, Computer Architecture & Parallel Processing, Mac Graw Hill 1984

Figure 6

4. MULTIPROCESSEURS

Les multiprocesseurs sont de deux types [5] :

- système à mémoire partagée : Denelcor HEP, Alliant, ...
- système à mémoire distribuée : Hypercube (iPSC, T20) ...

Le système à mémoire partagée est un système permettant à tous les processeurs (unités de traitement arithmétiques) de communiquer avec la mémoire. Cette dernière est partagée en bancs pour que les processeurs puissent communiquer simultanément avec elle. Les données sont rangées dans la mémoire partagée. Cette mémoire doit fournir les opérandes et recevoir les résultats de tous les processeurs. La communication entre les processeurs et la mémoire s'effectue par un réseau d'interconnexion. Chaque processeur peut avoir une mémoire locale pour le stockage temporaire. Elle n'est pas assez grande pour tenir toutes les données du problème à résoudre. La figure 7 présente l'architecture de ce système. Dans le second système, c'est à dire un multiprocesseur à mémoire distribuée, chaque processeur possède une mémoire propre. Un processeur ne peut pas accéder directement à une mémoire autre que la sienne (voir figure 8). La communication entre les processeurs s'effectue aussi à l'aide d'un réseau d'interconnexion qui peut avoir plusieurs topologies comme le tore, l'arbre, le cube sur l'hypercube. La différence profonde entre les deux systèmes réside dans la liaison entre les processeurs et la mémoire; ce qui établit des communications directes ou indirectes entre les processeurs et la mémoire suivant que le calculateur est à mémoire partagée ou distribuée.

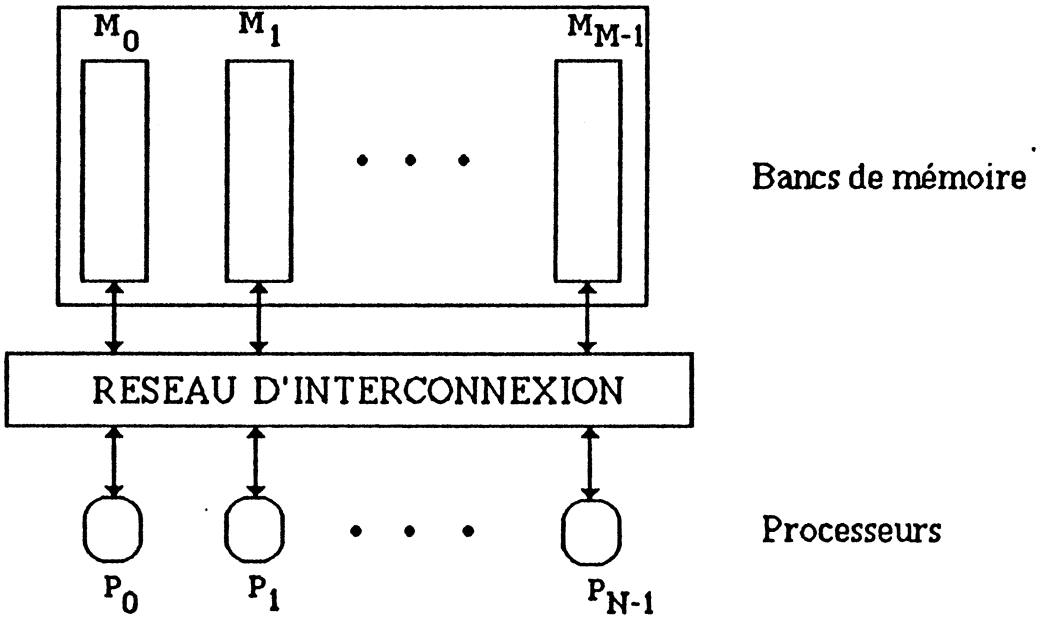


Figure 7: Architecture d'un système à mémoire partagée.

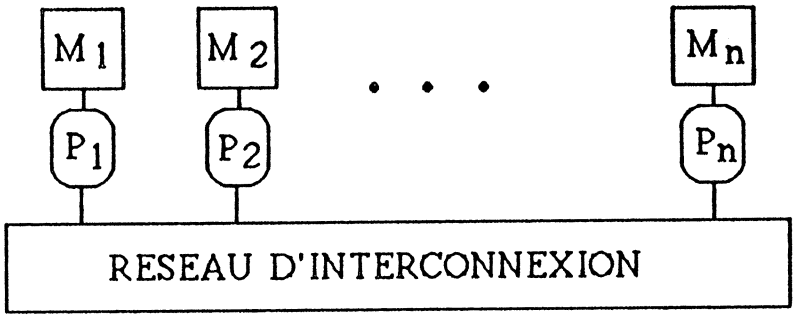


Figure 8 : Architecture d'un système à mémoire distribuée.

5. COMMUNICATION

En calcul parallèle, il est nécessaire d'échanger des données entre les processeurs. Cet échange est un obstacle principal pour améliorer les performances des algorithmes parallèles. Plusieurs algorithmes possèdent un coût de communication intensif par rapport au coût d'opérations arithmétiques élémentaires. Notre but dans ce paragraphe est d'analyser le coût de communication pour une machine à mémoire partagée. Cette dernière comporte n processeurs liés à une mémoire partagée par un réseau d'interconnexion (figure 7). Il possède un rôle très important dans l'échange des données. L'architecture et le mode de fonctionnement des principaux réseaux d'interconnexion sont présentés dans [4, 35]. Un bon réseau d'interconnexion doit à la fois minimiser le nombre total de connexions physiques et la distance entre les entrées et les sorties évaluée en nombre de connexions tout en passant le nombre maximum de messages simultanément [46]. Dans ce qui suit, on présente quelques remarques sur les réseaux d'interconnexion puis sur le coût de communication de certaines opérations de transfert dans un système à mémoire partagée.

5.1. RESEAUX D'INTERCONNEXION

Interconnecter des unités qui échangent des informations n'est pas un problème nouveau. Ce problème est devenu primordial depuis l'avènement des composants VLSI et des besions en grandes puissances de calculs qui nécessitent la prise en compte du parallélisme dans les applications [4]. Dans ce qui suit, afin d'avoir un très haut niveau de parallélisation des communications, nous présentons quelques remarques sur les réseaux d'interconnexion pour un système à mémoire partagée.

1- Pour un réseau avec blocage, la communication ne possède pas un très haut niveau de parallélisation.

2- La lecture simultanée d'une même donnée par deux processeurs ou plus n'est pas possible. Une entrée n'est en communication avec une sortie que si cette dernière est au repos.

3- Pour la même raison, l'écriture simultanée par deux processeurs ou plus sur un même banc mémoire n'est pas possible non plus . On a intérêt

à éviter l'écriture simultanée de plusieurs opérandes sur un même banc mémoire.

4- Si le nombre de bancs mémoire est strictement inférieur au nombre de processeurs, on risque d'avoir quelques uns de ces derniers bloqués. Pour améliorer le taux d'activité des processeurs, il est nécessaire que le nombre de bancs mémoire soit supérieur au nombre de processeurs.

5- Afin d'avoir un très haut niveau de parallélisme pour la communication, il faut que les données se répartissent convenablement sur les bancs mémoire. Il faut éviter la requète simultanée de deux données appartenant à un même banc mémoire.

6- Le nombre de points de connexion augmente avec le nombre de processeurs. Dès que ce dernier augmente, le temps moyen de communication augmente. Ce qui justifie le résultat de Saad [54] disant que les temps de communication prédominent pour résoudre un problème matriciel de taille n avec n^2 processeurs.

5.2. COUT DE COMMUNICATION

Programmer en parallèle soulève le probème de communication entre les processeurs. Cependant, l'échange de message possède certaines difficultés, comme le problème de conflit qui peut agir sur les complexités des algorithmes parallèles. D'après l'étude faite par Saad [53] sur la communication des processeurs dans une architecture parallèle, on déduit les résultats suivants pour un système à mémoire partagée. Pour échanger un paquet contenant m mots entre un processeur et la mémoire partagée, le temps nécessaire est égal à

$$\tau + m/b$$

où τ est le temps d'initialisation (start-up) et b est la largeur de bande (bandwidth) de chaque processeur. Les processeurs peuvent lire et écrire simultanément. La largeur de la bande de la mémoire ne peut accepter qu'un multiple n* de la largeur de bande b avec $n^* \leq n$ (n étant le nombre de processeurs). C'est à dire, dans le cas le plus favorable où il n'existe aucun problème de conflit mémoire, différents processeurs peuvent accéder à la mémoire avec la même vitesse b. Pour déplacer N éléments d'un processeur P_i au processeur P_j ($i \neq j$), il est nécessaire de les écrire sur la mémoire globale puis de les lire par le processeur P_j. Le temps total mis est égal à

$$2(\tau + N/b).$$

La même démarche décrit ci-dessus est utilisée pour transmettre un paquet de données d'un processeur P_i à tous les autres processeurs. C'est à dire le processeur P_i écrit sur la mémoire les données, le temps mis pour effectuer cette opération est égal à $\tau + N/b$, les autres processeurs lisent ce paquet de données de la mémoire. Si cette dernière opération se fait à la fois par tous les processeurs,

on peut avoir des problèmes de conflit mémoire. Pour que ces problèmes ne se produisent pas, il faut que les données se lisent de la mémoire par bloc de n* processeurs. Le temps mis pour lire les données est alors $(\tau + N/b)\ n/n^*$. Supposons que les données sont propagées convenablement dans tous les différents bancs mémoire et il n'existe aucun conflit entre les bancs, le temps mis pour déplacer une donnée d'un processeur aux autres est:

$$(\tau + N/b)(1 + n/n^*).$$

Dans l'étude que nous faisons, un exemple très fréquent donnée par l'algorithme de Gauss et met en évidence ce type de communication: c'est le transfert des sorties de la tâche diagonale (préparation du pivot). Un processeur calcule le pivot dont tous les processeurs ont besoin pour modifier la partie correspondante de la matrice.

Le cas le plus général dans l'échange de données par les processeurs est le suivant: chaque processeur transmet une donnée à tous les autres processeurs et reçoit une donnée de chacun des autres processeurs. On suppose que chaque processeur envoie N/n mots (on peut supposer qu'ils envoient des données de longueurs différentes, on est obligé dans ce cas, pour simplifier les calculs, de prendre la donnée de longueur maximale). Le temps mis par les processeurs pour écrire sur la mémoire est $(\tau + N/(nb))n/n^*$, tandis que le temps de la lecture de N mots par chacun des processeurs de la mémoire est $(\tau + N/b)\ n/n^*$. Par conséquent, le temps total est

$$(2\tau + N(n+1)/(nb))\ n/n^*.$$

Si N/n=m indépendant du nombre de processeurs, on remarque que le temps mis pour échanger de messages croit avec la longueur de la donnée à transmettre et croit aussi avec le nombre de processeurs si n/n* est constant.

6. GRAPHE DES TACHES

La notion de systèmes de tâches, telle que l'utilisent Kumar [32] et Coffman et Denning [6] permet d'introduire un formalisme qui sous-tendra toute nos études de parallélisation d'algorithmes. Ces notions sont aussi développées par Cosnard et Robert [11]. Un algorithme peut être segmenté en un ensemble de tâches. L'étude des dépendances entre les tâches, liées à leurs variables d'entrées et de sorties, permet de construire un système de précédence qui traduit le parallélisme interne à l'algorithme. Dans un tel système, il peut exister des contraintes de précédence entre tâches et dans ce cas leur exécution est séquentielle. Dans le cas contraire, les tâches sont non-interférentes et leur exécution peut être effectuée en parallèle. Dans un premier temps, nous allons préciser toutes ces notions intuitives.

Une **tâche** est une unité indivisible de traitement caractérisée uniquement par son comportement extérieur: entrées, sorties, fonction et temps d'exécution. Par conséquent, la tâche T sera représentée par un quadruplet $(E_T,S_T,f_T,Ex(T))$ tel que:

- E_T est l'ensemble des entrées de T,
- S_T est l'ensemble des sorties de T,
- f_T est un opérateur de E_T dans S_T, c'est à dire une suite ordonnée d'opérations élémentaires de E_T dans S_T.
- Ex(T) est un entier représentant le temps d'exécution de la tâche T.

Le **temps** (ou durée) **d'exécution** d'une tâche est en général la somme de deux termes: le **temps de calcul**, qui correspond au nombre d'opérations élémentaires effectuées dans la tâche, et le **temps de communication**, lié au nombre de transferts entre la mémoire partagée et le processeur exécutant la tâche. Dans les exemples que nous étudions la longueur d'une tâche est de l'ordre de O(n) pour que le temps de calcul prédomine, n la taille du problème matriciel à résoudre.

Le coût d'une opération arithmétique sera évaluée à une unité de temps (on admet qu'une multiplication équivaut à une addition).Les problèmes de disponibilité et d'intégrité des données ne sont pas pris en compte à ce niveau puisqu'ils dépendent de l'ordonnancement des tâches en vue de leur exécution. Par contre, les problèmes de conflit d'accès mémoire, comme par exemple lors d'un conflit de banc mémoire, sont difficiles à modéliser, puisqu'ils dépendent de l'architecture, des mécanismes de stockage, des mécanismes de transferts et bien entendu de l'exécution de l'algorithme. Nous supposerons dans la suite que le temps de résolution de ces problèmes est inclus dans le temps de communication. Enfin, une tâche est indivisible: dès qu'elle sera affectée à un processeur, d'une part, celui-ci ne pourra interrompre son traitement pour exécuter une autre tâche, d'autre part, au cours de l'exécution, aucun autre processeur ne peut ni écrire ni lire une donnée subissant une modification.

Un **algorithme séquentiel** est un ensemble de tâches, muni d'une relation d'ordre total: $A = (T_1,\ldots,T_n,\rightarrow)$. La relation $T_i \rightarrow T_k$ $(i\neq k)$ signifie que l'exécution de la tâche T_i doit être terminée avant que ne débute l'exécution de T_k. Un algorithme séquentiel peut être considéré lui aussi comme une tâche unique: $(E_A,S_A,f_A,Ex(A))$.

D'après la définition précédente, une tâche est un algorithme séquentiel et réciproquement: un algorithme séquentiel est une tâche dans le cas où nous connaissons son temps d'exécution. C'est le cas des principaux algorithmes d'algèbre linéaire.

Un algorithme séquentiel peut avoir plusieurs décompositions différentes. Le choix de la meilleure décomposition est un problème difficile. Il est lié aux temps

d'exécution des tâches et à l'architecture sous jacente. Les principaux facteurs de choix sont le nombre de processeurs, le rapport entre unité de temps de communication et unité de temps de calcul, les accès mémoire. Mais en négligeant le coût de communication, il est possible de comparer les différentes décompositions d'un même algorithme séquentiel. Remarquons que l'utilisateur a alors besoin d'utiliser un langage de programmation qui permette de définir des tâches et les communications entre elles, par exemple OCCAM [67] ou Lestap [22].

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

Soit $S = (T_1,\ldots,T_n,<<)$ un système de tâches. Deux tâches T_i et T_k sont **indépendantes** si elles ne modifient aucune variable commune. La relation d'indépendance s'exprime ainsi:

$$(S_{T_i} \cap S_{T_k}) = (S_{T_i} \cap E_{T_k}) = (E_{T_i} \cap S_{T_k}) = \emptyset$$

c'est à dire aucune sortie d'une tâche n'est ni une entrée ni une sortie de l'autre, ce qui permet leur exécution simultanée.
T_i et T_k sont **consécutives** si $T_i << T_k$ et s'il n'existe aucune autre tâche T_j telle que $T_i << T_j << T_k$. $T_i << T_k$ signifie qu'il existe au moins une sortie de T_i qui est une entrée ou une sortie de T_k et l'exécution T_k ne débute qu'après la fin de celle de T_i.

Un système de tâches $S = (T_1,\ldots,T_n,<<)$ est un **système de précédence** si, pout tout i et k $(i\neq k)$, une et une seule des trois 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** (ou graphe des tâches) G associé à un **système de précédence** $S = (T_1,\ldots,T_n,<<)$ est défini de la manière suivante:

- l'ensemble des sommets de G est l'ensemble des tâches de S
- T_i et T_k sont reliées par une arête si et seulement si T_i et T_k sont consécutives.

Etant donné un algorithme séquentiel $A = (T_1,\ldots,T_n,\rightarrow)$, on peut montrer qu'il existe **un unique système de précédence** $S = (T_1,\ldots,T_n,<<)$ tel que << **soit un sous ordre de**→ .<< est la relation suivante:

$$T_i << T_k \Leftrightarrow (T_i \rightarrow T_k \text{ et } (S_{T_i} \cap S_{T_k}) \cup (S_{T_i} \cap E_{T_k}) \cup (E_{T_i} \cap S_{T_k}) \neq \emptyset)$$

Pour paralléliser un algorithme séquentiel, on commencera donc toujours par calculer la relation << à partir de la relation →. Dans la suite, nous supposerons cette étape effectuée. Là encore, elle peut être soit à la charge du programmeur, soit faite par programme.

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

$$Ord: \{T_1,\ldots,T_n\} \longrightarrow \{1,\ldots,p\}xN$$

$$T_k \longrightarrow Ord(T_k) = (prc(T_k),tps(T_k)),$$ telle que:

(i) $T_i << T_k$ implique $tps(T_i) + Ex(T_i) \leq tps(T_k)$.

(ii) $prc(T_i) = prc(T_k)$ implique ($tps(T_i) + Ex(T_i) \leq tps(T_k)$ ou $tps(T_k) + Ex(T_k) \leq tps(T_i)$).

$prc(T_k)$ est le processeur qui exécutera T_k et $tps(T_k)$ est le date du début de l'exécution de T_k. La condition (i) signifie simplement 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. La condition (ii) 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'exécution de l'algorithme parallèle est la date de fin d'exécution de la dernière tâche exécutée. C'est donc:

$$T_{PA} = \max_{k=1}^{n} (tps(T_k) + Ex(T_k))$$

Remarquons que si le nombre de processeurs p=1, la définition d'un algorithme parallèle coincide avec celle d'un algorithme séquentiel. Pour cette raison, nous supposons que le nombre de processeurs est toujours supérieur où égal à deux. Même si nous disposons plus d'un processeur, il n'est pas possible d'avoir un algorithme parallèle dans le cas où un algorithme séquentiel ne possède aucune décomposition en tâches indépendantes. On dit que l'algorithme est complètement séquentiel.

Un algorithme parallèle associé à un algorithme séquentiel A est **optimal** s'il n'existe aucun algorithme parallèle associé à A ayant un temps d'exécution inférieur. Pour simplifier, nous essayons de construire des algorithmes asymptotiquement optimaux. Un algorithme parallèle est **asymptotiquement optimal** pour p processeurs si le rapport entre son temps d'exécution T_p et T_{opt} tend vers 1 lorsque n (taille du problème) tend vers l'infini. Autrement dit, la différence entre les deux temps est négligeable.

Soit A un algorithme séquentiel, quelques problèmes sont étudiés dans le cadre de la parallélisation d'algorithmes. Ils peuvent être formulés ainsi:

(1) en ne supposant pas de limite au nombre de processeurs disponibles, quel est le temps T_{opt} d'un algorithme parallèle optimal ? Si $T_{opt}(p)$ est le temps d'exécution d'un algorithme parallèle optimal avec p processeurs, T_{opt} est défini par:

$$T_{opt} = \min_{p=1}^{n} (T_{opt}(p))$$

(2) quel est le nombre mimimal de processeurs p_{opt} permettant de réaliser un algorithme s'exécutant en temps T_{opt} ? p_{opt} est défini par:

$$p_{opt} = \min_{p=1}^{n} (p \text{ tel que } T_{opt}(p)=T_{opt})$$

(3) étant donné p processeurs, quel est la valeur optimale de l'efficacité $e_{opt}(p)$?

(4) existe -t- il des algorithmes parallèles s'exécutant avec p processeurs et possédant l'efficacité $e_{opt}(p)$?

(5) quel est le nombre de processeurs maximisant l'efficacité?

Pour une étude détaillée de l'ordonnancement d'un système de précédence, nous renvoyons le lecteur aux ouvrages de Coffman, et Denning [6] et Coffman [7]. Ce problème n'est entièrement résolu que dans deux cas: lorsque le nombre de processeurs est égal à 2 et que toutes les tâches ont le même temps d'exécution et lorsque le graphe des tâches est un arbre et que toutes les tâches ont le même temps d'exécution. Même dans le cas d'une relation de précédence vide (toutes les tâches sont indépendantes) on ne connait pas d'algorithme optimal.

Rappelons que le temps d'exécution d'un chemin du graphe des tâches est la somme des temps d'exécution des tâches qui le composent. Le plus long chemin du graphe des tâches est celui dont le temps d'exécution est le plus grand. La **hauteur** H(G) du graphe des tâches est le nombre de tâches du plus long chemin.

Les **antécédents** d'une tâche T_k sont les T_i telles que$T_i<<T_k$. T_i est un **prédécesseur** de T_k et T_k est un **successeur** de T_i si T_i et T_k sont consécutives.

Nous appelerons **décomposition en niveaux** du graphe des tâches D(G) = $\{N_1,\ldots,N_{H(G)}\}$ une partition en H(G) sous ensembles (niveaux) des sommets de ce graphe vérifiant les conditions suivantes. Le niveau 1 est constitué de tâches n'ayant aucun prédécesseur. Le niveau k est constitué de tâches dont tous les prédécesseurs sont dans des niveaux inférieurs et dont tous les successeurs sont dans des niveaux supérieurs. Le niveau H(G) est constitué de tâches n'ayant aucun successeur. Remarquons que, puisque H(G) est le nombre de tâches du plus long chemin, chaque niveau est non vide: il contient une tâche de chacun des plus longs chemins. Remarquons aussi qu'un même graphe admet plusieurs décompositions en niveaux.
Deux décompositions particulières sont la **décomposition par prédécesseurs**

où chaque niveau contient tous les prédécesseurs possibles et la **décomposition par successeurs.** La première décomposition est aussi appelée décomposition au plus tôt, et la seconde décomposition au plus tard, car un ordonnancement par niveaux conduit à une date de début d'exécution minimum dans le premier cas et maximum dans le second pour chacune des tâches.

Nous appelerons **largeur** d'une décomposition $D(G) = \{N_1,\ldots,N_{H(G)}\}$ le maximum des cardinaux des niveaux: $L(D) = \max\ (1\leq k\leq H(G), \text{cardinal}(N_k))$. Nous appelerons **largeur** L(G) du graphe des tâches le minimum des largeurs des décompositions de G:

$$L(G) = \min_{D(G)} L(D) = \min_{D(G)} \max_{k=1}^{H(G)} (\text{cardinal}(N_k))$$

Toutes ces définitions nous permettent d'obtenir le résultat suivant:
Soit $A = (T_1,\ldots,T_n,\rightarrow)$ un algorithme séquentiel dont toutes les tâches d'un même niveau N_k ont le même temps d'exécution t_k.
Le temps T_{opt} d'un algorithme parallèle optimal est égal au temps d'exécution du plus long chemin du graphe des tâches. Il est égal à

$$\sum_{k=1}^{H(G)} t_k.$$

Pour diverses raisons: (contenu d'une tâche, duplication de quelques données,...), on peut avoir différents graphes d'ordonnancement. Ce qui entraîne qu'un algorithme séquentiel peut conduire à plusieurs versions parallèles. Pour définir un algorithme parallèle, on est amené à suivre la démarche suivante:

1- Découper l'algorithme séquentiel en tâches
2- Définir la relation de précédence et construire le graphe de tâches
3- Affecter à un instant donné l'exécution d'une tâche donnée au processeur correspondant tout en respectant les contraintes d'ordonnancement.

La boucle B suivante :

```
B :  DO i=1, n
        Ti
     END.
```

peut s'exécuter en parallèle de plusieurs manières suivant le nombre de processeurs et les contraintes d'ordonnancement. Supposons qu'on dispose de p processeurs, les différentes façons d'exécution parallèle de cette boucle B se situent entre les deux manières extrêmes d'exécution parallèle suivantes :

•Boucle complètement parallèle

Le début de l'exécution des tâches est simultané, on a $tps(T_i)=tps(T_{i+1})$ pour tout i=1, 2, ..., n-1. Aucune contrainte d'ordonnancement n'existe entre les tâches. Ces dernières sont situées sur un même niveau. Ce qui autorise une exécution simultanée de toutes les tâches. La largeur de décomposition est L(D)=n et le plus long chemin est donné par la tâche la plus longue. Si le nombre de processeurs p est strictement inférieur à n, il existe au moins un processeur exécutant plus qu'une tâche. Dans le cas contraire, c'est à dire $p \geq n$, chaque processeur exécute au plus une seule tâche et le temps mis est égal à T_{opt}. Dans le premier cas où $p<n$ et si les tâches ont la même longueur, il n'est pas possible de trouver un algorithme parallèle s'exécutant en temps T_{opt}.

•Boucle complètement séquentielle

Le début de l'exécution d'une tâche n'aura lieu que si l'exécution d'une autre se termine. Autrement dit, l'exécution d'une tâche quelconque T_i ($i \neq 1$) dépend obligatoirement au moins d'une autre tâche. Supposons que cette dernière est la tâche T_{i-1}, par conséquent nous sommes dans le cas où $tps(T_{i+1})=tps(T_i)+Ex(T_i)$ pour tout i=1, 2, ..., n-1, les tâches s'exécutent séquentiellement. Le graphe d'ordonnancement possède n niveaux, chaque tâche constitue un niveau. La largeur de décomposition L(D) est égale à 1. Le plus long chemin est formé par toutes les tâches, sa longueur est le temps T_{opt}. Dans ce cas, il n'existe aucun algorithme parallèle.

7. UN MODELE THEORIQUE

Les caractéristiques précises de l'architecture sur laquelle on programme l'algorithme sont sans importance. Nous considérons simplement un système capable de supporter plusieurs flux d'instructions s'exécutant indépendamment et en parallèle sur plusieurs flux de données [23], [26], [57]. Nous supposons qu'il existe des mécanismes permettant de synchroniser le processus de résolution, i.e. assurant le respect des contraintes de précédence temporelles qui sont imposées par la nature des algorithmes implémentés sur l'architecture. Le coût des mécanismes de synchronisation sera négligé devant le coût des opérations arithmétiques. De même, le temps d'accès en mémoire centrale pour lire ou stocker une donnée sera considéré comme nul. Pour rester réalistes, ces hypothèses appellent les restrictions suivantes:
- la communication entre processeurs se fait par l'intermédiaire d'une mémoire partagée plutôt que par un bus local
- pour un problème de taille n, le nombre de processeurs p est limité à O(n). Saad [54] montre que les temps de communications prédominent pour résoudre un problème matriciel de taille n avec n^2 processeurs. Plus précisément, nous poserons $p = \alpha n$, avec $\alpha<1$, ce qui facilite aussi l'évaluation des performances.

Pour évaluer les performances d'un algorithme parallèle, on négligera donc les temps d'accès en mémoire centrale, de stockage et d'échange des données, pour ne retenir que le coût des opérations arithmétiques.

Si nous négligeons le temps de communication, nous imposons un certain mode d'accès aux données. Pour rester proche d'un environnement de programmation FORTRAN, nous supposons que nous pouvons accéder aux éléments d'une matrice A par colonnes [35, 57]. Les deux opérations de transfert possibles seront alors la transmission d'une colonne de A de l'organe de stockage vers un organe de traitement et l'opération inverse. La duplication d'une colonne aura un coût nul, c'est à dire que nous supposons qu'il est possible de transférer simultanément une même donnée vers plusieurs processeurs. Dans ce cas, aucun processeur ne pourra modifier cette donnée. Inversement, un processeur ne pourra modifier une donnée que s'il est le seul à en posséder un exemplaire. Nous ne nous préoccupons pas des mécanismes informatiques qui permettent de résoudre ce problème d'allocation des données sans conflit. Remarquons simplement que les contraintes précédentes et le mécanisme qui les gère permettent d'assurer un déroulement sans ambiguité de l'algorithme et une certaine synchronisation.

En se servant des processeurs vectoriels, le coût d'une opération arithmétique s'évalue comme suit: supposons que l'additionneur (respectivement le multiplieur) possédant e_+ (respectivement e_x) étages. Le temps nécessaire pour effectuer une instruction arithmétique sur deux vecteurs de longueurs n chacun est:

$$(e_i + (n-1))t$$

où $i \in \{+, x\}$ et t est le temps de traversée d'une étage. Si l'enchaînement des opérations arithmétiques est possible, le temps total par exemple pour effectuer une addition suivie d'une multiplication devient:

$$(e_+ + e_x + (n-1))t.$$

Dans la suite, nous tenons compte seulement du coût arithmétique: nous choisissons comme unité de temps une des quatre opérations arithmétiques. Mais, conformément à l'usage en matière d'analyse de complexité pour l'élimination de Gauss avec pivotage partiel [37, 56, 68], nous supposons pour cet algorithme que chaque processeur peut réaliser en une unité de temps une multiplication suivie d'une soustraction, ou une comparaison suivie d'une multiplication, ou encore une division.

La négligence du temps de communication permet de simplifier les calculs des perfomances des algorithmes parallèles construits. Pour avoir des résultats plus proche de la réalité il suffit de tenir compte du temps de communication dans l'évaluation des performances. Le modèle s'étend aux coûts de communication sur une mémoire partagée. Le coût d'une tâche, possédant q opérandes, peut être évalué comme la somme du coût arithmétique:

$$\tau_a = aq + b$$

et du coût de communication:

$$\tau_c = \alpha q + \beta$$

Les paramètres a, b, α et β sont en fonction des caractéristiques de la machine. Le temps d'un algorithme parallèle devient alors la somme des coûts des opérations arithmétiques et de communication. Il est possible de pipe-liner les instructions arithmétiques, de les enchaîner entre elles et avec les transferts mémoire.

Pour un nombre de processeurs assez petit devant la taille du problème matriciel, le coût de communication devient négligeable par rapport à celui du calcul arithmétique. Dans ce cas, notre modèle choisi est très proche de la réalité et les résultats de complexité donnent une idée exacte sur les performances réelles de l'algorithme parallèle.

Dans tout ce qui suit, on note $e_{\alpha n}$ l'efficacité de l'algorithme parallèle considéré, le rapport $e_{\alpha n} = T_1/T(\alpha n)$ [26, 57], où αn est le nombre de processeurs, T_i le temps d'exécution de l'algorithme parallèle avec i processeurs (i=1 ou i=p).

Commentaires bibliographiques

L'introduction aux architectures multiprocesseurs s'inspire de plusieurs articles, citons [21, 29, 32, 61]. Les figures des paragraphes 2 et 3 sont dûes à [29] dans leur version anglaise originale. Le formalisme de la section 6 est introduit dans [32], l'exemple de la parallélisation de la boucle B est pris de l'article [44]. Je me suis beaucoup servi dans la rédaction de ce chapitre des travaux de Cosnard et Robert [11].

Chapitre II

TRIANGULARISATION DE MATRICES DENSES SUR UNE MACHINE MIMD

1. INTRODUCTION

La méthode d'élimination de Gauss est très utilisée pour résoudre un système d'équations linéaires sur une machine séquentielle. Six différentes versions pour une machine parallèle sont considérées en [15] dont trois sont implémentées par colonnes (comme il est préférable pour un environnement FORTRAN). Ces dernières sont discutées en détail, chacune d'elles correspond à un algorithme séquentiel dont la forme est désignée par une des permutations du triplet I, J, K. Elles sont nommées version SAXPY (forme KJI), version GAXPY (forme JKI) et version DOT (forme IJK).

Nous étudions aussi les versions parallèles des algorithmes LDM^t, LDL^t, Doolittle et de la décomposition de Cholesky. La version parallèle MIMD de l'algorithme d'élimination de Gauss avec pivotage partiel est discutée en [32, 37,60]. La parallélisation de la décomposition LDL^t est étudiée en [33].

Dans la section 2, nous présentons les versions séquentielles des algorithmes cités ci-dessus en indiquant les tâches avec leurs contraintes de précédence. Enfin, nous construisons le graphe des tâches pour chacune des versions. Nous remarquons que les graphes obtenus sont classifiés en six catégories nommées triangulaire (1), triangulaire (2), 2-pas, double triangulaire (1), double triangulaire (2) et double 2-pas.

La section 3 est consacrée à l'analyse théorique de ces graphes. Nous supposons que le nombre de processeurs est proportionnel à la dimension du problème, et nous établissons des résultats de complexité.

Dans la dernière section, nous calculons le temps d'exécution de chaque algorithme parallèle associé à l'un des dix-sept algorithmes séquentiels étudiés, et nous comparons les performances. Nous concluons que la version KJI-SAXPY de [15], sous une forme modifiée, apparaît la plus performante.

2. GRAPHES THEORIQUES

Nous considérons les algorithmes suivants:

(A) Algorithme de Gauss Forme KJI-SAXPY

Pour k=1à n-1
  Exécuter T_{kk} : < **Pour** i=k+1à n
    $a_{ik}=a_{ik}/a_{kk}$ >
  Pour j=k+1à n
    Exécuter T_{kj} : < **Pour** i=k+1à n
      $a_{ij}=a_{ij}-a_{ik}*a_{kj}$>

$Ex(T_{kk})$ = n-k opérations arithmétiques $1 \le k \le n-1$

$Ex(T_{kj})$ = 2(n-k) opérations arithmétiques $k+1 \le j \le n,\ 1 \le k \le n-1$

Le nombre total d'opérations arithmétiques est égal à $2n^3/3 + O(n^2)$.
Les contraintes de précédence sont:

$T_{kk} << T_{kj}$ $\quad k+1 \le j \le n,\ 1 \le k \le n-1$

$T_{kj} << T_{k+1,j}$ $\quad k+1 \le j \le n,\ 1 \le k \le n-1$

L'allure du graphe des tâches est:

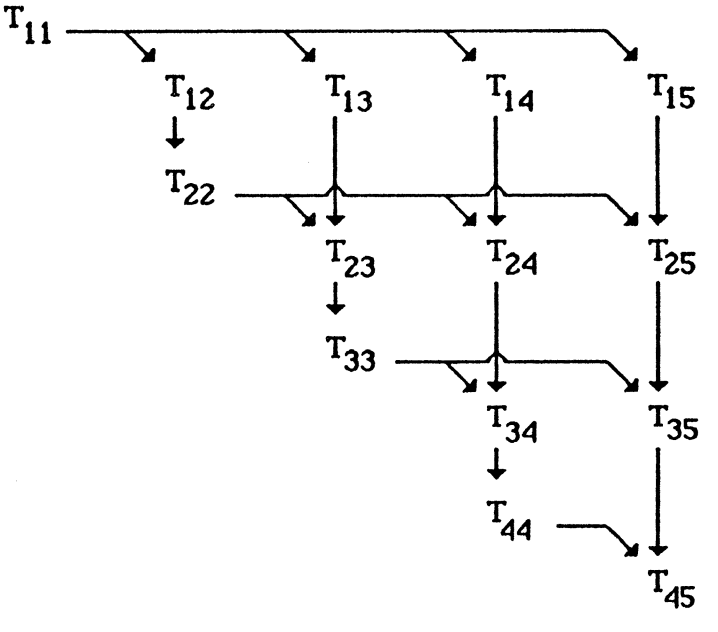


Figure 1: Graphe des tâches de l'Algorithme de Gauss pour une matrice de taille 5x5.

A l'étape k, les transformations effectuées sont les suivantes:

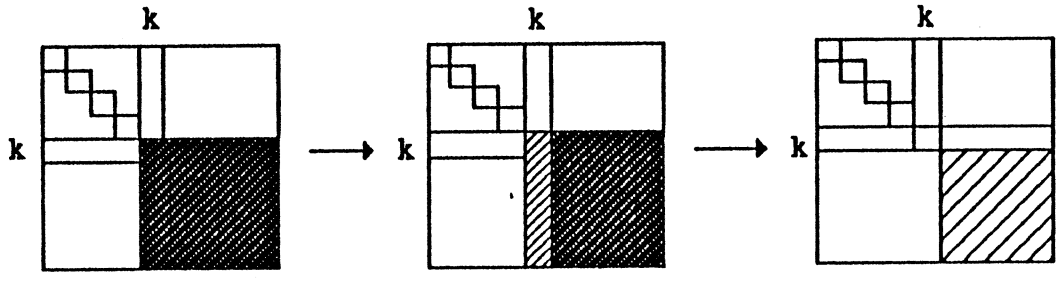


Figure 2: Exécution en parallèle de l'Algorithme de Gauss Forme KJI.

(B) Algorithme de Gauss Forme JKI-GAXPY

Pour j=1à n
  Pour k=1à j-1
    Exécuter T_{kj} : < **Pour** i=k+1à n
      $a_{ij}=a_{ij}-a_{ik}*a_{kj}$>
  Exécuter T_{jj} : < **Pour** i=j+1à n
    $a_{ij}=a_{ij}/a_{jj}$>

$Ex(T_{kj})$	=	2(n-k)	opérations arithmétiques	$1\leq k\leq j-1,\ 1\leq j\leq n$
$Ex(T_{jj})$	=	n-j	opérations arithmétiques	$1\leq j\leq n$

Le nombre total d'opérations arithmétiques est égal à $2n^3/3 + O(n^2)$.
Les contraintes de précédence sont:

$T_{kj} << T_{k+1,j}$ $\quad 1\leq k\leq n-1,\ k+1\leq j\leq n$
$T_{jj} << T_{jk}$ $\quad 1\leq k\leq n,\ k\leq j\leq n-1$

L'allure du graphe d'ordonnancement est présentée à la figure 1.
L'exécution en parallèle de l'étape k est présentée à la figure 3.

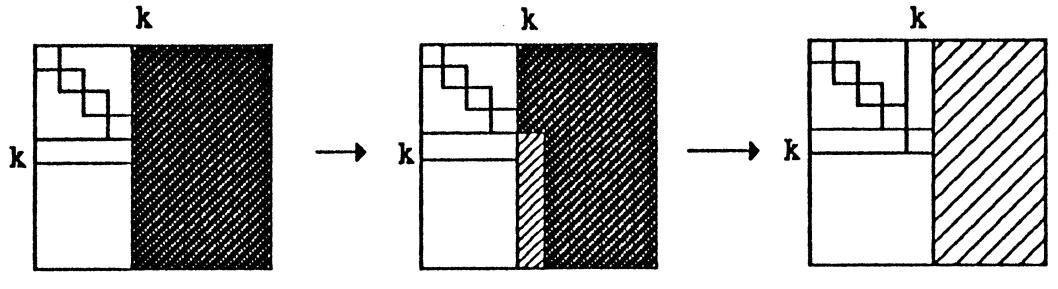


Figure 3: Exécution en parallèle de l'Algorithme de Gauss Forme JKI.

Les formes KJI-SAXPY et JKI-GAXPY ont le même graphe. Ceci est dû au fait que, pour un j donné, les tâches T_{ki}, $1 \leq k \leq i$, s'exécutent séquentiellement. Au contraire, pour un k donné, les tâches T_{ki}, $k+1 \leq i \leq n$, peuvent être exécutées en parallèle. Ceci montre que deux algorithmes différents peuvent avoir la même implémentation parallèle. L'élimination des cœfficients de la matrice A progresse différemment pour les deux algorithmes.

(C) Algorithme de Gauss Forme IJK-DOT

Pour i=2 **à** n
  Pour j=2 **à** i
    Exécuter T_{ij} :< $a_{i,j-1}=a_{i,j-1}/a_{j-1,j-1}$;
      Pour k=1 **à** j-1
        $a_{ij}=a_{ij}-a_{ik}*a_{kj}$>
  Pour j=i+1**à** n
    Exécuter U_{ij} : < **Pour** k=1 **à** i-1
        $a_{ij}=a_{ij}-a_{ik}*a_{kj}$>

$Ex(T_{ij})$	=	2j-1	opérations arithmétiques	$2 \leq j \leq i$, $3 \leq i \leq n$
$Ex(U_{ij})$	=	2(i-1)	opérations arithmétiques	$i+1 \leq j \leq n$, $2 \leq i \leq n$

Le nombre total d'opérations arithmétiques est égal à $2n^3/3 + O(n^2)$.
Les contraintes de précédence sont:

$T_{ij} << T_{i,j+1}$	$2 \leq j \leq i-1$, $j+1 \leq i \leq n$
$T_{ii} << U_{ij}$	$i+1 \leq j \leq n$, $2 \leq i \leq n-1$
$U_{ij} << U_{i+1,j}$	$i+2 \leq j \leq n$, $2 \leq i \leq n-2$
$U_{j,j+1} << T_{i,j+1}$	$i \leq j \leq n-1$, $2 \leq i \leq n$

Pour le graphe d'ordonnancement et l'exécution en parallèle voir les figures 4 et 5.

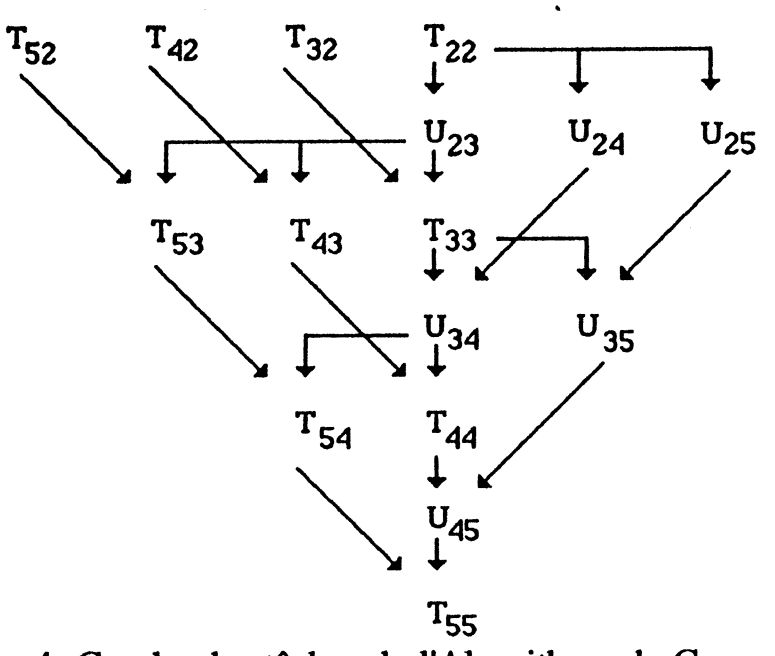


Figure 4: Graphe des tâches de l'Algorithme de Gauss Forme IJK pour une matrice de taille 5x5.

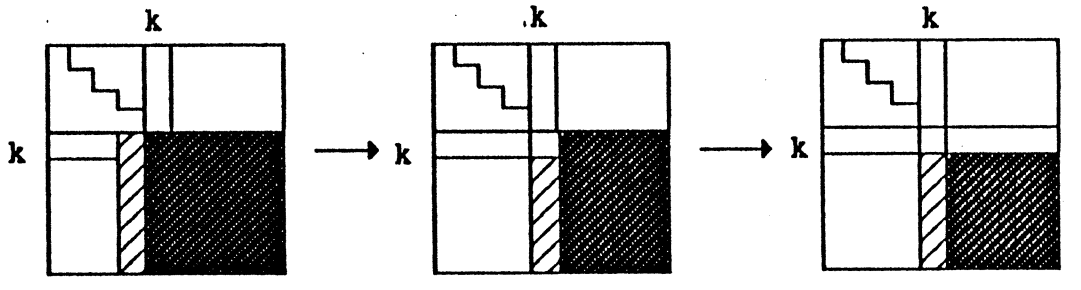


Figure 5: Exécution en parallèle de l'Algorihme de Gauss Forme IJK

(D) Algorithme de Gauss Forme IJK-DOT modifiée

Pour i=2 à n
  Pour j=2 à i-1
    Exécuter T_{ij} :< $a_{i,j-1}=a_{i,j-1} - a_{i,j-2} * a_{j-2,j-1}$;
      $a_{i,j-1}=a_{i,j-1}/a_{j-1,j-1}$;
      Pour k=1 à j-2
        $a_{ij}=a_{ij} - a_{ik}*a_{kj}$;>
  Exécuter T_{ii} :<$a_{i-1,i}=a_{i-1,i} - a_{i-1,i-2}* a_{i-2,i}$;
    $a_{i,\,i-1}=a_{i,i-1} - a_{i-2,i-1}* a_{i,i-2}$;
    $a_{i,i-1}=a_{i,i-1}/a_{i-1,i-1}$;
    Pour k=1à i-1
      $a_{ii}=a_{ii}-a_{ik}* a_{ki}$>

Pour j=i+1 à n
Exécuter U_{ij} : < $a_{i-1,j}=a_{i-1,j} - a_{i-1,i-2}*a_{i-2,j}$;
Pour k=1 à i-2
$a_{ij}=a_{ij} - a_{ik}*a_{kj}$;>

$Ex(T_{22})$	=	3	opérations arithmétiques	
$Ex(T_{i2})$	=	1	opération arithmétique	$3 \le i \le n$
$Ex(T_{ii})$	=	2i+3	opérations arithmétiques	$3 \le i \le n$
$Ex(T_{ij})$	=	2j-1	opérations arithmétiques	$i \le j \le n$, $3 \le i \le n-1$
$Ex(U_{ij})$	=	2(i-1)	opérations arithmétiques	$i+1 \le j \le n$, $3 \le i \le n-1$

Le nombre total d'opérations arithmétiques est égal à $2n^3/3 + O(n^2)$.
Les contraintes de précédence sont:

$T_{ii} << T_{j,i+1}$	$i+1 \le j \le n$, $3 \le i \le n-1$
$T_{ii} << U_{i+1,j}$	$i+1 \le j \le n$, $3 \le i \le n$
$T_{i+1,i} << T_{i+1,i+1}$	$2 \le i \le n-1$
$T_{i+1,i} << U_{i+1,j}$	$i+1 \le j \le n$, $3 \le i \le n-1$
$U_{ij} << U_{i+1,j}$	$i+2 \le j \le n$, $3 \le i \le n-2$
$T_{ij} << T_{i,j+1}$	$2 \le i \le n-1$, $i+1 \le j \le n$

Le graphe d'ordonnancement a l'allure présentée dans la figure suivante:

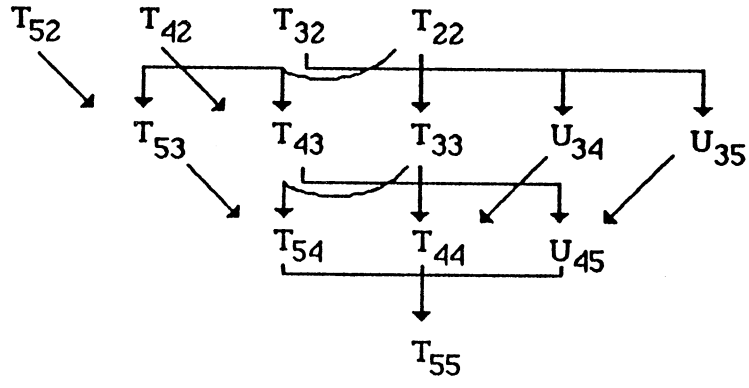


Figure 6: Graphe d'ordonnancement de l'Algorithme de Gauss
Forme IJK-DOT modifiée pour une matrice de taille 5x5.

Avec cette reformulation, les tâches T_{ii} et U_{ij}, $j \ge i+1$ sont maintenant indépendantes (on a éliminé $a_{i,i-1}$ de la tâche U_{ij}).
En exécutant le k-ème niveau du graphe, la matrice subit les transformations suivantes:

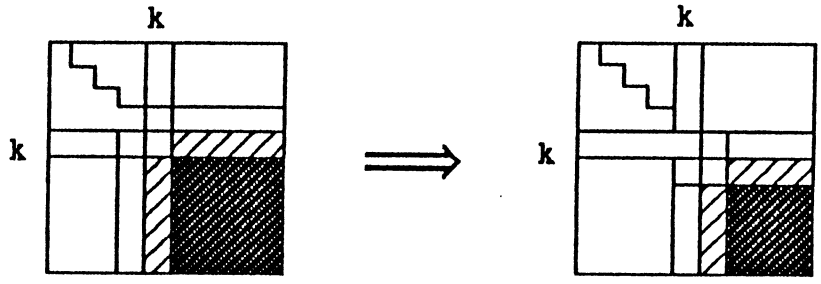


Figure 7: Exécution en parallèle de de l'Algorithme de Gauss Forme IJK-DOT modifiée.

(E) Algorithme de Gauss Forme KJI-SAXPY modifiée

Pour k=1à n-1
  Pour j=k+1à n
    Exécuter T_{kj} : < $a_{kj}=a_{kj}/a_{kk}$;
      Pour i=k+1à n
        $a_{ij}=a_{ij}-a_{ik}*a_{kj}$>

$Ex(T_{kj})$ = 2(n-k)+1 opérations arithmétiques $k+1 \leq j \leq n$, $1 \leq k \leq n-1$

Le nombre total d'opérations arithmétiques est égal à $2n^3/3 + O(n^2)$.

Les contraintes de précédence sont:

$T_{k,k+1} << T_{k+1,j}$ $k+2 \leq j \leq n$, $1 \leq k \leq n-2$

$T_{kj} << T_{k+1,j}$ $k+2 \leq j \leq n$, $1 \leq k \leq n-2$

L'allure du graphe est présentée à la figure 8.

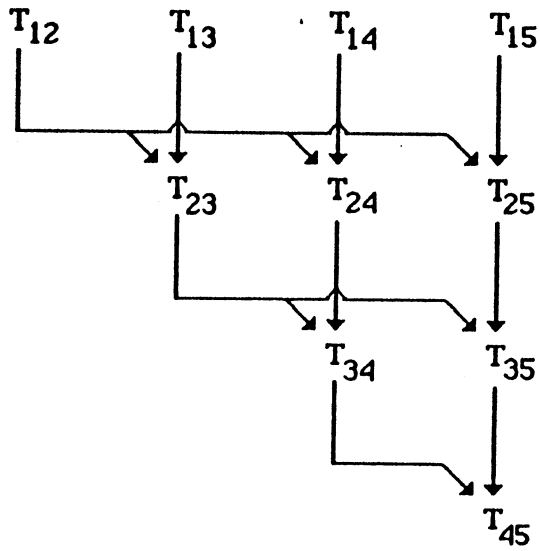


Figure 8: Graphe des tâches de l'Algorithme de Gauss Forme KJI modifiée pour une matrice de taille 5x5.

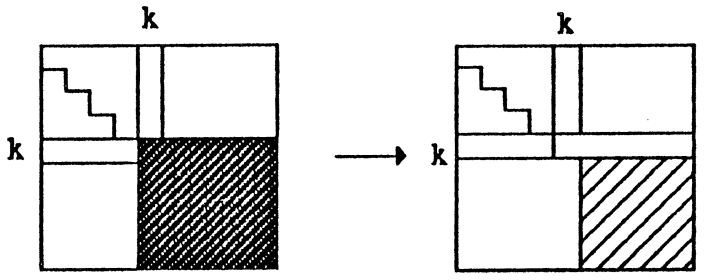


Figure 9: Exécution en parallèle de l'Algorithme de Gauss Forme KJI modifiée.

(F) Réduction de Doolittle (1) [25]

Pour k=1**à** n
  Pour j=k **à** n
    Exécuter T_{kj} :< **Pour** p=1**à** k-1
      $a_{kj}=a_{kj} - a_{kp}*a_{pj}$>
  Pour i=k+1**à** n
    Exécuter U_{ki} : < **Pour** p=1**à** k-1
      $a_{ik}=a_{ik} - a_{ip}*a_{pk}$;
    $a_{ik}=a_{ik} /a_{kk}$>

$Ex(T_{kj})$	=	2(k-1)	opérations arithmétiques	$k \le j \le n$, $1 \le k \le n$
$Ex(U_{kj})$	=	2k-1	opérations arithmétiques	$k+1 \le i \le n$, $1 \le k \le n-1$

Le nombre total d'opérations arithmétiques est égal à $2n^3/3 + O(n^2)$.
Les contraintes de précédence sont:

$T_{kj} << T_{k+1,j}$	$k+1 \le j \le n$, $2 \le k \le n-1$
$T_{kk} << U_{k,j}$	$k+1 \le j \le n$, $2 \le k \le n-1$
$U_{ki} << U_{k+1,i}$	$k+2 \le i \le n$, $2 \le k \le n-2$
$U_{kk+1} << T_{k+1,j}$	$k+1 \le j \le n$, $2 \le k \le n-1$

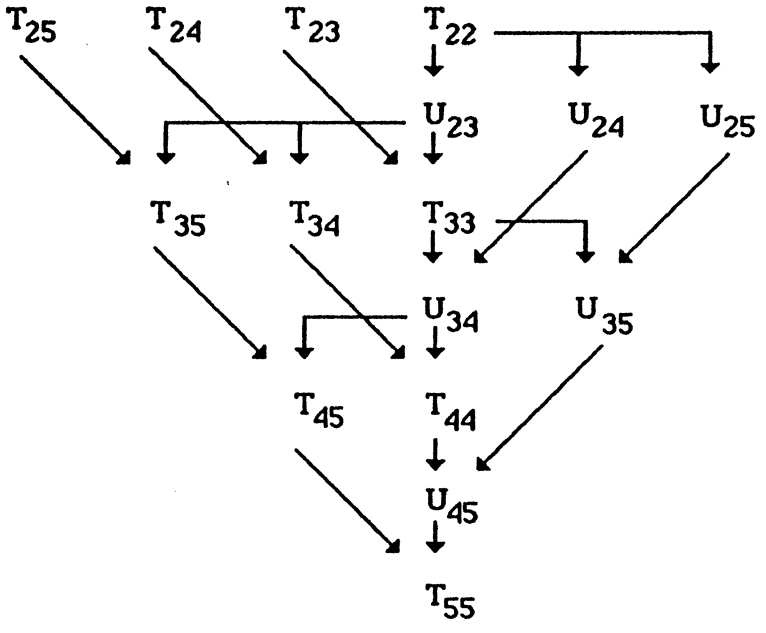


Figure 9: Graphe des tâches de l'Algorithme Réduction de Doolittle(1) pour une matrice de taille 5x5.

A l'étape k, la matrice subit les transformations suivantes:

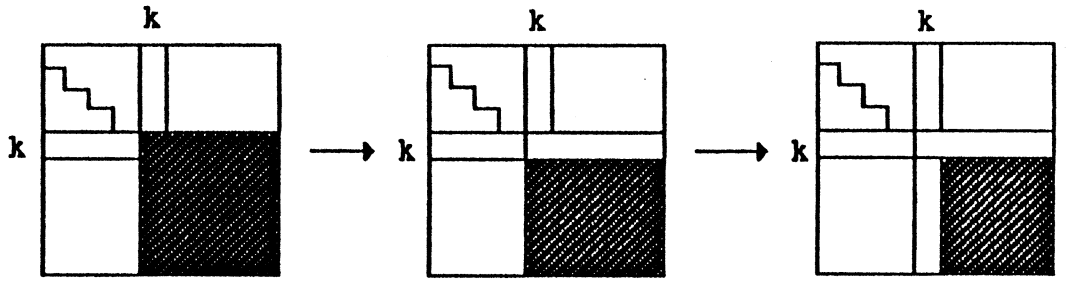


Figure 10: Exécution en parallèle de l'Algorithme Réduction de Doolittle (1).

Pour l'exécution parallèle de cet algorithme, le calcul de la ligne i débute avant celui de la colonne i, contrairement à l'algorithme forme (IJK).

(G) Réduction de Doolittle (2)

Pour k=2 à n
Exécuter T_{kk} :<$a_{k-1,k}=a_{k-1,k} - a_{k-1,k-2}*a_{k-2,k}$;
$a_{k,k-1}=a_{k,k-1} - a_{k,k-2}*a_{k-2,k-1}$;
$a_{k,k-1}= a_{k,k-1}/ a_{k-1,k-1}$;
Pour p=1à k-1
$a_{kk}=a_{kk} - a_{kp}*a_{pk}$;>

Pour j=k+1 à n
Exécuter T_{jk} :< $a_{k-1,j}=a_{k-1,j} - a_{k-1,k-2}*a_{k-2,j}$;
Pour p=1à k-2
$a_{kj}=a_{kj} - a_{kp}*a_{pj}$;>
Pour i=k+1à n
Exécuter U_{ik} : < $a_{i,k-1}=a_{i,k-1} - a_{i,k-2}*a_{k-2,k-1}$;
$a_{i,k-1}=a_{i,k-1} / a_{k-1,k-1}$;
Pour p=1à k-2
$a_{ik}=a_{ik} - a_{ip}*a_{pk}$;>

$Ex(T_{22})$	=	3	opérations arithmétiques	
$Ex(U_{i2})$	=	1	opération arithmétique	$3 \le i \le n$
$Ex(T_{kk})$	=	2k+3	opérations arithmétiques	$3 \le k \le n$
$Ex(T_{jk})$	=	2(k-1)	opérations arithmétiques	$k \le j \le n$, $3 \le k \le n$
$Ex(U_{ik})$	=	2k-1	opérations arithmétiques	$k+1 \le i \le n$, $3 \le k \le n-1$

Le nombre total d'opérations arithmétiques est égal à $2n^3/3 + O(n^2)$.

Les contraintes de précédence sont:

$T_{jk} << T_{j,k+1}$	$k+1 \le j \le n$, $3 \le k \le n-1$
$T_{kk} << U_{j,k+1}$	$k+2 \le j \le n$, $2 \le k \le n-2$
$U_{ik} << U_{i+1,k}$	$k+2 \le i \le n$, $2 \le k \le n-2$
$T_{kk} << T_{j,k+1}$	$k+1 \le j \le n$, $2 \le k \le n-1$
$U_{k,k-1} << T_{kj}$	$k \le j \le n$, $3 \le k \le n$

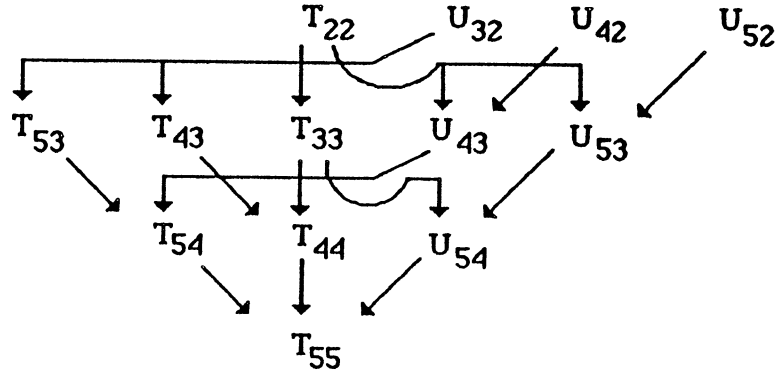


Figure 11: Graphe des tâches de l'Algorithme Réduction de Doolittle (2) pour une matrice de taille 5x5.

Ce graphe est le même que celui de la figure 6, et l'étape k de l'élimination est identique au schéma de la figure 7.

(H) Décomposition LDM^t non optimisée (1) [25]

Pour k=1à n
  Exécuter T_{kk} : < **Pour** p=1à k-1
    $a_{kk}=a_{kk} - a_{kp}*a_{pp}*a_{pk}$>
  Pour i=k+1à n
    Exécuter T_{ik} : < **Pour** p=1à k-1
      $a_{ik}=a_{ik} - a_{ip}*a_{pp}*a_{pk}$;
    $a_{ik}=a_{ik}/a_{kk}$>
  Pour j=k+1à n
    Exécuter T_{kj} : < **Pour** p=1à k-1
      $a_{kj}=a_{kj} - a_{kp}*a_{pp}*a_{pj}$;
    $a_{kj}=a_{kj}/a_{kk}$>

$Ex(T_{kk})$	=	3(k-1)	opérations arithmétiques	$1\le k\le n$
$Ex(T_{ik})$	=	3k-2	opérations arithmétiques	$k+1\le i\le n$, $1\le k\le n-1$
$Ex(T_{kj})$	=	3k-2	opérations arithmétiques	$k+1\le j\le n$, $1\le k\le n-1$

Le nombre total d'opérations arithmétiques est égal à $n^3 + O(n^2)$.
Les contraintes de précédence sont:

$T_{kk} << T_{kj}$	$k+1\le j\le n$, $2\le k\le n-1$
$T_{kk} << T_{ik}$	$k+1\le i\le n$, $2\le k\le n-1$
$T_{ik} << T_{i,k+1}$	$k+1\le i\le n$, $1\le k\le n-1$
$T_{kj} << T_{k+1,j}$	$k+1\le j\le n$, $1\le k\le n-1$

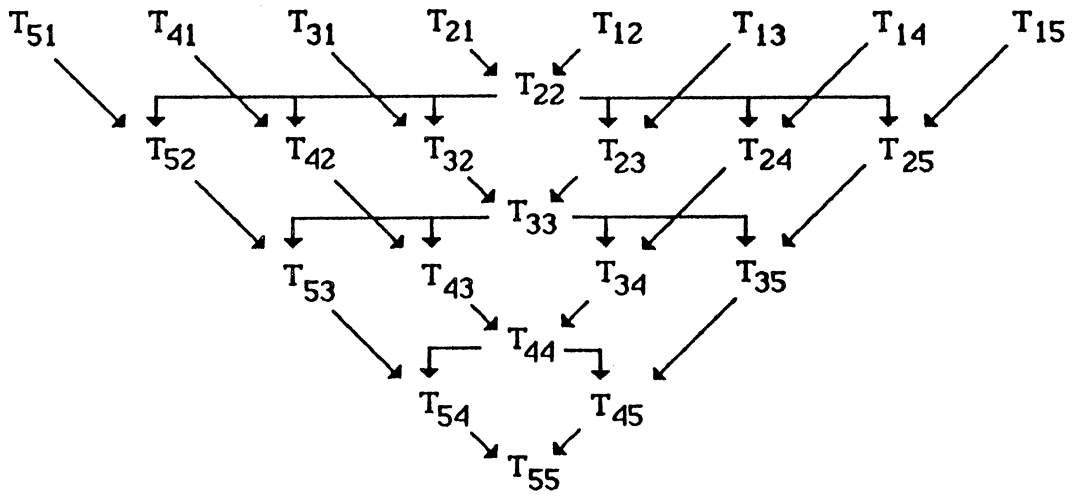


Figure 12: Graphe des tâches de l'Algorithme de Décomposition LDM^t non optimisée (1) pour une matrice de taille 5x5.

L'exécution à l'étape k transforme la matrice comme suit:

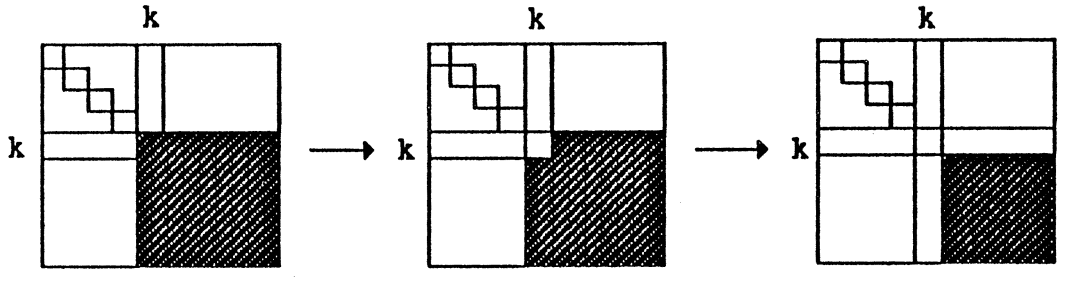


Figure 13: Exécution en parallèle de l'Algorithme de Décomposition LDM^t non optimisée (1).

(I) Décomposition LDM^t non optimisée (2)

Pour k=2 **à** n

Exécuter T_{kk} : < $a_{k,k-1}=a_{k,k-1} - a_{k,k-2}*a_{k-2,k-2}*a_{k-2,k-1}$;
$a_{k-1,k}=a_{k-1,k} - a_{k-2,k}*a_{k-2,k-2}*a_{k-1,k-2}$;
$a_{k,k-1}=a_{k,k-1}/a_{k-1,k-1}$;
$a_{k-1,k}=a_{k-1,k}/a_{k-1,k-1}$;

Pour p=1**à** k-1
$a_{kk}=a_{kk} - a_{kp}*a_{pp}*a_{pk}$;>

Pour i=k+1**à** n

Exécuter T_{ik} : < $a_{i,k-1}=a_{i,k-1} - a_{i,k-2}*a_{k-2,k-2}*a_{k-2,k-1}$;
$a_{k-1,i}=a_{k-1,i} - a_{k-2,i}*a_{k-2,k-2}*a_{k-1,k-2}$;

$a_{i,k-1}=a_{i,k-1}/a_{k-1,k-1}$;
$a_{k-1,i}=a_{k-1,i}/a_{k-1,k-1}$;
Pour p=1 à k-2 **faire**
début
$a_{ik}=a_{ik} - a_{ip}*a_{pp}*a_{pk}$;
$a_{ki}=a_{ki} - a_{pi}*a_{pp}*a_{kp}$;
fin;>

$Ex(T_{22})$	=	5	opérations arithmétiques	
$Ex(T_{i2})$	=	2	opérations arithmétiques	$3\leq i\leq n$
$Ex(T_{kk})$	=	3k+5	opérations arithmétiques	$3\leq k\leq n$
$Ex(T_{ik})$	=	6k-4	opérations arithmétiques	$k+1\leq i\leq n$, $3\leq k\leq n$

Le nombre total d'opérations arithmétiques est $n^3+O(n^2)$.
Les contraintes de précédence sont:

$T_{ik} << T_{i,k+1}$	$k+2\leq i\leq n$, $2\leq k\leq n-1$
$T_{kk} << T_{i,k+1}$	$k+1\leq i\leq n$, $2\leq k\leq n-1$
$T_{k,k-1}<<T_{ik}$	$k+1\leq i\leq n$, $3\leq k\leq n-1$

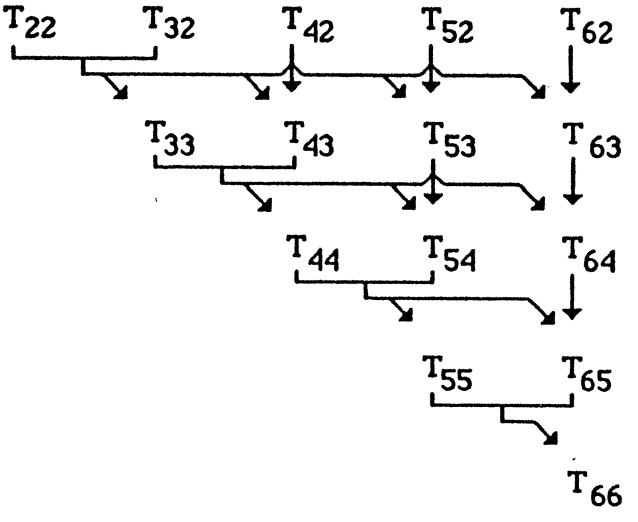


Figure 14: Graphe des tâches de la décomposition LDM^t non optimisée (2) pour une matrice de taille 6x6.

Lors de l'exécution en parallèle, à l'étape k, la matrice a la forme présentée à la figure 7.

(J) Décomposition LDM^t optimisée (1) [25]

Pour k=1à n
  Exécuter T_{kk} : < **Pour** p=1à k-1 **faire**
    début
    $r_p=d_p*a_{pk}$; $w_p=a_{kp}*d_p$;
    $a_{kk}=a_{kk} - a_{kp}*r_p$;
    fin;>
  Pour i=k+1à n
    Exécuter T_{ik} : < **Pour** p=1à k-1
      $a_{ik}=a_{ik} - a_{ik}*r_p$;
    $a_{ik}=a_{ik}/a_{kk}$>
  Pour j=k+1à n
    Exécuter T_{kj} : < **Pour** p=1à k-1
      $a_{kj}=a_{kj} - w_p*a_{pj}$;
    $a_{kj}=a_{kj}/a_{kk}$>

$Ex(T_{kk})$	=	4(k-1)	opérations arithmétiques	$1\le k\le n$
$Ex(T_{ik})$	=	2k-1	opérations arithmétiques	$k+1\le i\le n$, $1\le k\le n$
$Ex(T_{kj})$	=	2k-1	opérations arithmétiques	$k+1\le j\le n$, $1\le k\le n$

Le nombre total d'opérations arithmétiques est égal à $2n^3/3 + O(n^2)$.
Les contraintes de précédence et le graphe des tâches sont identiques à ceux de la décomposition LDM^t non optimisée (1) (voir figure 12), mais les longueurs des tâches sont différentes. L'exécution de l'étape k modifie la matrice comme le montre la figure 13.

(K) Décomposition LDM^t optimisée (2)

Pour k=2 à n
  Exécuter T_{kk} : < $a_{k,k-1}=a_{k,k-1} - a_{k,k-2}*r_{k-2}$;
    $a_{k-1,k}=a_{k-1,k} - a_{k-2,k}*w_{k-2}$;
    $a_{k,k-1}=a_{k,k-1}/a_{k-1,k-1}$;
    $a_{k-1,k}=a_{k-1,k}/a_{k-1,k-1}$;
    Pour p=1à k-1
      début
      $r_p=d_p*a_{pk}$;$w_p=a_{kp}*d_p$;
      $a_{kk}=a_{kk} - a_{kp}*r_p$;
      fin;>

Pour i=k+1 **à** n
Exécuter T_{ik}: < $a_{i,k-1}=a_{i,k-1} - a_{i,k-2}* a_{k-2,k-2}*a_{k-2,k-1}$;
$a_{k-1,i}=a_{k-1,i} - a_{k-2,i}* a_{k-2,k-2}*a_{k-1,k-2}$;
$a_{i,k-1}=a_{i,k-1}/a_{k-1,k-1}$;
$a_{k-1,i}=a_{k-1,i}/a_{k-1,k-1}$;
Pour p=1 **à** k-2 **faire**
début
$a_{ik}=a_{ik} - a_{ip}*r_p$;
$a_{ki}=a_{ki} - a_{pi}*w_p$;
fin;>

$Ex(T_{22})$	=	6	opérations arithmétiques	
$Ex(T_{i2})$	=	2	opérations arithmétiques	$3 \le i \le n$
$Ex(T_{kk})$	=	4k+2	opérations arithmétiques	$3 \le k \le n$
$Ex(T_{ik})$	=	4k	opérations arithmétiques	$k+1 \le i \le n$, $3 \le k \le n$

Le nombre total d'opérations arithmétiques est $2n^3/3+O(n^2)$.
Les contraintes de précédence et le graphe des tâches sont identiques à ceux de la décomposition LDM^t non optimisée (2) (voir figure 14), mais les temps d'exécution des tâches sont différents. L'exécution de l'étape k modifie la matrice comme le montre la figure 7.

(L) Décomposition LDL^t non optimisée (1) [25, 33]

Pour k=1 **à** n
Exécuter T_{kk} : < **Pour** p=1 **à** k-1
$a_{kk}=a_{kk} - a_{kp}*a_{pp}*a_{kp}$>
Pour i=k+1 **à** n
Exécuter T_{ik} : < **Pour** p=1 **à** k-1
$a_{ik}=a_{ik} - a_{ip}*a_{pp}*a_{kp}$;
$a_{ik}=a_{ik}/a_{kk}$>

$Ex(T_{kk})$	=	3(k-1)	opérations arithmétiques	$1 \le k \le n$
$Ex(T_{ik})$	=	3k-2	opérations arithmétiques	$k+1 \le i \le n$, $1 \le k \le n-1$

Le nombre total d'opérations arithmétiques est égal à $n^3/2 + O(n^2)$.
Les contraintes de précédence sont:

$T_{kk} << T_{ik}$ $\quad k+1 \le i \le n,\ 2 \le k \le n-1$

$T_{ik} << T_{i,k+1}$ $\quad k+1 \leq i \leq n,\ 2 \leq k \leq n-1$

L'allure du graphe des tâches est présentée à la figure suivante :

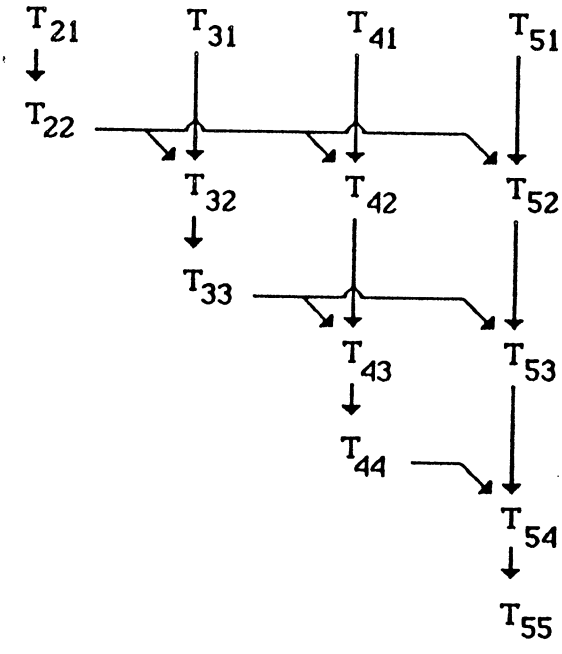


Figure 15: Graphe des tâches de la décomposition LDL^t non optimisée (1) pour une matrice de taille 5x5.

En parallèle, l'exécution à l'étape k s'effectue comme l'indique la figure suivante :

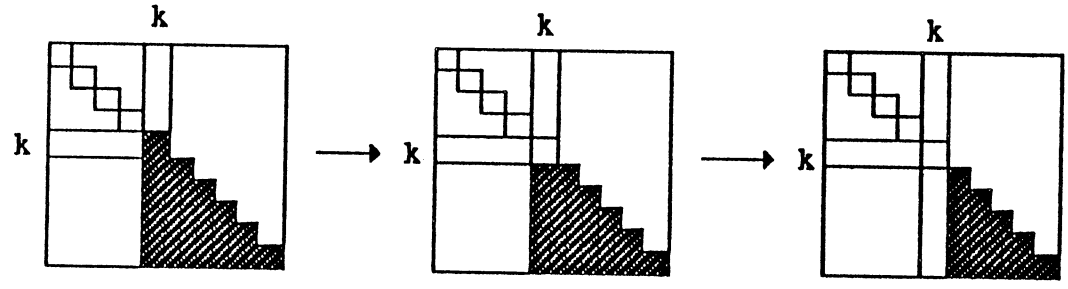


Figure 16: Exécution en parallèle de la décomposition LDL^t non optimisée (1).

(M) Décomposition LDL^t non optimisée (2)

Pour k=2 **à** n
  Exécuter T_{kk} : $<a_{k,k-1}=a_{k,k-1} - a_{k,k-2}*a_{k-2,k-2}*a_{k-1,k-2}$;
    $a_{k,k-1}=a_{k,k-1}/a_{k-1,k-1}$;
    Pour p=1 **à** k-1

```
                                    akk=akk - akp*app*akp>
        Pour  i=k+1à n
              Exécuter Tik : < ai,k-1=ai,k-1 - ai,k-2*ak-2,k-2*ak-1,k-2;
                               ai,k-1=ai,k-1/ak-1,k-1;
                               Pour  p=1 à k-2
                                     aik=aik - aip*app*akp;>
```

$Ex(T_{22})$	=	4	opérations arithmétiques	
$Ex(T_{i2})$	=	1	opération arithmétique	$3 \le i \le n$
$Ex(T_{kk})$	=	3k+1	opérations arithmétiques	$3 \le k \le n$
$Ex(T_{ik})$	=	3k-2	opérations arithmétiques	$k+1 \le i \le n$, $3 \le k \le n-1$

Le nombre total d'opérations arithmétiques est égal à $n^3/2 + O(n^2)$.
Les contraintes de précédence et le graphe des tâches sont identiques à ceux de la décomposition LDM^t non optimisée (2) (voir figure 14), mais les temps d'exécution des tâches d'un même niveau sont différents. L'exécution de l'étape k modifie la matrice comme le montre la figure 7. Cette modification ne concerne que la partie inférieure de la matrice puisque cette dernière est symétrique.

(N) Décomposition LDL^t optimisée (1) [25]

```
        Pour  k=1à n
              Exécuter Tkk  : < Pour  p=1à k-1faire
                               début
                               rp=dp*apk;akk=akk - akp*rp;
                               fin >
              Pour  i=k+1à n
                    Exécuter Tik  : < Pour   p=1à k-1
                                         aik=aik - aip*rp;
                                     aik=aik/akk>
```

$Ex(T_{kk})$	=	3(k-1)	opérations arithmétiques	$1 \le k \le n$
$Ex(T_{ik})$	=	2k-1	opérations arithmétiques	$k+1 \le i \le n$, $1 \le k \le n-1$

Le nombre total d'opérations arithmétiques est égal à $n^3/3 + O(n^2)$.
Les contraintes de précédence et le graphe des tâches sont identiques à ceux de la décomposition LDL^t non optimisée (1) (voir figure 15), mais les temps d'exécution des tâches sont différents. La figure 16 montre les modifications que la matrice doit subir à l'étape k. La matrice étant symétrique, les transformations

se font seulement sur la partie inférieure.

(O) Décomposition LDL^t optimisée (2)

Pour k=2 **à** n
  Exécuter T_{kk} : <$a_{k,k-1}=a_{k,k-1} - a_{k,k-2}*r_{k-2}$;
    $a_{k,k-1}=a_{k,k-1}/a_{k-1,k-1}$;
    Pour p=1 **à** k-1
      début
      $r_p=d_p*a_{pk}$; $a_{kk}=a_{kk} - a_{kp}*r_p$;
      fin;>
  Pour i=k+1 **à** n
    Exécuter T_{ik} : < $a_{i,k-1}=a_{i,k-1} - a_{i,k-2}*a_{k-2,k-2}*a_{k-1,k-2}$;
      $a_{i,k-1}=a_{i,k-1}/a_{k-1,k-1}$;
      Pour p=1 **à** k-2
        $a_{ik}=a_{ik} - a_{ip}*r_p$;>

$Ex(T_{22})$	=	4	opérations arithmétiques	
$Ex(T_{i2})$	=	1	opération arithmétique	$3 \le i \le n$
$Ex(T_{kk})$	=	3k	opérations arithmétiques	$3 \le k \le n$
$Ex(T_{ik})$	=	2k	opérations arithmétiques	$k+1 \le i \le n$, $3 \le k \le n-1$

Le nombre total d'opérations arithmétiques est égal à $n^3/3 + O(n^2)$.
Les contraintes de précédence et le graphe des tâches sont identiques à ceux de la décomposition LDM^t non optimisée (2) (voir figure 14), mais les temps d'exécution des tâches sont différents. L'exécution de l'étape k modifie la matrice comme le montre la figure 7. Cette modification ne concerne que la partie inférieure de la matrice puisque cette dernière est symétrique.

(P) Décomposition de Cholesky (1) GG^t [25]

Pour k=1 **à** n
  Exécuter T_{kk} : < **Pour** p=1 **à** k-1
      $a_{kk}=a_{kk} - a_{kp}*a_{kp}$;
    $a_{kk}=sqrt(a_{kk})$>
  Pour i=k+1 **à** n
    Exécuter T_{ik} : < **Pour** p=1 **à** k-1
      $a_{ik}=a_{ik} - a_{ip}* a_{kp}$;

$a_{ik}=a_{ik}/a_{kk}$ >

$Ex(T_{kk})$	=	2k-1	opérations arithmétiques	$1 \le k \le n$
$Ex(T_{ik})$	=	2k-1	opérations arithmétiques	$k+1 \le i \le n$, $1 \le k \le n-1$

Le nombre total d'opérations arithmétiques est égal à $n^3/3 + O(n^2)$, en supposant qu'une racine carrée s'exécute en une unité de temps.
Les contraintes de précédence et le graphe des tâches sont les mêmes que ceux de la décomposition LDL^t non optimisée (1) (voir figure 15). Les longueurs des tâches d'un même niveau sont différentes.

(Q) Décomposition de Cholesky (2) GG^t

Exécuter T_{11} :<$a_{11}=sqrt(a_{11})$;>
Pour k=2 **à** n
Exécuter T_{kk} : < $a_{k,k-1}=a_{k,k-1} - a_{k,k-2}* a_{k-1,k-2}$;
$a_{k,k-1}=a_{k,k-1}/a_{k-1,k-1}$;
Pour p=1**à** k-1
$a_{kk}=a_{kk} - a_{kp}*a_{kp}$;
$a_{kk}=sqrt(a_{kk})$>
Pour i=k+1**à** n
Exécuter T_{ik} : < $a_{i,k-1}=a_{i,k-1} - a_{i,k-2}* a_{k-1,k-2}$;
$a_{i,k-1}=a_{i,k-1}/a_{k-1,k-1}$;
Pour p=1**à** k-2
$a_{ik}=a_{ik} - a_{ip}* a_{kp}$;>

$Ex(T_{11})$	=	1	opération arithmétique	
$Ex(T_{22})$	=	4	opérations arithmétiques	
$Ex(T_{i2})$	=	1	opération arithmétique	$3 \le i \le n$
$Ex(T_{kk})$	=	2k+2	opérations arithmétiques	$3 \le k \le n$
$Ex(T_{ik})$	=	2k-1	opérations arithmétiques	$k+1 \le i \le n$, $3 \le k \le n$

Le nombre total d'opérations arithmétiques est égal à $n^3/3 + O(n^2)$.
On ne tiendra pas compte de la tâche T_{11} dans le reste du travail.
Les contraintes de précédence et le graphe des tâches sont identiques à ceux de la décomposition LDM^t non optimisée (2) (voir figure 14), mais les temps d'exécution des tâches sont différents. L'exécution de l'étape k modifie la matrice comme montre la figure 7. Cette modification ne concerne que la partie inférieure de la matrice puisque cette dernière est symétrique.

Chacun des algorithmes (C), (F), (H), (J), (L), (N) et (P) a été reformulé sous forme d'une version modifiée. Celle-ci est obtenue en modifiant le contenu des tâches de la première version. Cette modification permet d'avoir une seule étape par niveau au lieu de deux étapes: c'est grâce à l'élimination de la contrainte qui partage un niveau en deux sous niveaux.

En général, la triangularisation d'une matrice constitue la première phase de la résolution d'un système linéaire. Soit Ax=b un système à résoudre, où A est une matrice dense de taille nxn, x et b deux vecteur de longueur n chacun, le deuxième membre de l'équation précédente subit les mêmes transformations qu'une colonne de la matrice A. La matrice à triangulariser possède alors la taille nx(n+1). Pour valider les décompositions présentées, nous avons programmé toutes les versions précédentes sur une machine séquentielle. La résolution d'un système triangulaire est nécessaire après la triangularisation. Pour la version (E): algorithme de Gauss Forme KJI-SAXPY modifiée, il faut arrêter l'exécution à la ligne k=n, contrairement aux autres versions où on arrête l'exécution à la ligne k=n-1. Remarquons aussi que si on triangularise seulement la matrice A, la matrice triangulaire obtenue en utilisant cette dernière version est différente de celle obtenue en utilisant les autres versions de l'algorithme d'élimination de Gauss.

3. ANALYSE DES GRAPHES THEORIQUES

Au paragraphe précédent, nous avons passé en revue dix-sept algorithmes. Ils correspondent à neuf graphes. En réalité, nous avons seulement six types de graphes notés:

- triangulaire (1): graphe de l'algorithme (E).
- triangulaire (2): graphe des algorithmes (I), (K), (M), (O) ou (Q).
- 2-pas: graphe des algorithmes (A), (B), (L), (N) ou (P).
- double triangulaire (1): graphe des algorithmes (C) ou (F).
- double triangulaire (2): graphe algorithmes suivants (D) ou (G).
- double 2-pas: graphe des algorithmes (H) ou (J).

La différence entre les graphes triangulaire (i) (i=1, 2) réside dans le fait que le graphe triangulaire (2) possède une contrainte d'ordonnancement supplémentaire. Les contraintes d'ordonnancement du graphe triangulaire (1) sont aussi des contraintes d'ordonnancement pour le graphe triangulaire (2). Tout algorithme exécutant le graphe triangulaire (2) exécute aussi le graphe triangulaire (1). Nous étudions donc les deux graphes en même temps. Les graphes double triangulaire (1), (2) et double 2-pas peuvent être déduits respectivement des graphes triangulaires (1), (2) et du graphe 2-pas.

Dans ce paragraphe, on s'intéresse tout d'abord à l'analyse des trois premiers types de graphes. Nous construisons les algorithmes parallèles correspondants et nous établissons les résultats de complexité pour ces types de graphes. Nous déduisons de ces résultats, l'analyse des graphes restants. Comme nous avons

indiqué au départ, nous limitons le nombre de processeurs à p=O(n) pour la triangularisation d'une matrice d'ordre n. Plus précisement, nous prenons $p=\alpha n$ où $\alpha \in]0,1]$ et nous déduisons les résultats asymptotiques quand n tend vers l'infini. Nous notons les p processeurs $P_1, P_2, \ldots, P_p$.

3.1. GRAPHE TRIANGULAIRE

Nous étudions dans ce paragraphe les graphes triangulaires (1) et (2). Le graphe triangulaire est rencontré dans la parallélisation de la méthode de résolution des systèmes linéaires par diagonalisation [9, 10].

Forme générale de l'algorithme du graphe triangulaire

Pour k<-1 **à** n-1
  Pour i<-k+1 **à** n
    Exécuter T_{ki}

Les contraintes d'ordonnancement sont:

(A)	$T_{k,k+1}$	<<	$T_{k+1,j}$	$1 \le k \le n-1,\ k+2 \le j \le n$
(B)	$T_{k,j}$	<<	$T_{k+1,j}$	$1 \le k \le n-1,\ k+1 \le j \le n$
(C)	$T_{k,k+2}$	<<	$T_{k+1,j}$	$1 \le k \le n-1,\ k+2 \le j \le n$

La contrainte (C) n'existe pas pour le graphe triangulaire (1). Les tâches des exemples de la section 2 pour le graphe triangulaire (2) et celles du cas général n'ont pas les mêmes indices.

$Ex(T_{ki}) = bt_k$ unités de temps $k+1 \le i \le n$

où b est un entier et $t_k=k$ pour tout k ou $t_k=n-k$ pour tout k.

Nous négligeons les termes indépendants de k parce que ces derniers n'affectent pas l'évaluation asymptotique.

Nous avons maintenant un graphe où le nombre de tâches par niveau décroit.

T_{12}	T_{13}	...	T_{1k}	...	T_{1n}	niveau N_1
	T_{23}	...	T_{2k}	...	T_{2n}	niveau N_2
		...		...		
			$T_{k-1,k}$	...	$T_{k-1,n}$	niveau N_{k-1}
					$T_{n-1,n}$	niveau N_{n-1}

Figure 16 : L'allure du graphe triangulaire pour une matrice de taille nxn.

Le plus long chemin est formé par les tâches $T_{k-1,k}$ où k vérifie $2 \leq k \leq n$. Sa longueur est égale au temps T_{opt} qui est la borne inférieure des temps mis pour l'exécution parallèle de tout algorithme. On a dans les deux cas

$$T_{opt} = \sum_{k=2}^{n} bt_k = b\frac{n^2}{2} + O(n)$$

Il est évident de construire en utilisant n-1 processeurs un algorithme s'exécutant en T_{opt}, avec une efficacité asymptotique égale à $e_{n-1,\infty}=2/3$.
Nous allons montrer que parmi les algorithmes asymptotiquement optimaux, il en existe qui satisfont de plus à la contrainte ($\mathfrak{C}$):

$$(\mathfrak{C})\quad T_{kj} \leq\leq T_{k,j+1}$$

où la notation $T \leq\leq T'$ signifie que l'exécution de la tâche T' ne peut débuter avant celle de la tâche T (mais la simultanéité est possible)

Algorithme Glouton G

Nous introduisons maintenant l'algorithme Glouton, défini informellement de la manière suivante: l'algorithme G exécute les tâches en balayant les lignes du graphe d'ordonnancement de gauche à droite, en commençant à tout instant le maximum de tâches. Plus précisement, l'algorithme G exécute les tâches dans l'ordre partiel suivant:

$$T_{12} \leq\leq T_{13} \leq\leq \ldots \leq\leq T_{1n} \leq\leq T_{23} \leq\leq T_{24} \leq\leq T_{2n} \leq\leq T_{34} \leq\leq \ldots \leq\leq T_{n-1,n}$$

A l'instant t=1, l'algorithme G commence l'exécution des tâches $T_{12}, T_{13}, \ldots, T_{1p}$.
A tout instant $t \geq 1$, si $q \leq p$ processeurs sont disponibles, l'algorithme G les affecte à l'exécution des q tâches suivantes, du moins tant que les contraintes le permettent (sinon, on effectue le nombre maximum de tâches exécutables).

Pour évaluer le temps d'exécution de l'algorithme Glouton, nous notons que jusqu'à l'exécution de la tâche $T_{n-p-1,n}$ tous les processeurs sont actifs sans interruption: en effet, d'une part il y a plus de p tâches par niveau jusqu'au niveau N_{n-p-1} et d'autre part, la longueur des tâches situées sur le même niveau est constante. Le temps d'exécution jusqu'au niveau N_{n-p-1} est donc égal au temps d'exécution séquentielle de toutes les tâches situées entre les niveaux N_1 et N_{n-p-1}, divisé par le nombre de processeurs p (en d'autres termes G est d'efficacité asymptotique 1 jusqu'au niveau N_{n-p-1}). Nous obtenons une première valeur,

$$\left(\frac{b}{p}\right)\sum_{i=1}^{n-p-1}(n-i)t_i+O(n)$$

à laquelle on ajoute le temps d'exécution des tâches $T_{n-p,n-p+1}$, $T_{n-p+1,n-p+2}$, ..., $T_{n-1,n}$, soit

$$b\sum_{i=n-p}^{n-1}t_i+O(n)$$

Soit finalement

$$T_p=\left(\frac{b}{p}\right)\sum_{i=1}^{n-p-1}(n-i)t_i+b\sum_{i=n-p}^{n-1}t_i+O(n)$$

Proposition 2.1. L'algorithme Glouton G est asymptotiquement optimal. Son temps d'exécution est donné par

$$T_p=\left(\frac{b}{p}\right)\sum_{i=1}^{n-p-1}(n-i)t_i+b\sum_{i=n-p}^{n-1}t_i+O(n)$$

Preuve

Pour montrer que l'algorithme Glouton est asymptotiquement optimal, nous allons calculer une borne inférieure de T_{opt}. Soit t_0 le temps minimal de fin d'exécution de la tâche $T_{n-p,n}$ par un algorithme asymptotiquement optimal, nous avons alors

$$t_0 \geq \frac{b}{p}\sum_{i=1}^{n-p}(n-i)t_i+O(n).$$

Toutes les tâches situées sur les niveaux N_1, N_2, ..., N_{n-p} s'exécutent avant la tâche $T_{n-p+1,n}$. Les tâches $\{T_{n-p+1,n}, T_{n-p+2,n}, \ldots, T_{n-1,n}\}$ s'exécutent obligatoirement séquentiellement. Alors, le temps d'exécution optimal T_{opt} vérifie:

$$T_{opt} \geq \frac{b}{p}\sum_{i=1}^{n-p}(n-i)t_i+b\sum_{i=n-p+1}^{n-1}t_i+O(n).$$

Posons $p=\alpha n$ nous obtenons

(i) $t_k = k$ $\qquad T_p = bn^2\dfrac{1+3\alpha^2-\alpha^3}{6\alpha}+O(n)$

(ii) $t_k = n-k$ $\qquad T_p = bn^2 - \frac{2+\alpha^3}{6\alpha} + O(n)$

L'algorithme, (O) décomposition LDL^t optimisée (2), possède comme graphe de tâches le graphe triangulaire (2). La première tâche de chaque niveau N_k a une longueur ak, tandis que les autres tâches du même niveau ont une même longueur qui est bk, avec a et b des entiers différents. L'algorithme défini pour exécuter le graphe 2-pas (paragraphe suivant), exécute aussi ce graphe avec $s = \lceil a/b \rceil$. Le processeur P_0 exécute la première tâche de chaque niveau. Les autres processeurs exécutent la partie restante du graphe.

L'algorithme balaye le graphe d'ordonnancement sans interruption jusqu'au niveau $N_{(n-sp)^+}$ (l'efficacité égale à 1 jusqu'au niveau $N_{(n-sp)^+}$). Le temps d'exécution de cette première partie du graphe est:

$$\tau_1 = \left(\frac{b}{p}\right) \sum_{k=1}^{(n-sp)^+} (n-k)t_k + O(n)$$

Du niveau $N_{(n-sp+1)^+}$ jusqu'au niveau $N_{(n-(s-1)p)^+}$, l'exécution a pour longueur

$$\tau_2 = \sum_{k=(n-sp+1)^+}^{(n-(s-1)p)^+} sbt_k + O(n)$$

La longueur de l'exécution la dernière partie du graphe, du niveau $N_{(n-(s-1)p)^+ +1}$ jusqu'au niveau N_n, est

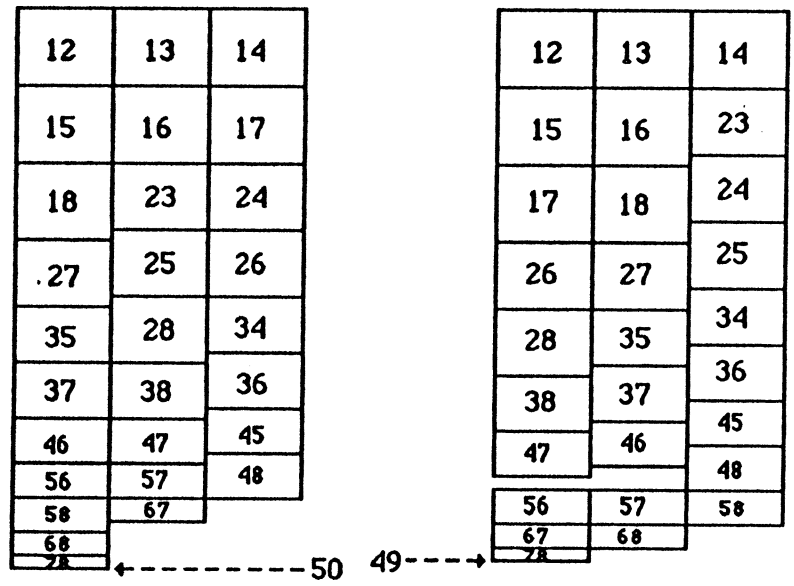
$$\tau_3 = \sum_{k=(n-(s-1)p)^+ +1}^{n} \sup(a,b)t_k + O(n)$$

Pour $t_k=k$ ou $t_k=n-k$, on a les résultats suivants avec $p=\alpha n$ ($\alpha \in]0,1]$) :

Pour	$\alpha \in]0,1/s[$	T_p	=	$\tau_1+\tau_2+\tau_3$	
Pour	$\alpha \in [1/s,1/(s-1)[$	T_p	=	$\tau_2+\tau_3$	
Pour	$\alpha \in [1/(s-1),1]$	T_p	=	τ_3	(s>2)

Comme l'indique l'exemple suivant communiqué par G.Rote [52], l'algorithme Glouton, s'il est asymptotiquement optimal, n'est pas optimal. Prenons n=8, p=3, b=1 et $t_k=n-k$, le temps d'exécution de l'algorithme Glouton est 50 unités de temps par contre il existe un autre algorithme s'exécutant en 49

unités de temps (voir figure 17).



Algorithme Glouton Un autre Algorithme

Figure17: Comparaison de l'algorithme Glouton avec un autre algorithme.

3.2. GRAPHE 2-PAS

Forme générale de l'algorithme du graphe 2-pas

Pour k<-1 **à** n-1
  Pour i<-k+1 **à** n
    Exécuter T_{ki}
  Exécuter $T_{k+1,k+1}$.

$Ex(T_{ki}) = bt_k$ unités de temps $1 \le k \le n-1$, $k+1 \le i \le n$
$Ex(T_{kk}) = at_{k-1}$ unités de temps $2 \le k \le n$

où b est un entier et $t_k=k$ pour tout k ou $t_k=n-k$ pour tout k.
les contraintes d'ordonnacement sont:

(A) $T_{kk} << T_{ki}$ $1 \le k \le n-1$, $k+1 \le i \le n$
(B) $T_{ki} << T_{k+1,i}$ $1 \le k \le n-1$, $k+1 \le i \le n$

L'allure du graphe d'ordonnancement est présentée à la figure 18.

Le plus long chemin du graphe 2-pas est constitué par les tâches diagonales T_{kk} et les tâches $T_{k,k+1}$, d'où

$$T_{opt} = \sum_{k=1}^{n} at_{k-1} + \sum_{k=1}^{n-1} bt_k = (a+b)\frac{n^2}{2} + O(n)$$

Il est aisé de construire un algorithme parallèle s'exécutant en T_{opt} avec n-1 processeurs. L'efficacité asymptotique de cet algorithme est alors

$$e_{n-1,\infty} = \frac{2b}{3(a+b)}$$

T_{12}	T_{13}	...	...	$T_{1,n-1}$	$T_{1,n}$	niveau N_1
T_{22}						
	T_{23}	...	...	$T_{2,n-1}$	$T_{2,n}$	niveau N_2
		...	...			
			...	$T_{k,n-1}$	$T_{k,n}$	niveau N_k
					...	
					$T_{n-1,n}$	niveau N_n
					$T_{n,n}$	

Figure 18: L'allure du graphe 2-pas pour une matrice de taille nxn.

Nous définissons un algorithme qui possède trois phases. La première correspond à un algorithme de type Glouton dont l'efficacité asymptotique est égale à 1. Les p processeurs exécutent p tâches du graphe simultanément si ces dernières sont prêtes à être exécutées. L'algorithme exécute les tâches en balayant les lignes du graphe d'ordonnancement de gauche à droite, en commençant à tout instant le maximum de tâches exécutables. La différence entre la deuxième phase et la troisième phase réside dans le temps mis pour exécuter un niveau. Soit $s=\lceil a/b \rceil+1$ et q un entier tel que $q^+=\max(1,q)$. Pour simplifier la présentation, on considère l'existence d'un processeur supplémentaire P_0. Ce dernier est destiné à exécuter les tâches diagonales T_{kk}. Le nombre total de processeurs est toujours proportionnel à n. Cet hypothèse ne modifie pas l'évaluation asymptotique des performances.

Première phase

Au départ, l'algorithme exécute les niveaux compris entre N_1 et $N_{(n-sp)^+}$. Nous affectons les tâches T_{kk}, $2 \le k \le (n-sp)^+$ au processeur P_0. En se servant de l'algorithme Glouton, les processeurs $P_1, P_2, \ldots, P_p$ exécutent les tâches T_{ki} pour $i \ge k+1$. La différence entre cet algorithme et l'algorithme du paragraphe 3.1 réside dans la présence des tâches diagonales T_{kk} dont l'exécution est effectuée par le processeur P_0. Il commence l'exécution de la tâche $T_{k+1,k+1}$ juste après

avoir fini celle de la tâche $T_{k,k+1}$.

Pour avoir une efficacité asymptotique égale à 1 dans cette première phase, il faut que les p processeurs soient actifs pendant toute la durée de cette phase. Par conséquent, on doit avoir assez de tâches à exécuter par niveau. Soit t l'instant où nous commençons l'exécution de la tâche $T_{k,k+1}$. P_0 débute alors l'exécution de la tâche $T_{k+1,k+1}$ à l'instant $t+bt_k$ et la termine à l'instant $t+(a+b)t_k$. Les tâches d'un niveau donné s'exécutent de gauche à droite. Aucune tâche du niveau N_k n'est commencée à l'instant t-1. Ce qui entraîne que la fin de l'exécution d'au plus $\lceil (a+b)/b \rceil p=sp$ tâches du niveau N_k aura lieu à l'instant $t+(a+b)t_k$. Nous avons $k \geq (n-sp)^+$ par hypothèse, donc le niveau N_k comprend plus de sp tâches. Cela assure que l'exécution de la tâche $T_{k+1,k+1}$ par le processeur P_0 sera possible, pendant que les autres processeurs sont en train d'exécuter les tâches du niveau N_k. Les contraintes d'ordonnancement sont par conséquent respectées. Pendant l'exécution de cette première partie du graphe, tout processeur est actif. Pour évaluer le temps d'exécution de cette première phase, nous utilisons le fait que l'efficacité asymptotique de l'algorithme présenté est égal à 1. Le temps perdu par le processeur P_0 est négligeable ($p=\alpha n$ avec n tend vers l'infini). Le temps d'exécution est donc égal au temps d'exécution séquentielle de toutes les tâches de cette partie, divisé par le nombre de processurs p. Nous obtenons la valeur suivante:

$$\tau_1 = \left(\frac{b}{p}\right) \sum_{k=1}^{(n-sp)^+} (n-k)t_k + O(n)$$

Seconde phase

On considère maintenant la seconde phase de l'algorithme: du niveau $N_{(n-sp)^+}$ jusqu'au niveau $N_{(n-(s-1)p)^+}$. Dans cette deuxième phase, l'algorithme n'est pas d'efficacité 1. Avant de commencer, nous synchronisons les processeurs pour qu'ils débutent simultanément l'exécution des tâches suivantes. Cette synchronisation ne coûte qu'un facteur linéaire dans le temps total d'exécution parallèle, et est donc négligeable. L'algorithme procède niveau par niveau, mais le commencement d'un niveau quelconque n'a lieu qu'après la fin du niveau précédent. Les tâches d'un niveau donné sont exécutées de gauche à droite. En d'autres termes, l'exécution d'un niveau N_k possédant n-k tâches $T_{k,k+1}$, $T_{k,k+2}$, ..., $T_{k,n}$ s'effectue de la façon suivante: nous avons $n-k=(s-1)p+r_k$ tâches où r_k vérifie $0<r_k<p$. A l'instant t, P_1, P_2, ..., P_p commencent l'exécution des tâches $T_{k,k+1}$, $T_{k,k+2}$, ..., $T_{k,k+p}$. Dès qu'ils finissent celles-ci, ils commencent l'exécution des p tâches suivantes si s>2 ou les r_k dernières tâches si s=2 ($s=\lceil a/b \rceil+1 \geq 2$): c'est à dire à l'instant $t+bt_k$, P_1, P_2, ..., P_q débutent l'exécution des q tâches $T_{k,k+p+1}$, $T_{k,k+p+2}$, ..., $T_{k,k+p+q}$ avec q=p si s>2 et $q=r_k$ si s=2. Dit d'une autre façon, P_1,P_2, ..., P_p commencent l'exécution du nouveau bloc

contenant p tâches toutes les bt_k unités de temps. Ce processus se termine à l'instant $t+(s-1)t_k$ où nous commençons l'exécution des dernières r_k tâches. L'exécution du niveau N_k se termine à l'instant $t+sbt_k$. P_0 débute l'exécution de la tâche $T_{k+1,k+1}$ dès que possible, c'est à dire à l'instant $t+bt_k$ quand l'exécution de la tâche $T_{k,k+1}$ se termine. Par conséquent l'exécution de la tâche $T_{k+1,k+1}$ finit à l'instant $(t+bt_k)+at_k$. Puisque $s=1+\lceil a/b \rceil$ on a alors $(b+a)t_k \leq sbt_k$. Le processeur P_0 termine l'exécution sa tâche associée avant que les autres ne finissent le niveau N_k. Le temps mis pour exécuter cette phase est:

$$\tau_2 = \sum_{k=(n-sp+1)^+}^{(n-(s-1)p)^+} sbt_k + O(n)$$

Troisième phase

La dernière phase est consacrée à l'exécution des tâches des niveaux $N_{(n-(s-1)p)^+}$, $N_{(n-(s-1)p)^+ +1}$, ..., N_n. Les processeurs opèrent comme dans la phase 2. La seule différence est que les p processeurs P_1, P_2, ..., P_p complètent l'exécution du niveau donné N_k avant que P_0 ne termine l'exécution de la tâche $T_{k+1,k+1}$.Quand ce dernier finit celle-ci, nous débutons l'exécution du niveau N_{k+1}. Le temps mis pour exécuter cette phase est :

$$\tau_3 = \sum_{k=(n-(s-1)p)^+ +1}^{n} (a+b)t_k + O(n)$$

Nous remarquons que pour les valeurs de p telles que $p \geq n/(s-1)$, les deux premières phases de l'algorithme n'existent pas. Dans ce cas, Notre algorithme est alors asymptotiquement optimal. Le temps d'exécution correspond alors au plus long chemin du graphe d'ordonnancement. Une autre remarque dérive du fait que si a est multiple de b, c'est à dire a+b=sb, la seconde phase de l'algorithme s'exécute en temps optimal. Pour les valeurs de p telles que $p \geq n/s$, l'algorithme est donc asymptotiquement optimal.

Pour $t_k=k$ ou $t_k=n-k$, on a les résultats suivants avec $p=\alpha n$ ($\alpha \in]0,1]$) :

Pour	$\alpha \in]0,1/s[$	T_p	=	$\tau_1+\tau_2+\tau_3$
Pour	$\alpha \in [1/s,1/(s-1)[$	T_p	=	$\tau_2+\tau_3$
Pour	$\alpha \in [1/(s-1),1]$	T_p	=	τ_3 (s>2)

Proposition 2.2. Soit $s=\lceil a/b \rceil+1$

1. Le temps d'exécution T_p de l'algorithme décrit est donné par $T_p=\tau_1+\tau_2+\tau_3$
2. Si $p \geq n/(s-1)$, l'algorithme est asymptotiquement optimal, le temps mis pour l'exécuter est $T_p=T_{opt}=\tau_3$
3. Si $p \geq n/s$ et a/b est un entier, l'algorithme présenté est asymptotiquement optimal. Le temps mis pour l'exécuter est $T_p=T_{opt}=\tau_2+\tau_3$.

Pour mieux comprendre, nous considérons un exemple. Nous prenons a=1, b=2, s=2, p=3 et n=9. Nous rappelons que durant la phase 1, les processeurs opèrent de façon asynchrone avec une efficacité maximale. Dans la table suivante, ij dans la colonne k signifie que le processeur k effectue l'exécution de la tâche T_{ij}. La valeur b est le double de celle de a, le temps d'exécution de T_{ij} est deux fois plus grand que celui de T_{jj}. Pour faciliter la compréhension, nous avons scindé (conceptuellement) l'exécution des tâches T_{ij} en deux parties. L'exécution de cet exemple est détaillée à la figure 19.

P_0	P_1	P_2	P_3	
	12	13	14	Phase 1
	12	13	14	
22	15	16	17	
	15	16	17	
	18	19	23	
	18	19	23	
33	24	25	26	
	24	25	26	
	27	28	29	
	27	28	29	
	34	35	36	Phase 2
	34	35	36	
44	37	38	39	
	37	38	39	
	45	46	47	
	45	46	47	
55	48	49	-	
	48	49	-	
	56	57	58	
	56	57	58	
66	59	-	-	
	59	-	-	

	67	68	69	Phase 3
	67	68	69	
77	-	-	-	
	78	79	-	
	78	79	-	
88	-	-	-	
	89	-	-	
	89	-	-	

Figure 19 : Table d'exécution

Nous reviendrons sur l'étude du graphe 2-pas au chapitre suivant; en particulier, nous établirons des résultats de complexité.

3.3. DOUBLE GRAPHE

L'étude des algorithmes parallèles des graphes double triangulaire et double 2-pas se déduit des résutats précédents. L'idée essentielle est de rassembler deux niveaux successifs pour en former un seul. Nous nous ramenons alors à un graphe déjà étudié.

3.3.1. Double triangulaire (1)

Forme générale de l' Algorithme du graphe double triangulaire (1)

Pour k <- 1 à n
  Pour j <- k à n
    Exécuter T_{kj}
  Pour i <- k+1 à n
    Exécuter U_{ki}

$Ex(T_{kj})$ = bt_k unités de temps $1 \le k \le n$, $k \le j \le n$
$Ex(U_{ki})$ = ct_k unités de temps $1 \le k \le n$, $k+1 \le i \le n$

où b et c sont des entiers et $t_k=k$ pour tout k ou $t_k=n-k$ pour tout k.
Les contraintes d'ordonnancement sont:

(A)	T_{kj}	<<	$T_{k+1,j}$	$k+1 \le j \le n$, $2 \le k \le n-1$
(B)	T_{kk}	<<	$U_{k,j}$	$k+1 \le j \le n$, $1 \le k \le n-1$
(C)	U_{ki}	<<	$U_{k+1,i}$	$k+2 \le i \le n$, $1 \le k \le n-2$
(D)	$U_{k,k+1}$	<<	$T_{k+1,j}$	$k+1 \le j \le n$, $1 \le k \le n-1$

Dans un niveau quelconque N_k, il ya 2(n-k)+1tâches à exécuter. L'allure du graphe d'ordonnancement est présentée à la figure 20.

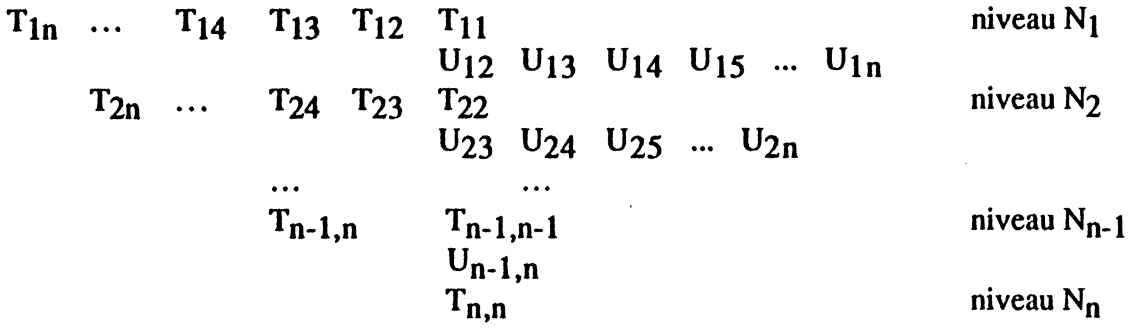


Figure 20 : L'allure du Graphe double triangulaire (1) pour une matrice de taille nxn.

Les tâches T_{kk} ($1 \le k \le n$) et $U_{k,k+1}$ ($1 \le k \le n-1$) forment le plus long chemin de ce type de graphe d'où

$$T_{opt} = \sum_{k=1}^{n} bt_k + \sum_{k=1}^{n-1} ct_k = (b+c)\frac{n^2}{2} + O(n)$$

Il est facile de construire un algorithme s'exécutant en T_{opt} avec n processeurs d'où l'efficacité asymptotique:

$$e_{n,\infty} = \quad 2/3$$

On remarque que cette efficacité asymptotique est la même pour ce type de graphe et le graphe triangulaire. En se servant de l'algorithme Glouton défini pour le graphe triangulaire, nous construisons un algorithme exécutant ce type de graphe. Pour un niveau N_k compris entre les niveaux N_1 et N_{n-p}, les tâches s'exécutent dans l'ordre suivant:

$$T_{kk}, T_{k,k+1}, \ldots, T_{kn}, U_{k,k+1}, U_{k,k+2}, \ldots, U_{kn}$$

Nous avons $n-k \ge p$ donc le début de l'exécution de la tâche $U_{k,k+1}$ aura lieu après la fin de celle de la tâche T_{kk}; cela assure que la contrainte (B) est respectée. Pour la même raison les contraintes (A), (C) et (D) sont aussi respectées. Le nombre des tâches T ou U situées sur un des niveaux N_{n-p+1}, N_{n-p+2}, ..., N_n est inférieur ou égal à p. Dans cette partie du graphe, l'algorithme procède niveau par niveau. L'exécution d'un niveau débute par l'exécution du premier sous niveau, formé par les tâches T, puis le second sous niveau, formé par les tâches U. Il n'est pas difficile de montrer que les contraintes d'ordonnancement sont respectées dans cette partie du graphe. Pour le calcul du temps d'exécution parallèle, nous prenons la longueur de la tâche R_{ki} $(c+b)t_k$ au lieu de $ct_{k-1}+bt_k$ pour tout $k \ge 1$. Le temps d'exécution total n'est

modifié qu'à un facteur linéaire près, ce qui n'affecte pas l'évaluation asymptotique. L'algorithme est d'efficacité 1 jusqu'au niveau possèdant p tâches: le temps parallèle mis pour exécuter cette première partie du graphe est égal à la somme des longueurs de toutes les tâches appartenant à cette partie divisée par le nombre de processeurs p. Pour la seconde partie du graphe, c'est à dire du niveau N_{n-p+1} jusqu'au niveau N_n, son temps d'exécution parallèle est égal à la longueur du plus long chemin. Soit T_p le temps mis pour exécuter l'algorithme.

(i) $t_k = k$ $\qquad T_p = (b+c)\, n^2 \frac{1+3\alpha^2-\alpha^3}{6\alpha} + O(n)$

(ii) $t_k = n-k$ $\qquad T_p = (b+c)\, n^2 \frac{2+\alpha^3}{6\alpha} + O(n)$

3.3.2. Double triangulaire (2)

Forme générale de l' Algorithme du graphe double triangulaire (2)

Pour k=2 à n
  Exécuter T_{kk}
  Pour j=k+1 à n
    Exécuter T_{jk}
  Pour i=k+1à n
    Exécuter U_{ik} .

$Ex(T_{jk}) = bt_k$ unités de temps $2\le k\le n,\ k+1\le j\le n$
$Ex(U_{ik}) = bt_k$ unités de temps $2\le k\le n,\ k+1\le i\le n$.

où b est un entier et $t_k=k$ pour tout k ou $t_k=n-k$ pour tout k.
Les contraintes de précédence sont:

(A)	T_{jk}	<<	$T_{j,k+1}$	$k+1\le j\le n,\ 2\le k\le n-1$
(B)	T_{kk}	<<	$U_{j,k+1}$	$k+2\le j\le n,\ 2\le k\le n-2$
(C)	U_{ik}	<<	$U_{i+1,k}$	$k+2\le i\le n,\ 2\le k\le n-2$
(D)	T_{kk}	<<	$T_{j,k+1}$	$k+1\le j\le n,\ 2\le k\le n-1$
(E)	$U_{k,k-1}$	<<	$T_{k,j}$	$k\le j\le n,\ 3\le k\le n-1$

On suppose que les tâches T_{i2} $(3\le i\le n)$ existent, cela ne modifie pas l'évaluation asymptotique. Dans un niveau quelconque N_k, il ya 2(n-k)+1 tâches à exécuter. L'allure du graphe d'ordonnancement est représentée par la figure 21.

Le plus long chemin est formé par les tâches $\{T_{22}, T_{33}, \ldots, T_{nn}\}$. En utilisant 2n-3 processeurs, il est facile d'exécuter ce graphe en minimum de temps $T_{opt}=bn^2/2$. L'efficacité asymptotique est donc égale à $e_{2n-3,\infty}=1/3$.

T_{2n} ...	T_{24}	T_{23}	T_{22}	U_{23}	U_{24}	U_{25}	... U_{2n}	niveau N_2
T_{3n} ...	...	T_{34}	T_{33}	U_{34}	...	...	U_{3n}	niveau N_3
	...			...				
	$T_{n-1,n}$		$T_{n-1,n-1}$	$U_{n-1,n}$				niveau N_{n-1}
			$T_{n,n}$					niveau N_n

Figure 21 : L'allure du Graphe double triangulaire (2) pour une matrice de taille nxn.

En se servant de l'algorithme Glouton défini pour le graphe triangulaire, nous construisons un algorithme exécutant ce type de graphe. Les tâches d'un niveau N_k compris entre les niveaux N_2 et $N_{n-\lceil p/2 \rceil}$, les tâches s'exécutent dans l'ordre suivant:

$$T_{kk}, U_{k,k+1}, U_{k,k+2}, \ldots, U_{kn}, T_{k,k+1}, \ldots, T_{kn}$$

Le fait d'avoir au moins p+1 tâches par niveau, assure que les contraintes sont respectées. Le nombre de tâches situées sur un des niveaux $N_{n-\lceil p/2 \rceil+1}$, $N_{n-\lceil p/2 \rceil+2}$, ..., N_n ne dépasse pas p+1 tâches. Dans cette partie du graphe, l'algorithme procède niveau par niveau. On n'aborde un niveau qu'après la fin de l'exécution du niveau précédent. Il n'est pas difficile de montrer que les contraintes sont respectées dans cette partie du graphe. L'algorithme est d'efficacité 1 jusqu'au niveau $N_{n-\lceil p/2 \rceil}$: le temps parallèle de cette première partie du graphe est égal à la somme des longueurs de toutes les tâches appartenant à cette partie divisée par le nombre de processeurs p. Pour la seconde partie du graphe, c'est à dire du niveau $N_{n-\lceil p/2 \rceil+1}$ jusqu'au niveau N_n, son temps d'exécution parallèle est égal à la longueur du plus long chemin. Soit T_p le temps mis pour exécuter l'algorithme.

(i) $t_k = k$. $T_p = bn^2 \frac{8+6\alpha^2-\alpha^3}{24\alpha} + O(n)$

(ii) $t_k = n-k$ $T_p = bn^2 \frac{16+\alpha^3}{24\alpha} + O(n)$

3.3.3. Double 2-pas

Forme générale de l'algorithme du graphe double 2-pas

```
Pour k <- 1 à n-1
    Pour j <- k+1 à n
        Exécuter Tkj
    Pour i <- k+1 à n
        Exécuter Uik
    Exécuter Sk+1,k+1
```

$Ex(T_{kj}) = bt_k$ unités de temps $1 \le k \le n-1,\ k+1 \le j \le n$
$Ex(U_{ik}) = bt_k$ unités de temps $1 \le k \le n-1,\ k+1 \le i \le n$
$Ex(S_{kk}) = at_{k-1}$ unités de temps $2 \le k \le n$

où a et b sont des entiers et $t_k=k$ pour tout k ou $t_k=n-k$ pour tout k. Les contraintes d'ordonnancement sont les suivantes:

(A)	S_{kk}	<<	T_{kj}	$k+1 \le j \le n,\ 2 \le k \le n-1$
(B)	S_{kk}	<<	U_{ik}	$k+1 \le i \le n,\ 2 \le k \le n-1$
(C)	$T_{k,k+1}$	<<	$S_{k+1,k+1}$	$1 \le k \le n-1$
(D)	T_{kj}	<<	$T_{k+1,j}$	$k+2 \le j \le n,\ 1 \le k \le n-2$
(E)	$U_{k+1,k}$	<<	$S_{k+1,k+1}$	$1 \le k \le n-1$
(F)	U_{ik}	<<	$U_{i,k+1}$	$k+2 \le i \le n,\ 1 \le k \le n-2$

Le plus long chemin est formé par les tâches diagonales $S_{k+1,k+1}$ et les tâches $T_{k,k+1}$ $(1 \le k \le n-1)$ d'où

$$T_{opt} = \sum_{k=1}^{n-1} (a+b)t_k = (a+b)\frac{n^2}{2} + O(n)$$

Il est simple de construire un algorithme s'exécutant en T_{opt} avec n processeurs, d'où l'efficacité asymptotique est:

$$e_{n,\infty} = \frac{2b}{3(a+b)}$$

On remarque que cette efficacité asymptotique est égale à celle du graphe 2-pas. L'allure du graphe d'ordonnancement est donnée par la figure 22.

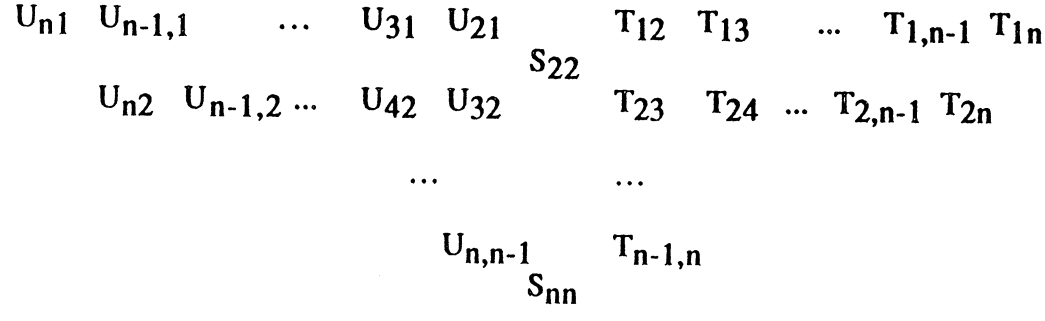


Figure 22 : Allure du graphe double 2-pas pour une matrice de taille nxn.

L'algorithme que nous proposons possède trois phases, comme l'algorithme du graphe 2-pas. Le nombre de tâches non diagonales est multiplié par 2 dans ce double graphe. Soit p un entier pair. La phase 1 s'effectue d'une façon asynchrone du niveau N_1 au niveau $N_{(n-sp/2)^+}$. Les phases 2 et 3 s'effectuent

d'une façon synchrone. Elles correspondent respectivement à l'exécution des niveaux $N_{(n-sp/2)^+}+1, \ldots, N_{(n-(s-1)p/2)^+}$ et des niveaux $N_{(n-(s-1)p/2)^+}+1, \ldots, N_n$.

Les temps mis pour exécuter les trois phases sont les suivants:

$$\tau_1 = \left(\frac{1}{p}\right) \sum_{k=1}^{(n-\frac{sp}{2})^+} 2(n-k)bt_k + O(n)$$

$$\tau_2 = \sum_{k=(n-\frac{sp}{2})^+ + 1}^{(n-\frac{(s-1)p}{2})^+} sbt_k + O(n)$$

$$\tau_3 = \sum_{k=(n-\frac{(s-1)p}{2})^+ + 1}^{n} (a+b)t_k + O(n)$$

Le temps d'exécution parallèle de l'algorithme peut être compté comme dans le paragraphe 3.2. Etant donné $p=\alpha n$, on a alors

Pour $\alpha \in]0,2/s[$	T_p =	$\tau_1+\tau_2+\tau_3$
Pour $\alpha \in [2/s,2/(s-1)[$	T_p =	$\tau_2+\tau_3$
Pour $\alpha \in [2/(s-1),1[$	T_p =	τ_3

4. PERFORMANCES

Dans cette section, nous donnons le temps mis pour exécuter chacun des algorithmes parallèles proposés dans le paragraphe 3. Ensuite, nous comparons leurs performances.

4.1. CALCUL DES TEMPS D'EXECUTION

L'analyse précédente conduit aux résultats suivants où nous présentons le temps mis pour l'exécution parallèle avec $p=\alpha n$ processeurs $\alpha \in]0,1]$. Puisque notre étude est effectuée pour n grand nous supprimons les termes linéaires. Nous définissons aussi l'efficacité asymptotique:

$$e_\alpha = \frac{T_1.}{\alpha n.T_{\alpha n}}$$

où T_1 est le temps mis pour l'exécution de l'algorithme parallèle par un seul processeurs et $T_{\alpha n}$ est le temps mis pour l'exécution parallèle avec $p = \alpha n$ processeurs.

(A) Elimination de Gauss: Forme KJI-SAXPY

Le graphe d'ordonnancement est un graphe 2-pas avec a=1, b=2, s=2 et t_k=n-k.

Pour α=1, l'exécution s'effectue en temps asymptotiquement optimal.

Pour $\alpha \in]0,\frac{1}{2}]$ $\quad T_{\alpha n}=\frac{4+13\alpha^3}{6\alpha}n^2 + O(n)$ $\quad e_\alpha=\frac{4}{4+13\alpha^3}$

Pour $\alpha \in [\frac{1}{2},1]$ $\quad T_{\alpha n}=\frac{4-\alpha^2}{2}n^2 + O(n)$ $\quad e_\alpha=\frac{4}{12\alpha-13\alpha^3}$

(B) Elimination de Gauss: Forme JKI-GAXPY

Le graphe et les contraintes d'ordonnancement sont identiques à ceux de (A). L'évaluation des performances est donc la même.

(C) Elimination de Gauss: Forme IJK-DOT

Le graphe d'ordonnancement est un graphe double triangulaire (1) avec b=c=2 et t_k=k.

Pour $\alpha \in]0,1]$ $\quad T_{\alpha n}=\frac{2+6\alpha^2-2\alpha^3}{3\alpha}n^2 + O(n)$ $\quad e_\alpha=\frac{1}{1+3\alpha^2-\alpha^3}$

(D) Elimination de Gauss: Forme IJK-DOT modifiée

Le graphe d'ordonnancement est un graphe double triangulaire (2) avec b=2 et t_k=k.

Pour $\alpha \in]0,1]$ $\quad T_{\alpha n}=\frac{8+6\alpha^2-\alpha^3}{12\alpha}n^2 + O(n)$ $\quad e_\alpha=\frac{8}{8+6\alpha^2-\alpha^3}$

(E) Elimination de Gauss: Forme KJI-SAXPY modifiée

Le graphe d'ordonnancement est un graphe triangulaire avec b=2 et t_k=n-k. L'algorithme décrit pour ce type de graphe s'exécute en temps asymptotiquement optimal.

Pour $\alpha \in]0,1]$ $\quad T_{\alpha n}=\frac{2+\alpha^3}{3\alpha}n^2 + O(n)$ $\quad e_\alpha=\frac{2}{2+\alpha^3}$

(F) Réduction de Doolittle (1)

Le graphe et les contraintes d'ordonnancement sont identiques à ceux donnés par (C).

(G) Réduction de Doolittle(2)

Le graphe et les contraintes d'ordonnancement sont identiques à ceux donnés par (D).

(H) Décomposition LDM^t: code non optimisé (1)

Le graphe d'ordonnancement est un graphe double 2-pas avec a=3, b=c=3, s=2 et t_k=k.

Pour $\alpha \in]0,1]$ $\quad T_{\alpha n} = \frac{1+3\alpha^2-\alpha^3}{\alpha} n^2 + O(n)$ $\quad e_\alpha = \frac{1}{1+3\alpha^2-\alpha^3}$

(I) Décomposition LDM^t: code non optimisé (2)

Le graphe d'ordonnancement est un graphe triangulaire (2) avec b=6 et t_k=k.

Pour $\alpha \in]0,1]$ $\quad T_{\alpha n} = \frac{1+3\alpha^2-\alpha^3}{\alpha} n^2 + O(n)$ $\quad e_\alpha = \frac{1}{1+3\alpha^2-\alpha^3}$

(J) Décomposition LDM^t: code optimisé (1)

Le graphe d'ordonnancement est un graphe double 2-pas avec a=4, b=c=2, s=3 et t_k=k.

Pour $\alpha \in]0,\frac{2}{3}]$ $\quad T_{\alpha n} = \frac{8+54\alpha^2-27\alpha^3}{12\alpha} n^2 + O(n)$ $\quad e_\alpha = \frac{8}{8+54\alpha^2-27\alpha^3}$

Pour $\alpha \in [\frac{2}{3},1]$ $\quad T_{\alpha n} = 3n^2 + O(n)$ $\quad e_\alpha = \frac{2}{9\alpha}$

(K) Décomposition LDM^t: code optimisé (2)

Le graphe d'ordonnancement est un graphe triangulaire (2) avec b=4 et t_k=k.

Pour $\alpha \in]0,1]$ $\quad T_{\alpha n} = \frac{2+6\alpha^2-2\alpha^3}{3\alpha} n^2 + O(n)$ $\quad e_\alpha = \frac{1}{1+3\alpha^2-\alpha^3}$

(L) Décomposition LDL^t: code non optimisé (1)

Le graphe d'ordonnancement est un graphe 2-pas avec a=b=3, s=2 et t_k=k.

Pour $\alpha \in]0,\frac{1}{2}]$ $\quad T_{\alpha n}=\frac{1+12\alpha^2-8\alpha^3}{2\alpha}n^2+O(n)$ $\quad e_\alpha=\frac{1}{1+12\alpha^2-8\alpha^3}$

Pour $\alpha \in [\frac{1}{2},1]$ $\quad T_{\alpha n}=3n^2+O(n)$ $\quad e_\alpha=\frac{1}{6\alpha}$

(M) Décomposition LDL^t: code non optimisé (2)

Le graphe d'ordonnancement est un graphe triangulaire (2) avec b=3 et t_k=k.

Pour $\alpha \in]0,1]$ $\quad T_{\alpha n}=\frac{1+3\alpha^2-\alpha^3}{2\alpha}n^2+O(n)$ $\quad e_\alpha=\frac{1}{1+3\alpha^2-\alpha^3}$

(N) Décomposition LDL^t: code optimisé (1)

Le graphe d'ordonnancement est un graphe 2-pas avec a=3, b=2, s=3 et t_k=k.

Pour $\alpha \in]0,\frac{1}{3}]$ $\quad T_{\alpha n}=\frac{1+21\alpha^2-21\alpha^3}{3\alpha}n^2+O(n)$ $\quad e_\alpha=\frac{1}{1+21\alpha^2-21\alpha^3}$

Pour $\alpha \in [\frac{1}{3},\frac{1}{2}]$ $\quad T_{\alpha n}=(3-2\alpha+2\alpha^2)n^2+O(n)$ $\quad e_\alpha=\frac{1}{(9-6\alpha+6\alpha^2)\alpha}$

Pour $\alpha \in [\frac{1}{2},1]$ $\quad T_{\alpha n}=\frac{5}{2}n^2+O(n)$ $\quad e_\alpha=\frac{2}{15\alpha}$

(O) Décomposition LDL^t: code optimisé (2)

Le graphe d'ordonnancement est un graphe triangulaire (2) avec b=2 et t_k=k. La première tâche de chaque niveau N_k de longueur ak avec a=3, s=2.

Pour $\alpha \in]0,\frac{1}{2}]$ $\quad T_{\alpha n}=\frac{2+18\alpha^2-13\alpha^3}{6\alpha}n^2+O(n)$ $\quad e_\alpha=\frac{2}{2+18\alpha^2-13\alpha^3}$

Pour $\alpha \in [\frac{1}{2},1]$ $\quad T_{\alpha n}=\frac{4-2\alpha+\alpha^2}{2}n^2+O(n)$ $\quad e_\alpha=\frac{2}{12\alpha-6\alpha^2+3\alpha^3}$

(P) Décomposition de Cholesky (1): GG^t

Le graphe d'ordonnancement est un graphe 2-pas avec a=b=2, s=2 et t_k=k.

Pour $\alpha \in]0,\frac{1}{2}]$ $\quad T_{\alpha n}=\frac{1+12\alpha^2-8\alpha^3}{3\alpha}n^2+O(n)$ $\quad e_\alpha=\frac{1}{1+12\alpha^2-8\alpha^3}$

Pour $\alpha \in [\frac{1}{2},1]$ $\quad T_{\alpha n}=2n^2+O(n)$ $\quad e_\alpha=\frac{1}{6\alpha}$

(Q) Décomposition de Cholesky (2): GG^t

Le graphe d'ordonnancement est un graphe triangulaire (2) avec b=2 et $t_k=k$.

Pour $\alpha \in]0,1]$ $\quad T_{\alpha n} = \frac{1+3\alpha^2-\alpha^3}{3\alpha} n^2 + O(n) \quad e_\alpha = \frac{1}{1+3\alpha^2-\alpha^3}$

4.2. COMPARAISON

Dans cette partie, $T_p(X)$ désigne le temps asymptotique d'exécution parallèle de l'algorithme que nous avons proposé pour la version (X). On définit aussi l'efficacité relative par

$$E_\alpha = \frac{\min(T_1(X), X \text{ est une version parallèle})}{pT_p(X)}$$

Pour une même transformation mathématique, on peut avoir plusieurs algorithmes séquentiels et parallèles. Pour pouvoir comparer les efficacités de deux versions parallèles, dans le cas où ces dernières n'ont pas le même coût arithmétique, on se sert de l'efficacité relative E_α . Elle donne une idée plus exacte que e_α.

4.2.1. Comparaison des versions de Cholesky

Les deux versions ont le même temps d'exécution séquentiel donc $E_\alpha=e_\alpha$. Les représentations graphiques des efficacités e_α (figure 23) par rapport à α montrent que la version de Cholesky (2) est meilleure à paralléliser que la version (1).

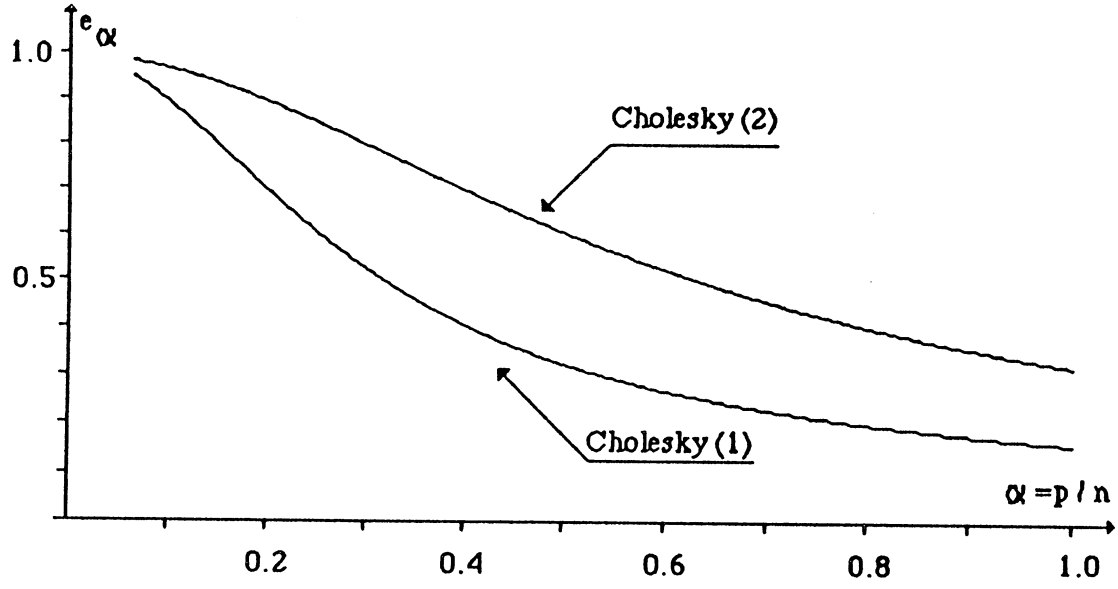


Figure 23: Représentations graphiques des efficacités des versions de Cholesky.

4.2.2. Comparaison des versions LDL^t

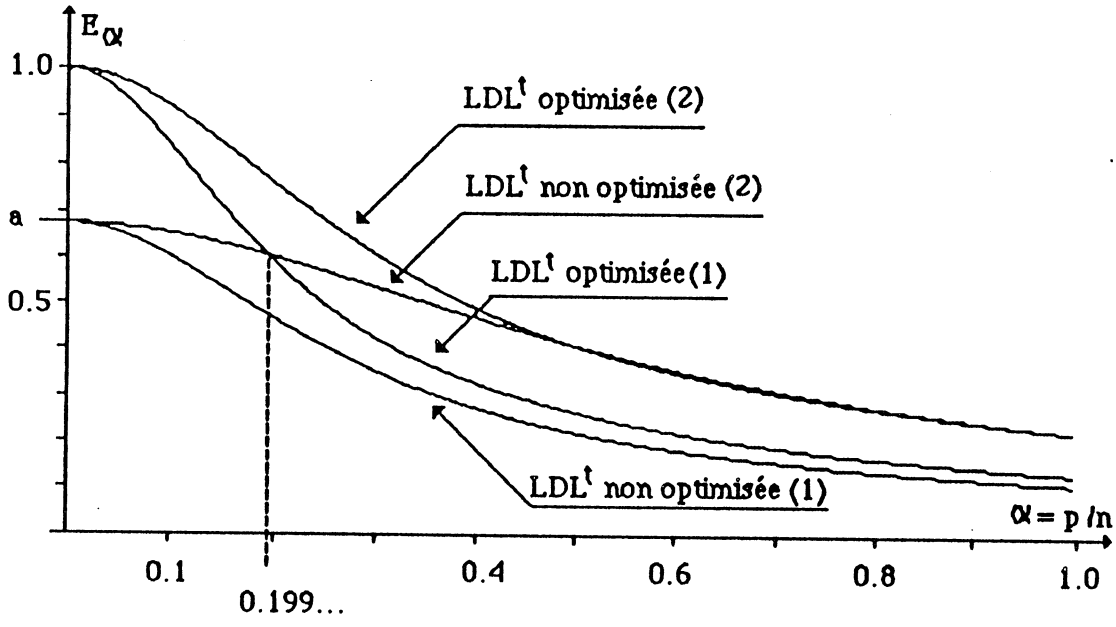


Figure 24: Représentations graphiques des efficacités relatives des versions LDL^t.

On a quatre versions dont deux non optimisées. Le temps d'exécution séquentiel est supérieur à celui des versions optimisées. Pour les versions optimisées on a $e_\alpha = E_\alpha$, pour les autres, $E_\alpha = 2e_\alpha/3$. La version LDL^t optimisée (2) est la plus performante (figure 24). La version LDL^t non optimisée (1) est toujours la moins bonne. Les deux autres versions, LDL^t optimisée (1) et non optimisée (2) ont des performances intermédiaires. Pour $\alpha \in]0,b]$, ($b \in]0,1/3]$ est la solution de l'équation $1-33x^2+39x^3=0$, b= 0.199...), la version optimisée (1)

est meilleure que la version non optimisée (2), tandis que sur l'intervalle [b,1] c'est l'inverse: les performances de la version non optimisée (2) deviennent meilleures que celles de l'autre version. La valeur a est égale à 2/3. Remarquons que les algorithmes parallèles de Cholesky (1) et (2) ont respectivement les mêmes efficacités que les algorithmes parallèles de LDL^t non optimisée (1) et (2).

4.2.3. Comparaison des versions LDM^t

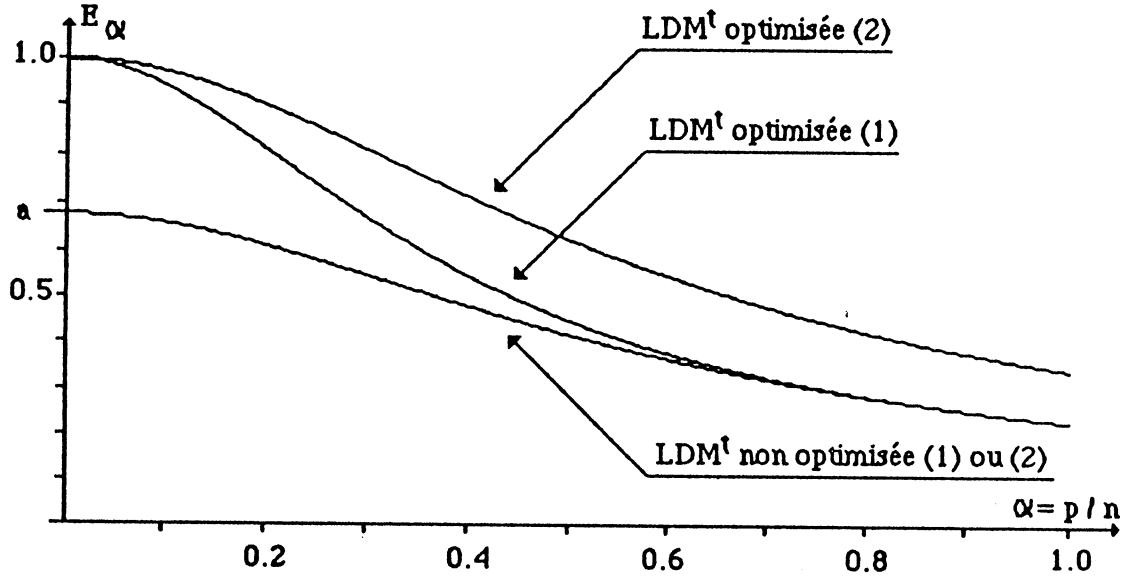


Figure 25: Représentations graphiques des efficacités relatives des versions LDM^t.

Pour les versions optimisées on a $e_\alpha=E_\alpha$, pour les autres $E_\alpha=2e_\alpha/3$. La figure 25, nous montre que la version LDM^t optimisée (2) est la meilleure. La version LDM^t optimisée (1) est assez efficace par rapport aux deux versions non optimisées qui ont les mêmes performances. La valeur a est égale à 2/3.

4.2.4. Comparaison des versions de Gauss

Au début, remarquons que le temps mis pour exécuter chacun des algorithmes parallèles est le même pour toutes les versions. Les versions à comparer sont les suivantes:

(A) forme KJI - SAXPY
(B) forme JKI - GAXPY
(C) forme IJK - DOT
(D) forme IJK - DOT modifiée
(E) forme KJI - SAXPY modifiée
(F) réduction de Doolittle (1)
(G) réduction de Doolittle (2)

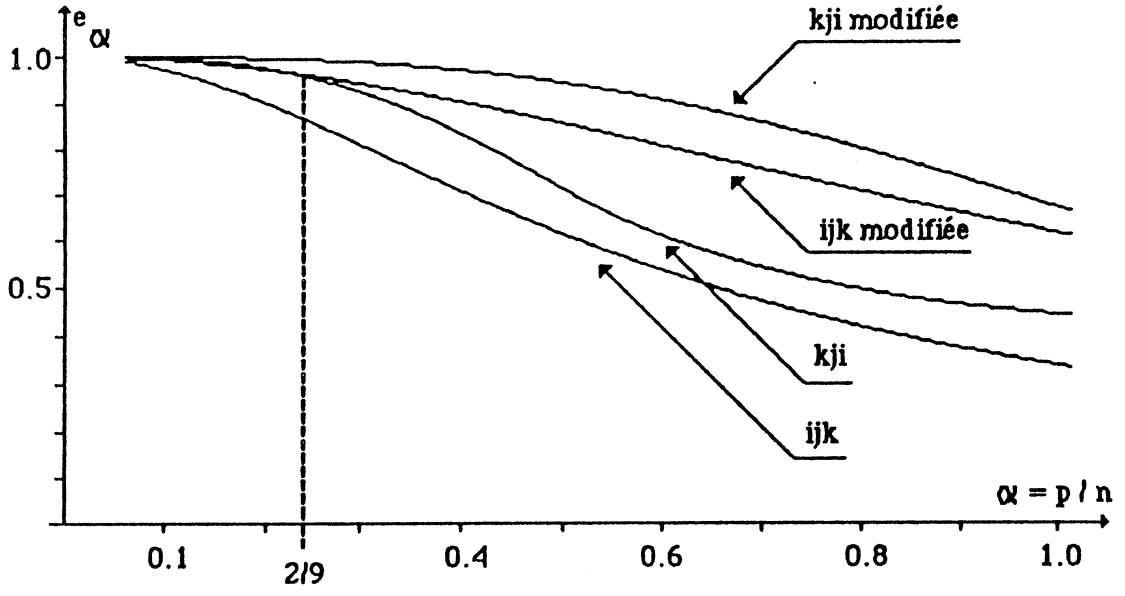


Figure 26: Représentations graphiques des efficacités des versions de l'algorihme de Gauss.

Nous savons déjà que $T_p(A) = T_p(B)$, $T_p(C) = T_p(F)$ et $T_p(D)=T_p(G)$.
Parmi les sept versions, la forme modifiée KJI-SAXPY qui correspond au graphe triangulaire possède les meilleures performances (figure 26). L'algorithme parallèle défini pour les versions IJK et réduction de Doolittle (1) possède des performances moins bonnes que celles des autres algorithmes. Sur l'intervalle]0,2/9], l'algorithme des formes KJI et JKI est meilleur que celui des formes IJK modifiée et réduction de Doolittle (2), tandis que sur l'intervalle [2/9,1], le dernier algorithme possède des performances meilleures que celles des formes KJI et JKI. Remarquons que l'algorithme de la décomposition LDM^t optimisée (2) possède la même fonction efficacité e_α que l'algoritme des versions IJK et réduction de Doolittle (1).

5. CONCLUSION ET REMARQUES

Nous avons présenté dix-sept versions parallèles d'algorithmes de triangularisation pour la résolution d'un système linéaire dense. Nous avons utilisé le formalisme du graphe des tâches pour analyser ces versions et par la suite comparer leurs performances.

Les graphes d'ordonnancement de ces dix-sept versions se réduisent à six classes de graphes. En utilisant p processeurs $p = \alpha n$ où $\alpha \leq 1$, nous avons construit pour chaque classe un algorithme parallèle, et nous avons calculé son temps d'exécution asymptotique.

Il est clair que l'implémentation parallèle conduit à des résultats différents de l'étude des versions vectorielles. Les deux versions KJI - SAXPY et JKI-GAXPY de [15] possèdent le même graphe d'ordonnancement, par

conséquent les mêmes algorithmes parallèles. Contrairement à l'implémenttation vectorielle où la forme JKI-GAXPY [15] apparaît la meilleure à pipeliner. La forme KJI-SAXPY modifiée est la meilleure à paralléliser sur une machine MIMD.

Notre étude a été consacrée à la triangularisation des algorithmes sans pivotage [42]. Il est possible de traiter la triangularisation d'une matrice avec pivotage partiel [37]. Pour cela, il suffit de prendre les calculs pour a=b=2 avec un graphe 2-pas pour obtenir les complexités de l'élimination de Gauss avec pivotage partiel.

Chapitre III

GRAPHE 2-PAS

1. INTRODUCTION

Dans ce chapitre, on étudie le graphe 2-pas. Comme on l'a vu dans le chapitre II, c'est le graphe de plusieurs algorithmes d'algèbre linéaire (décomposition LU, décomposition LDL^t, décomposition Cholesky, inversion d'une matrice triangulaire, ...). La difficulté de construire un algorithme optimal pour ce type de graphe réside dans le fait qu'un niveau quelconque possède deux sous-niveaux. Il n'est pas facile de trouver un algorithme optimal dans le cas général. Conformément à l'usage en matière d'analyse de complexité pour l'élimination de Gauss avec pivotage partiel [37, 55, 68], nous supposons, pour évaluer le temps parallèle et séquentiel de cet algorithme, que chaque processeur peut réaliser en une unité de temps une multiplication suivie d'une soustraction, ou une comparaison suivie d'une multiplication, ou encore une division. Dans le cas particulier où t_k=n-k, on améliore l'algorithme construit au chapitre II. On présente aussi des algorithmes parallèles pour le graphe 2-pas dont l'un est asymptotiquement optimal, après avoir donné la forme générale du graphe 2-pas et les valeurs limites de α. Des résultats de simulation sont présentés à la section 6.

2. BORNE SUR α_0

Forme générale de l'algorithme du graphe 2-pas

Pour k<-1 **à** n-1
  Pour i<-k+1 **à** n
    Exécuter T_{ki}
  Exécuter $T_{k+1,k+1}$.

$Ex(T_{ki}) = bt_k$ unités de temps $1 \le k \le n-1$, $k+1 \le i \le n$
$Ex(T_{kk}) = at_{k-1}$ unités de temps $2 \le k \le n$

où a et b sont des entiers et t_k=k pour tout k ou t_k=n-k pour tout k. Les contraintes d'ordonnancement sont:

(A) $T_{kk} << T_{ki}$ $1 \le k \le n-1$, $k+1 \le i \le n$
(B) $T_{ki} << T_{k+1,i}$ $1 \le k \le n-1$, $k+1 \le i \le n$

La figure 1 présente l'allure du graphe d'ordonnancement.

Le plus long chemin du graphe 2-pas est constitué par les tâches diagonales T_{kk} et les tâches $T_{k,k+1}$, d'où dans les deux cas (t_k=k ou t_k=n-k) on a

$$T_{opt} = \sum_{k=1}^{n} at_{k-1} + \sum_{k=1}^{n-1} bt_k = (a+b)\frac{n^2}{2} + O(n)$$

Il est aisé de construire un algorithme parallèle s'exécutant en T_{opt} avec n-1 processeurs. L'efficacité asymptotique de cet algorithme est

$$e_{n-1,\infty} = \frac{2b}{3(a+b)}$$

T_{12}	T_{13}	...	...	$T_{1,n-1}$	$T_{1,n}$	niveau N_1
T_{22}						
	T_{23}	...	...	$T_{2,n-1}$	$T_{2,n}$	niveau N_2
		...	...			
			...	$T_{k,n-1}$	$T_{k,n}$	niveau N_k
					...	
					$T_{n-1,n}$	niveau N_n
					$T_{n,n}$	

Figure 1: L'allure du graphe 2-pas pour une matrice de taille nxn.

Soit le problème (P):
Chercher la valeur minimale α_0 de α pour laquelle l'exécution du graphe 2-pas peut s'effectuer en un temps T_{opt}?
Notons $T_{opt,p}$ le temps d'un algorithme optimal avec $p=\alpha n$ processeurs. On cherche α_0 tel que

$$\alpha_0 = \min(\alpha \text{ tel que } 0 \le \alpha \le 1, \lim T_{opt,p} / T_{opt} = 1 \text{ quand n tend vers } \infty)$$

Informellement, $\alpha_0 n$ est le nombre minimal de processeurs pour pouvoir exécuter l'algorithme dans un temps égal à la longueur du plus long chemin du graphe (cette longueur est une borne universelle pour tout algorithme).

Lord, Kowalik et Kumar [37] ont proposé une borne inférieure α_0 dans le cas particulier a=b=1 et t_k=n-k. Dans ce qui suit, on retrouve cette borne dans le cas général.

Quand on commence l'exécution de la première tâche $T_{n-1,n}$ du niveau N_{n-1}, au plus un seul processeur peut être actif.
Quand on commence l'exécution de la première tâche $T_{n-2,n-1}$ du niveau N_{n-2}, au maximum deux processeurs peuvent être actifs.
D'une façon générale, quand on aborde l'exécution de la première tâche $T_{k,k+1}$ du niveau N_k, au maximum n-k processeurs fonctionnent: n-k est le nombre de tâches situées sur le niveau N_k (voir figure 2).
Supposons que tous les processeurs soient actifs jusqu'au début de l'exécution du niveau N_{n-p+1}: le nombre de tâches d'un niveau situé au dessus du niveau N_{n-p+1}

est supérieur ou égal à p ce qui permet éventuellement le fonctionnement simultané des p processeurs. Au début de l'exécution de chacun des niveaux N_{n-p+1}, N_{n-p+2}, ..., N_{n-1}, un processeur doit se libérer. Le nombre de processeurs libérés au début du niveau N_{n-p+k} $(1 \leq k \leq p-1)$ est k.

Soit TIA la somme total des temps libres des processeurs (total idle area), dans notre cas, nous cherchons le temps minimal d'inactivité des processeurs. Soit CA la somme des temps des tâches à exécuter diminuée de TIA (computational area). Autrement dit, c'est le produit du temps T_p par le nombre de processeurs p utilisés pour l'exécution de l'algorithme parallèle moins la somme total des temps libres des processeurs [37].

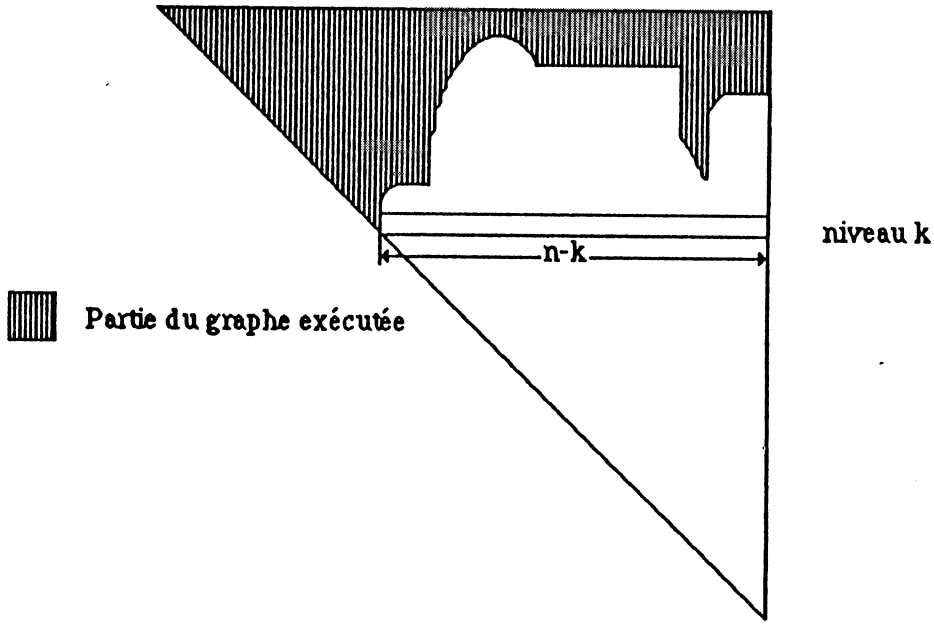


Figure 2: Commencement du niveau N_k.

Dans les deux cas, le temps libre total obligatoire des processesurs est:

(i) $t_k = n-k$ $\qquad TIA = \frac{a+b}{6}p^3 + O(n^2)$

(ii) $t_k = k$ $\qquad TIA = \frac{a+b}{2}np^2 - \frac{a+b}{6}p^3 + O(n^2)$

Ce qui entraîne alors respectivement les égalités suivantes:

$$(i)\ CA = \frac{a+b}{2}p(n^2 - \frac{p^2}{3}) + O(n^2)$$

$$(ii)\ CA = \frac{a+b}{2}p(n^2 - np + p^2) + O(n^2)$$

Le temps T_1 mis pour exécuter le graphe par un seul processeur est la somme totale de toutes les tâches du graphe. Il est égal à:

(i) $t_k = n - k$ $\qquad TW = b\frac{n^3}{3} + O(n^2)$

(ii) $t_k = k$ $\qquad TW = b\frac{n^3}{6} + O(n^2)$

Quand n tend vers l'infini, le quotient CA / TW doit nécessairement tendre vers 1. Pour $p=\alpha n$ $\alpha \in]0,1]$, on a donc les équations suivantes:

(i) $t_k=n-k$ $\qquad 3\alpha - \alpha^3 = \frac{2b}{a+b}$

(ii) $t_k=k$ $\qquad (\alpha - 1)^3 = \frac{b}{a+b} - 1$

Une solution de chacune des deux équations ci-dessus appartient à l'intervalle]0,1]. On la note α_0. C'est la borne inférieure de α pour laquelle l'exécution du graphe 2-pas peut s'effectuer en temps T_{opt}.

3. DECOMPOSITION LU

Soit A une matrice carrée dense de taille n. On rappelle l'algorithme de la méthode d'élimination de Gauss avec pivotage partiel [37, 66]:

pour k := 1 **à** n-1 **faire**
exécuter T_{kk}:
< trouver p tel que $|a_{pk}| = \max \{ a_{kk}, ..., a_{nk} \}$
piv(k) := p (* choix de la ligne p comme ligne pivot *)
échanger $a_{piv(k),k}$ et a_{kk}
c := 1 / a_{kk}
pour i := k+1 **à** n **faire**
$a_{ik} := a_{ik} * c$ >
pour j := k+1 **à** n **faire**
exécuter T_{kj}:
< échanger $a_{piv(k),j}$ et a_{kj}
pour i := k+1 **à** n **faire**
$a_{ij} := a_{ij} - a_{ik} * a_{kj}$ >

Les contraintes d'ordonnancement sont :

(A) $T_{kk} << T_{kj}$ $\qquad k+1 \le j \le n, 1 \le k \le n-1$

(B) $T_{kj} << T_{k+1,j}$ $\qquad k+1 \le j \le n, 1 \le k \le n-1.$

La contrainte (A) est dûe à la préparation de la colonne k à l'étape k, son exécution doit se terminer avant le commencement des modifications des colonnes j>k. L'exécution de la colonne j à l'étape k (j>k) doit se terminer avant que l'exécution de la même colonne ne commence à l'étape k+1: c'est la contrainte (B). Les temps d'exécution sont donnés par $Ex(T_{kk}) = n+1-k$ et $Ex(T_{kj}) = n-k$ si j>k (une unité de temps est une multiplication suivie d'une soustraction ou une comparaison suivie d'une multiplication). Le temps d'exécution séquentiel est $T_1 = n^3/3 + O(n^2)$. Le temps d'exécution d'un algorithme avec $p=\alpha n$ processeurs sera noté $T_p = f(\alpha)n^2 + O(n)$, d'où une efficacité asymptotique $e_\alpha = 1/(3\alpha f(\alpha))$.

L'allure du graphe d'ordonnancement est la suivante :

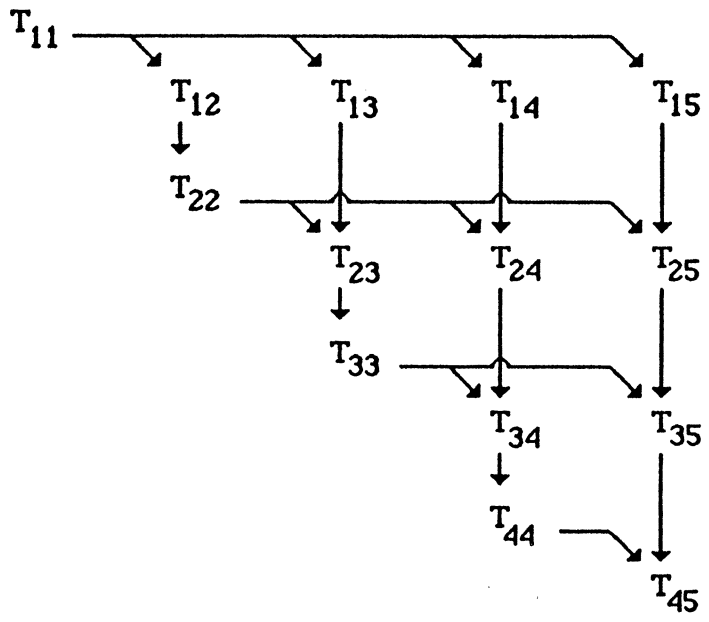


Figure 3 : L'allure du graphe d'ordonnancement pour n=5.

Le plus long chemin s_1 du graphe est constitué des tâches T_{11}, T_{12}, T_{22}, ..., T_{kk}, $T_{k,k+1}$, ..., $T_{n-1,n-1}$, $T_{n-1,n}$. Sa longueur est $T_{opt} = L(s_1) = n^2-1$. Il est facile de construire un algorithme s'exécutant en temps T_{opt} (nécessairement asymptotiquement optimal) si l'on dispose de $\lceil n/2 \rceil$ processeurs. Pour $\alpha \geq 1/2$, l'efficacité asymptotique est alors $e_\alpha = 1/(3\alpha) \leq 2/3$.

3.1. UN PREMIER ALGORITHME PARALLELE

Nous décrivons un algorithme s'exécutant avec $p=\alpha n$ processeurs, où $\alpha \leq 1/3$. Nous traitons tout d'abord le cas où $n=3p+1$ avant d'aborder le cas général. Soit donc $n=3p+1$, désignons par P_1, ..., P_p les p processeurs. Nous partageons le graphe d'ordonnancement en trois parties comme indiqué figure 4. La partie 1 est divisée en p blocs B_0, ..., B_{p-1}. Le bloc B_k est composé des tâches non-diagonales $T_{2k+i,j}$, $1 \leq i \leq 2$ et $2k+i<j \leq 2p+1$, et des deux tâches diagonales $T_{2k+2,2k+2}$ et $T_{2k+3,2k+3}$. La partie 2 se compose des tâches T_{kj}, $1 \leq k \leq 2p$ et $2p+2<j \leq 3p+1$. Enfin, la partie 3 est constituée du bloc triangulaire inférieur $\{T_{kj},\ 2p<k \leq 3p,\ k<j \leq 3p+1\} \cup \{T_{kk},\ 2p+2 \leq k \leq 3p\}$. Nous négligeons le temps d'exécution de la tâche T_{11} et supposons que tous les processeurs commencent l'exécution du bloc B_0 à l'instant t=0; ceci n'affecte le temps d'exécution global T_p de l'algorithme que par l'omission d'un facteur linéaire et ne modifie donc pas l'évaluation asymptotique.

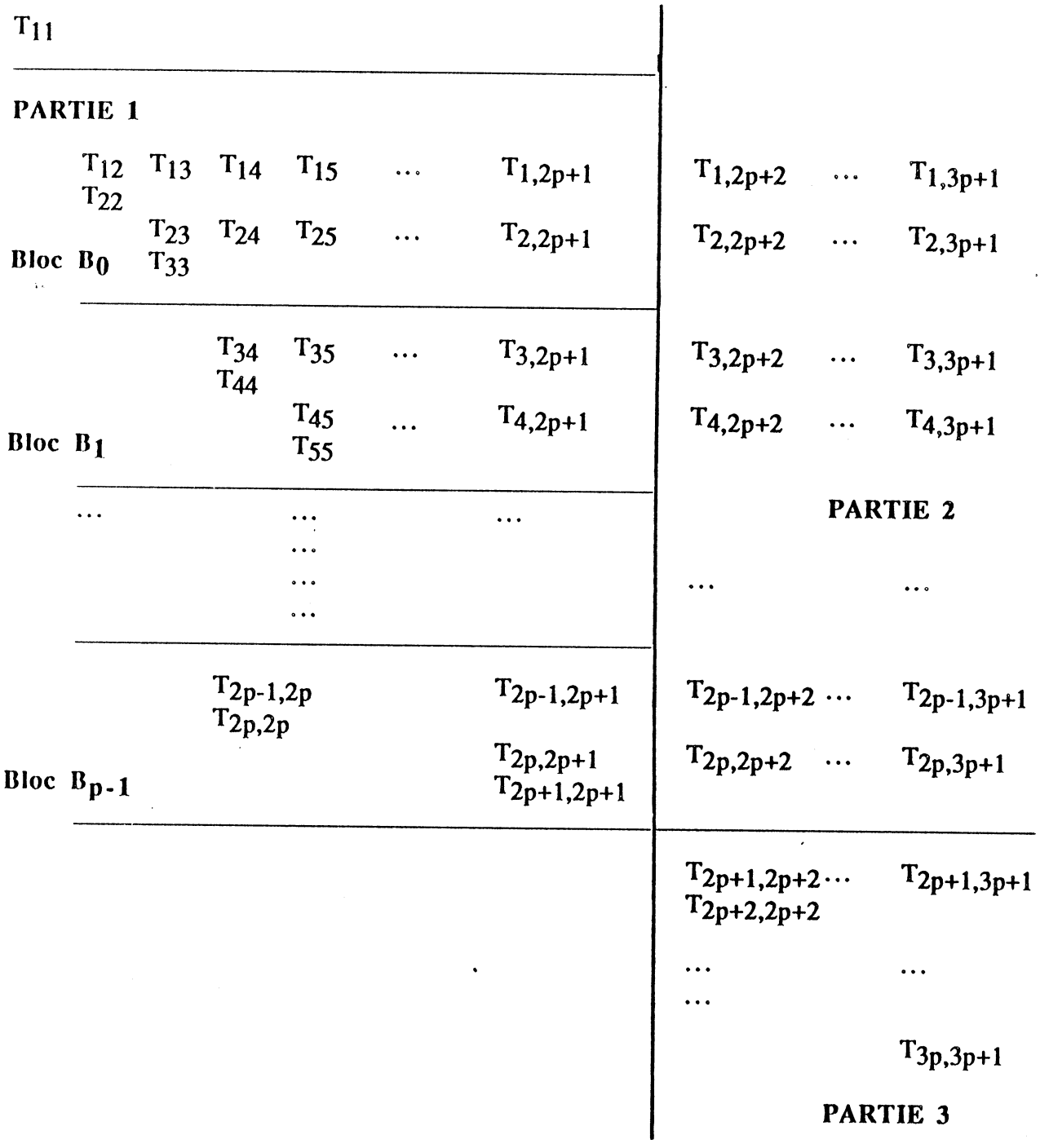


Figure 4: Partitionnement du graphe pour l'exécution de l'algorithme.

D'une manière générale, l'exécution du bloc B_k est confiée aux p-k premiers processeurs. Quand on passe à l'exécution du bloc suivant B_{k+1}, le processeur P_{p-k} rejoint les k processeurs restants pour exécuter désormais uniquement des tâches de la partie 2.

Notons ∂_k le temps d'exécution du plus long chemin (composé des tâches $T_{2k+1+i+j,2k+2+i,}$ $0 \leq i,j \leq 1$) du bloc B_k et $\Delta_k = \Sigma_{0 \leq j < k} \partial_k$. L'exécution du bloc B_k débute à l'instant Δ_k. Les p-k processeurs exécutent simultanément les p-k tâches $T_{2k+1,j}$, $2k+1 < j \leq p+k+1$. Puis P_1 exécute la tâche diagonale $T_{2k+2,2k+2}$

pendant que les autres processeurs exécutent les p-k-1 tâches $T_{2k+1,j}$, $p+k+1<j\leq 2p$. Les p-k processeurs exécutent ensuite simultanément les p-k tâches $T_{2k+2,j}$, $2k+2<j\leq p+k+2$. P_1 exécute alors la tâche $T_{2k+3,2k+3}$ tandis que P_{p-k} exécute la tâche $T_{2k+1,2p+1}$ et P_2, ..., P_{p-k-1} les tâches $T_{2k+2,j}$, $p+k+2<j\leq 2p$. A l'instant Δ_{k+1}, les processeurs P_1, ..., P_{p-k-1} abordent l'exécution du bloc B_{k+1}. Le processeur P_{p-k} débute l'exécution de la dernière tâche $T_{2k+2,2p+1}$ du bloc B_k à l'instant $\Delta_{k+1}+1$ et sera affecté au pool des processeurs effectuant des tâches de la partie 2 dès la fin de cette tâche. Pour mieux comprendre, on présente l'exemple suivant avec p=4 (on note le processeur i par P_i et la tâche T_{ij} par ij) :

P_1	P_2	P_3	P_4	
12	13	14	15	Tâches
0	0	0	0	Temps
22	16	17	18	Tâches
t_1	t_1	t_1	t_1	Temps
23	24	25	26	Tâches
t_2	t_2	t_2	t_2	Temps
33	27	28	19	Tâches
t_3	t_3	t_3	t_3	Temps
			29	Tâche
			t_4	Temps

Table 1 : Exécution du bloc B_0 de la partie 1.

On indique les temps de commencement des tâches à la table 1:

$t_1=Ex(T_{12})$, $t_2 = 2t_1$, $t_3=3t_1-1$ et $t_4=4t_1-1$.

Au temps t_4-1, tous les p processeurs sauf le processeur P_p commencent l'exécution du bloc B_1.

La partie 2 est composée d'un tableau rectangulaire de tâches, de taille 2p x p. Les tâches de la partie 2 sont exécutées ligne par ligne et de la gauche vers la droite: nous ajoutons la contrainte $T_{kj} \leq\leq T_{k,j+1}$, $1\leq k\leq 2p$, $2p+2<j\leq 3p$. La présence de p tâches indépendantes par ligne assure la possibilité de conserver actifs à tout instant, et en respectant les contraintes (B), les processeurs affectés à l'exécution de la partie 2. Les parties 1 et 2 sont effectuées avec une efficacité 1, i.e. aucun des p processeurs n'est inactif à un instant donné. Le temps total d'exécution est

donc le temps séquentiel des parties 1 et 2 divisé par p, soit $(n^3-p^3)/(3p) + O(n)$.

Lemme 3.1. Le schéma d'exécution précédent respecte les contraintes d'ordonnancement.

Pour établir le lemme, il suffit de comparer pour tout $k<p$, la somme

$$S_1(k) = p\Sigma_{0\le i\le k}\, 2(6p-4i-1) = 2(k+1)(6p-1)-4k(k+1)$$

des temps d'exécution des tâches T_{kj}, $1\le k\le 2k$, $2p+1<j\le 3p+1$, des 2k premières lignes de la partie 2 avec la somme

$$S_2(k) = (k+1)(-8k^2/3+(6p-16/3)k+9p-2)$$

des temps d'activité $\Delta_k-\Delta_{p-j+1}-Ex(T_{2j+2,2p+1})-1 = 2(k+1)(6p-2k-1)-3p-12pj+4j^2$ des k derniers processeurs P_j, $p-k<j\le p$, dans la partie 2. L'inégalité $S_1(k) > S_2(k)$, toujours vérifiée, assure que les contraintes (A) sont bien vérifiées jusqu'au début de l'exécution du bloc k.

On resynchronise tous les processeurs à la fin de la partie 2, ce qui fait perdre un facteur de temps linéaire. L'exécution de la partie 3 se fait en temps égal à la longueur du plus long chemin $T_{2p+1,2p+2}$, $T_{2p+2,2p+2}$, ..., $T_{3p,3p+1}$, soit $p^2 + O(n)$.

Le temps d'exécution total de l'algorithme proposé est donc $T_p = n^3/(3p) + 2p^2/3$ si $n=3p+1$, d'où une efficacité $e_{1/3} = 27/29$. Dans le cas géneral $p = \alpha n$, $\alpha\le 1/3$, il faut tenir compte d'une partie 0 du graphe d'ordonnancement composée des tâches $\{T_{kj},\ 1\le k\le n-3p-1,\ k\le j\le n\} \cup \{T_{n-3p,n-3p}\}$. Il est très facile de concevoir un schéma d'exécution d'efficacité 1 pour cette partie 0, par exemple un schéma de type glouton similaire à celui décrit pour l'exécution de la partie 2 où l'on fusionne les tâches $T_{k,k+1}$ et $T_{k+1,k+1}$ pour tout $k<n-3p$. Le temps total d'exécution de l'algorithme est encore $n^3/(3p) + 2p^2/3$. On a donc $f(\alpha) = 1/(3\alpha) + 2\alpha^2/3$, ce qui établit le

Théorème 3.1. L'algorithme proposé pour un problème de taille n avec $p=\alpha n$ processeurs, $\alpha\le 1/3$, est d'efficacité asymptotique

$$e_\alpha = 1 / (1+2\alpha^3) \ge 27/29.$$

L'algorithme ainsi défini ne possède pas une efficacité optimale mais est simple à programmer sur une machine parallèle. Nous construisons ci-dessous un algorithme asymptotiquement optimal pour toute valeur de α, mais beaucoup plus difficile à décrire et mettre en œuvre. Nous avons simulé sur Macintosh l'algorithme précédent. Le programme est les résutats sont présentés à la section 6.

3.2. RESULTATS DE COMPLEXITE

Lord, Kowalik et Kumar [37] ont proposé un algorithme parallèle avec $p = \lceil n/2 \rceil$ processeurs ($\alpha=1/2$). Il termine l'exécution du graphe en temps T_{opt}. Son efficacité asymptotique est $e_{\infty,1/2} = 2/3$. Il est clair que cet algorithme est asymptotiquement optimal. Ce résultat est amélioré par Veldhorst [66], qui a présenté un algorithme pour l'exécution du même graphe en temps T_{opt} avec seulement $p = \lceil (\sqrt{2}/4)n \rceil$ processeurs. Pour $\alpha \geq \sqrt{2}/4 \simeq 0.354$, on déduit qu'il existe des algorithmes balayant le graphe 2-pas en temps T_{opt} avec une efficacité asymptotique égale à $e_{\infty,\alpha} = 1/(3\alpha)$, et ce résultat est asymptotiquement optimal. Existe -t- il un algorithme exécutant le graphe avec αn processeurs en temps T_{opt} ? où $\alpha \geq \alpha_0 \simeq 0.347$ (α_0 est la valeur précédente, solution de l'équation $3\alpha - \alpha^3 = 1$). La valeur $\sqrt{2}/4$ est la plus petite publiée dans la littérature pour exécuter le graphe 2-pas en temps T_{opt}. Pour les valeurs de α telles que $0 < \alpha < \alpha_0$, on ne dispose que de résultats partiels. Il est difficile de montrer l'optimalité dans ce cas: on résoud toujours un problème matriciel de taille n avec un nombre de processeurs assez petit par rapport à n. Lord, Kowalik et Kumar [37] ont posé le problème du graphe 2-pas, et ils sont les premiers qui ont présenté des résultats de complexité sur ce type de problème. Les meilleurs résultats de la littérature à notre connaissance sont les suivants:

- si $\alpha \leq 2/\sqrt{43} \simeq 0.305$, il existe un algorithme possédant une efficacité asymptotique égale à $e_{\infty,\alpha} = 1/(1+\alpha^3)$, et cette valeur est optimale [48].
- si $2/\sqrt{43} < \alpha \leq 1/3$, il existe un algorithme dont l'efficacité asymptotique est $e_{\infty,\alpha} = 1/(1+2\alpha^3)$ [39].
- si $1/3 < \alpha < \sqrt{2}/4$, il existe un algorithme dont l'efficacité asymptotique est $e_{\infty,\alpha} = 1/(1+4\alpha^3)$ [8].

Pas d'algorithme optimal dans l'intervalle]0.305,0.354[auquel α_0 appartient.

Théorème 3.2. Pour un problème de taille n, avec $p = \alpha n$ processeurs, on construit ci-dessous un algorithme asymptotiquement optimal. L'efficacité asymptotique pour $\alpha \leq \alpha_0$, est $e_{\infty,\alpha} = 1/(1+\alpha^3)$, et pour $\alpha \geq \alpha_0$ $e_{\infty,\alpha} = 1/(3\alpha)$.

En particulier, on construit un algorithme avec $\alpha_0 n$ processeurs qui exécute le graphe d'ordonnancement en T_{opt}. On atteint aussi la borne du paragraphe précédent. Pour $\alpha \leq \alpha_0$, l'efficacité asymptotique est excellente: on a $e_{\infty,\alpha} \geq 0.959$. On atteint donc un très haut niveau de parallélisme.
Dans ce qui suit, on démontre le théorème 3.2. La figure 5 montre que le résultat de ce chapitre est meilleur que celui de Lord, Kowalik et Kumar [37].

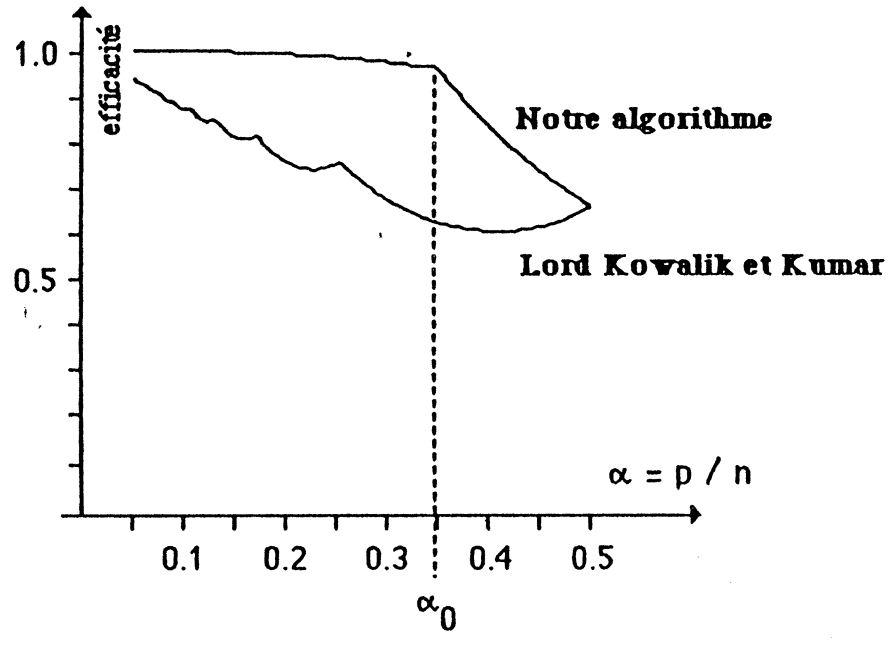


Figure 5 : Courbes des efficacités.

4. ALGORITHME ASYMPTOTIQUEMENT OPTIMAL

Dans cette section, on démontre le théorème 3.2. On commence par décrire un algorithme parallèle puis on montre qu'il est asymptotiquement optimal. Soit $\alpha_0 \simeq 0.347$ la solution de l'équation $3\alpha - \alpha^3 = 1$, et $\beta_0 = 1/\alpha_0$.

4.1. UN ALGORITHME AVEC $p = \alpha_0 n$ PROCESSEURS

Pour décrire l'algorithme, on partage le graphe des tâches en cinq régions comme l'indique la figure 6.
L'idée-clé est de garder chacun des processeurs actif pendant le plus long temps possible. Il est impossible de laisser après le niveau N_{n-p} tous les processeurs actifs. Après avoir exécuté la tâche $T_{n-p,n-p}$, il y a moins de p tâches pouvant être exécutées en parallèle, donc il y aura nécessairement des processeurs inactifs. Toutes les tâches du chemin $\{ T_{n-p+1,n-p}, T_{n-p+1,n-p+1}, \cdots, T_{n-1,n-2}, T_{n-1,n-1}, T_{n-1,n} \}$ s'exécutent séquentiellement. Soit t la longueur de ce chemin. Son exécution nécessite au moins t unités de temps. L'idée est de maximiser le

travail des p processeurs durant le temps d'exécution du chemin précédent. Ce qui correspond exactement à la région (V) de la figure 6.

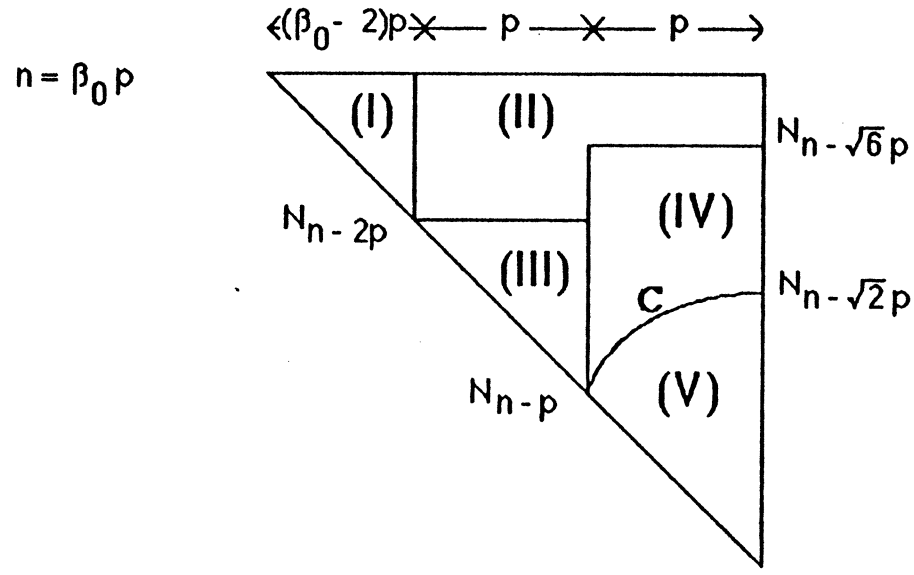


Figure 6 : Partitionnement du graphe de tâches.

Dans notre algorithme l'exécution des régions (I) à (IV) se fait en efficacité 1, tous les processeurs sont actifs. Quand on exécute la région (V), des processeurs deviennent obligatoirement inactifs, mais le temps perdu est minimum [48].

Description de l'algorithme

On appelle niveau N_k du graphe les tâches $\{ T_{k,k+1}, \ldots, T_{k,n} \}$. Elles peuvent être exécutées en parallèle. La courbe frontière C entre les regions (IV) et (V) est une courbe équipotentielle, elle contient la tâche $T_{n-p,n-p}$: elle est composée des tâches $T_{i(j),j}$, $n-p \leq j \leq n$, $i(j) = \max \{ i / cp(T_{ij}) \geq cp(T_{n-p,n-p+1}) \}$, où $cp(T_{ij})$ est la longueur du chemin critique à partir de la tâche T_{ij}. C'est la somme des longueurs des tâches T_{ij}, $T_{i+1,j}$, ..., T_{jj}, $T_{j,j+1}$, $T_{j+1,j+1}$, $T_{j+1,j+2}$, ..., $T_{n-1,n-1}$, $T_{n-1,n}$.

L'algorithme procède en trois phases séquentielles :

1 Dans la première phase, les tâches des régions (I) et (II) s'exécutent. $\lceil (\beta_0 - 2)p/2 \rceil$ processeurs débutent l'exécution de la region (I), et les autres commencent l'exécution de la région (II). Quand l'exécution de la région (I) progresse, des processeurs se libèrent. Ces derniers passent à la région (II). Les tâches de la région (II) s'exécutent par l'algorithme Glouton, niveau par niveau.

2 Dans la seconde phase, on exécute les tâches des régions (III) et (IV) . On associe une paire de colonnes de la région (III) et une autre paire de colonnes de la région (IV) à chaque paire de processeurs.

3 Dans la troisième phase, c'est l'exécution de la région (V), on associe simplement une colonne à chacun des processeurs.

Pour chaque région X, soit TW(X) le temps total des tâches appartenant à X, c'est à dire, la somme des longueurs des tâches de X. Soient $s_{1,1}$, $s_{1,2}$ et $s_{1,3}$ les sous-ensembles du plus long chemin s_1 compris respectivement dans les régions (I), (III) et (V). On désire exécuter le graphe en temps (équivalent à) $T_{opt} = L(s_1)$. Pour que l'exécution du graphe s'effectue en T_{opt}, il faut que la phase 1 dure $L(s_{1,1})$ unités de temps, la phase 2 dure $L(s_{1,2})$ unités de temps et la phase 3 dure $L(s_{1,3})$ unités de temps. On désire aussi exécuter les régions (I) à (IV) avec une efficacité égale à 1. Durant la première phase, tous les processeurs sont actifs pendant tout le temps d'exécution de cette phase, c'est à dire pendant $L(s_{1,1})$ unités de temps. Par conséquent, la surface balayée pendant la première phase est égale à $pL(s_{1,1})$, ce qui implique la relation $p\ L(s_{1,1}) = TW(I) + TW(II)$. De même, on doit avoir $p\ L(s_{1,2}) = TW(III) + TW(IV)$. On démontre plus tard ces égalités.

Première phase

On ne s'intéresse pas à l'exécution de la tâche T_{11}. Elle affecte le temps d'exécution seulement d'un facteur linéaire O(n), ce qui ne modifie donc pas l'évaluation asymptotique. Il y a deux pools P et Q de processeurs: les processeurs du pool P exécutent la région (I), les processeurs du pool Q exécutent la région (II). Au début, $q = \lceil(\beta_0-2)p/2\rceil$ processeurs appartiennent à P, et p-q à Q. A la fin de l'exécution de la phase 1, tous les processeurs sont dans Q. Plus précisément, un nouveau processeur passe au pool Q après l'exécution de deux niveaux de (I).
Pour mieux comprendre, on présente l'exemple suivant. Soit q=4 et supposons que l'exécution de la phase 1 débute à l'instant t=0. On est dans la situation suivante (figure 7):

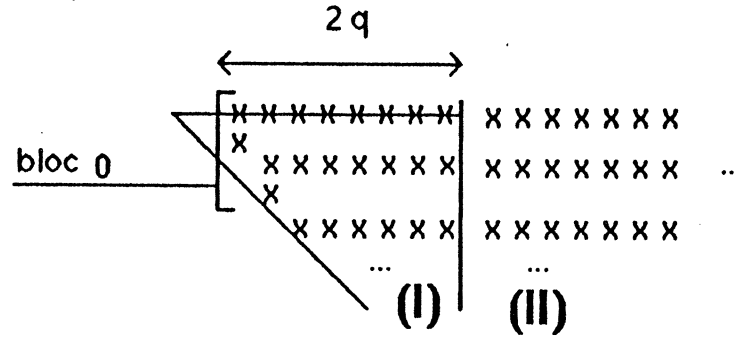


Figure 7 : Première phase de l'algorithme, région (I).

Le bloc 0 est composé des deux premiers niveaux de la région (I). Il s'exécute comme indiqué à la table 1. Les tâches successivement exécutées par le processeur 1 sont spécifiées à la figure 8.

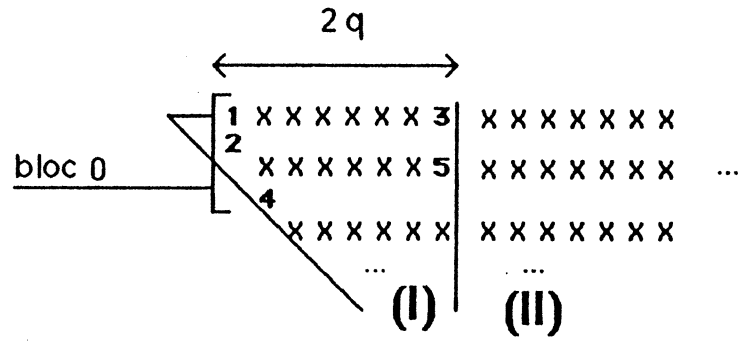


Figure 8 : Tâches exécutées par le processeur 1 du bloc 0 de la région (I).

P_1	P_2	P_3	P_4	P_2	P_3	P_4	P_1	
12	13	14	15	16	17	18	22	Tâches
0	0	0	0	t_1	t_1	t_1	t_1	Temps
19	23	24	25	26	27	28	33	Tâches
t_2-1	t_2	t_2	t_2	t_3	t_3	t_3	t_3	Temps
29								Tâche
t_4								Temps

Table 2 : Exécution du premier bloc de la région (I)

Dans la table 2, on note le processeur i par P_i ($1 \leq i \leq 4$), la tâche T_{ij} par ij.On indique les temps de commencement des tâches:

$$t_1 = Ex(T_{1,2}),\ t_2 = 2t_1,\ t_3 = 3t_1 - 1 \text{ et } t_4 = 4t_1 - 3.$$

A l'instant t_4, tous les q processeurs sauf le premier commencent l'exécution des deux niveaux suivants (bloc 1). Notons que le début de l'exécution du nouveau bloc par les q-1 processeurs aura lieu au même instant. Quand le processeur 1 termine l'exécution de la dernière tâche du niveau N_2, il est affecté au pool Q formé par les processeurs exécutant les tâches de la région II. Après la fin de l'exécution du bloc 1, le processeur 2 rejoint le pool Q. D'une façon générale, quand l'exécution du bloc k se termine , le processeur k-1 passe du pool P au pool Q.

Pool Q

Tous les processeurs du pool Q exécutent les tâches de la région (II), niveau par niveau et de gauche à droite en commençant à tout instant le maximum de tâches exécutables [8]. La région (II) possède plus de p tâches par niveau ce qui assure que tous les processeurs peuvent être simultanément actifs.

Dans cette première phase, pour démontrer que les contraintes d'ordonnancement sont respectées, il suffit de prouver que les processeurs exécutant la région (II) se situent toujours sur un niveau inférieur par rapport aux processeurs exécutant la région (I).

Lemme 3.2. $p\,L(s_{1,1}) = TW(I) + TW(II) + O(n^2)$.

On a

$$L(s_{1,1}) = 2\sum_{x=1}^{n-2p}(n-x) = (\beta_0^2-4)p^2 + O(n)$$

où $n = \beta_0 p$

On calcule la surface totale de chacune des régions (I) et (II) :

$$TW(I) = \sum_{x=1}^{n-2p}(n-2p-x)(n-x) = p^3\,\frac{\beta_0^3 - 3\beta_0^2 + 4}{3} + O(n^2) = p^3 + O(n^2)$$

β_0 est une solution de l'équation $\beta^3 - 3\beta^2 + 1 = 0$

$$TW(II) = \sum_{x=1}^{n-2p} p(n-x) + \sum_{x=1}^{n-\sqrt{6}p} p(n-x) = p^3\,(\beta_0^2-5) + O(n^2)$$

Par la suite

$$p\,L(s_{1,1}) = TW(I) + TW(II) + O(n^2).$$

D'après cette relation, quand le dernier processeur du pool P termine l'exécution de la région (I), les processeurs du pool Q terminent également l'exécution de la région (II) à un facteur linéaire près.

Lemme 3.3. Les contraintes d'ordonnancement sont respectées durant la première phase de l'algorithme.

Par construction de l'algorithme, les contraintes (A) et (B) sont respectées durant l'exécution de la région (I). Pour la région (II), il est évident que la contrainte (B) est vérifiée. La contrainte (A) est vérifiée lors de l'exécution de la partie constituée des niveaux $N_1, N_2, \ldots, N_{n-\sqrt{6}p}$, en effet, cette partie de la région (II) possède 2p tâches par niveau : chaque processeur du pool Q exécute au moins deux tâches de chaque niveau. Par suite, l'exécution de cette partie progresse verticalement au plus à la même vitesse que la partie (I), et les contraintes (A) sont respectées. Pour vérifier que la contrainte (A) est respectée dans la deuxième partie de la région (II), on renvoie le lecteur à la figure 9.

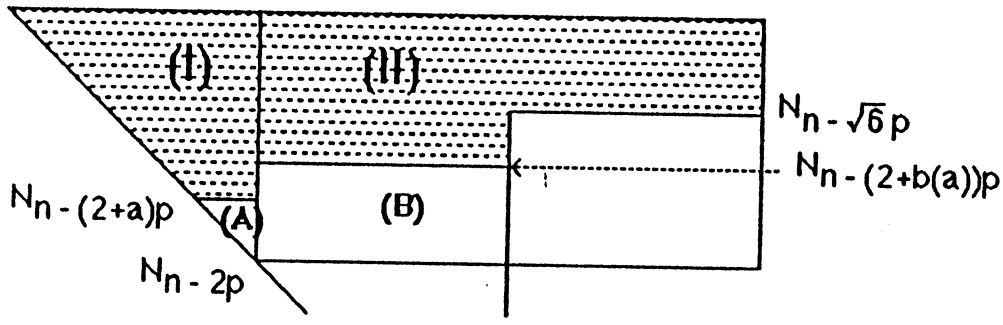


Figure 9 : Vérification de la contrainte (A) dans la phase 1.

Supposons que quand les processeurs du pool P se trouvent sur le niveau $N_{n-(2+a)p}$, de la région (I) ($0 \le a \le \sqrt{6}-2$), les processeurs du pool Q se situent sur le niveau $N_{n-(2+b(a))p}$ de la région (II). Si on obtient $b(a) \ge a$, l'algorithme dans cette première phase respecte bien les contraintes d'ordonnancement.
Avec les notations de la figure 9, le lemme 3.2. implique la relation suivante :

$$p\, L(s_{1,1} \cap (A)) = TW(A) + TW(B) + O(n^2)$$

Ce qui entraîne que

$$p^3 (4a + a^2) = p^3 (a^3/3 + a^2) + p^3 (b(a)^2/2 + 2b(a)) + O(n^2)$$

Donc

$$b(a)^2/2 + 2b(a) = 4a - a^3/3$$

Il est facile de vérifier que $b(a) \ge a$ pour tout a tel que $0 \le a \le \sqrt{6}-2$.
On resynchronise tous les processeurs à la fin de la phase 1, au prix d'un facteur linéaire. Ce qui n'affecte pas l'évaluation asymptotique du temps d'exécution.

Seconde phase

Supposons que le nombre de processeurs est pair. L'idée fondamentale dans cette phase est la suivante: on réunit les processeurs par paires, chaque paire de processeurs exécute une paire de colonnes dans chacune des régions (III) et (VI). On explique en détail l'algorithme pour les régions (III) et (IV) après avoir présenté les notations de la figure 10:

Soit $x = j / p$, $0 \le x \le 1/2$. Les processeurs j et p-j+1 sont associés à l'exécution des quatre colonnes suivantes :
- la colonne j de la région (III), notée $c_1(x)$ dans la figure 10
- la colonne (p-j+1) de la région (III), notée $d_1(x)$ dans la figure 10
- la colonne j de la région (IV), notée $c_2(x)$ dans la figure 10
- la colonne (p-j+1) de la région (IV), notée $d_2(x)$ dans la figure 10

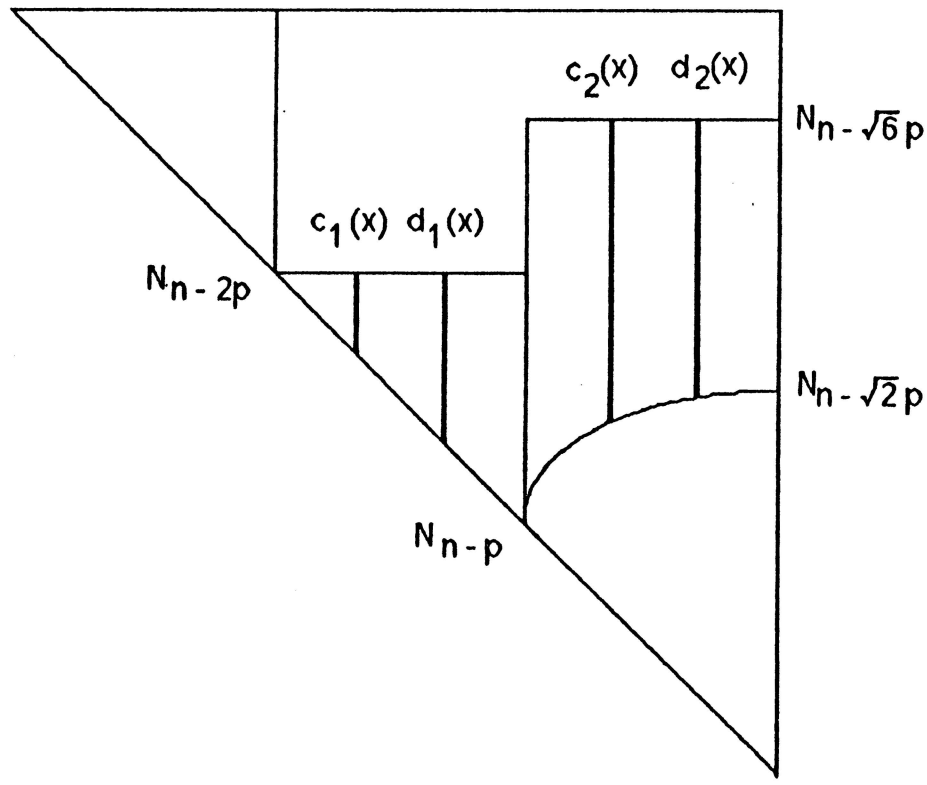


Figure 10 : Exécution de la phase 2.

On désire exécuter la phase 2 en temps $L(s_{1,2})$, et avec une efficacité égale à 1. On doit avoir la relation $p\ L(s_{1,2}) = TW(III) + TW(IV)$. Remarquons que la somme des temps des quatre colonnes exécutées par une paire de processeurs est indépendante de x et est égale à $2L(s_{1,2})$. Le lemme suivant prouve cette remarque :

Lemme 3.4. Pour tout x tel que $0 \le x \le 1/2$, on a

$$2\,L(s_{1,2}) = TW(c_1(x)) + TW(d_1(x)) + TW(c_2(x)) + TW(d_2(x)) + O(n)$$

Soit x tel que $0 \le x \le 1/2$ et $j = x\,p$. On calcule facilement $TW(c_1(x))$ et $TW(d_1(x))$:

$$TW(c_1(x)) = \sum_{i=1}^{j} (2p-j) = \left(\frac{4 - (2 - x)^2}{2}\right)p^2 + O(n)$$

De même

$$TW(d_1(x)) = \left(\frac{4 - (1 + x)^2}{2}\right)p^2 + O(n)$$

Pour $c_2(x)$ et $d_2(x)$, on calcule d'abord les niveaux où les colonnes de la région (IV) coupent la courbe (C). Soit $N_{n-c(j)}$ le niveau où la colonne j de la région (IV) coupe la courbe (C): les deux chemins présentés dans la figure 11 sont égaux.

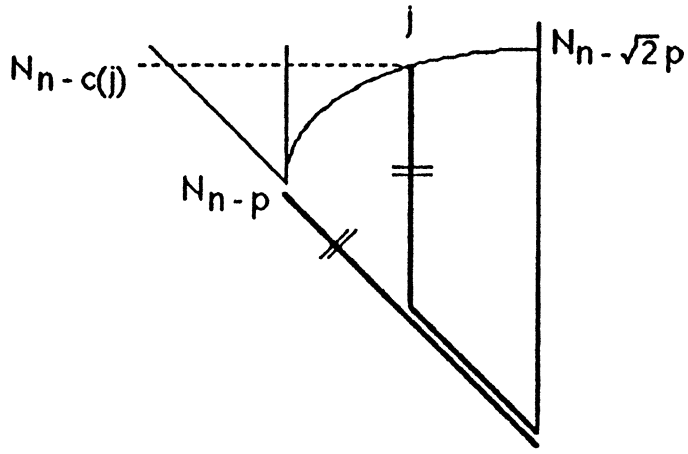


Figure 11 : Détermination de la courbe (C) anlytiquement.

Alors, on a

$$\frac{c(j)^2 - (p-j)^2}{2} = L(s_{1,3}) - (p-j)^2 + O(n)$$

où

$$L(s_{1,3}) = p^2 + O(n)$$

Ce qui entraîne que

$$c(j) = \sqrt{2p^2 - (p-j)^2} + o(n)$$

On obtient facilement pour tout x tel que $0 \leq x \leq 1/2$:

$$TW(c_2(x)) = \left(\frac{4 + (1-x)^2}{2}\right)p^2 + O(n) \text{ et } TW(d_2(x)) = \left(\frac{4 + x^2}{2}\right)p^2 + O(n)$$

On a alors

$$TW(c_1(x)) + TW(d_1(x)) + TW(c_2(x)) + TW(d_2(x)) = 6p^2 + O(n)$$

Puisque $L(s_{1,2}) = 3p^2 + O(n)$ alors le lemme 3.4. est prouvé.

Bien sûr, il n'est pas suffisant de connaître la longueur des quatre colonnes associées à une paire de processeurs pour avoir un algorithme fonctionnant convenablement. Dans ce qui suit, on détermine un algorithme affectant les tâches aux processeurs. Notons tout d'abord qu'on a $TW(c_1(x)) + TW(c_2(x)) \leq TW(d_1(x)) + TW(d_2(x))$ pour tout x tel que $0 \leq x \leq 1/2$. Le processeur j (avec $j = xp$) exécute toutes les tâches des colonnes $c_1(x)$ et $c_2(x)$, et quelques tâches de la colonne $d_2(x)$. Le processeur $p-j+1$ exécute toutes les tâches de la colonne $d_1(x)$ et la partie restante de la colonne $d_2(x)$. On considère la figure 12:

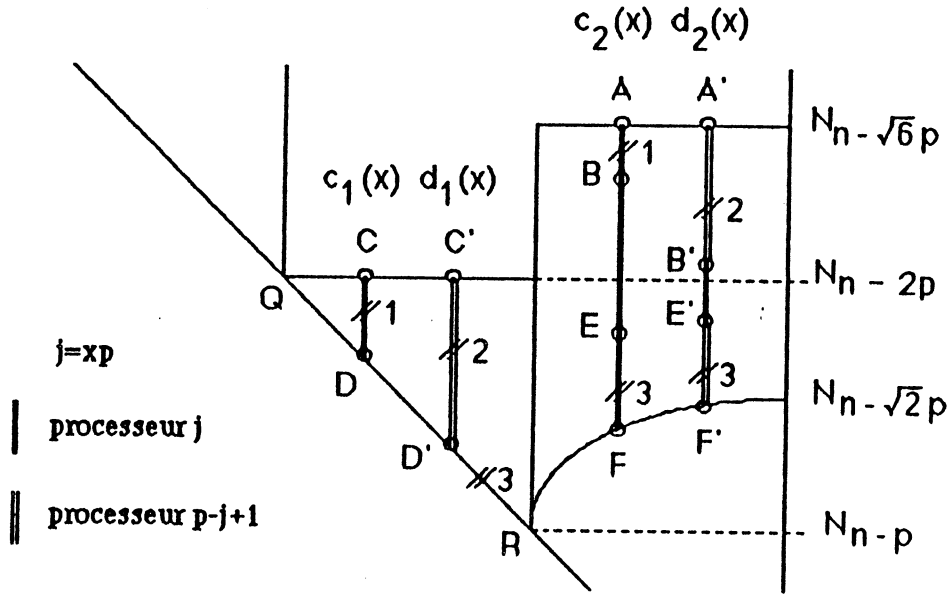


Figure 12 : Répartition des tâches aux processeurs dans la seconde phase.

Dans la figure 12, deux segments marqués // i, $1 \le i \le 3$, ont la même longueur. Etant donné $x \le 1/2$, B,B',E et E' sont choisis de telle façon qu'on a les relations suivantes :

- $L(AB) = L(CD) = [2 - (2-x)^2/2]\, p^2 + O(n)$ (notons que $L(CD) = WT(c_1(x)) = L(QD)/2$)
- $L(A'B') = L(C'D') = [2 - (1+x)^2/2]\, p^2 + O(n)$ (notons que $L(C'D') = WT(d_1(x)) = L(QD')/2$)
- $L(EF) = L(E'F') = L(D'R) = (2x+x^2)p^2 + O(n)$

Le processeur j ($j \le 1/2$):

(i) exécute le segment AB

(ii) exécute le segment CD

(iii) exécute les segments BE et B'E'. Au début le processeur j exécute seulement les tâches du segment BE jusqu'au niveau où le point B' appartient. En atteignant ce dernier niveau, le processeur j exécute les deux segments. Plus précisement, les deux segments BE et B'E' ne sont pas exécutés successivement, mais l'exécution s'effectue de haut en bas et de gauche à droite, c'est à dire alternativement une tâche du segment BE, et une tâche du segment B'E'.

(iv) exécute le segment EF

Le processeur p-j+1($j \le 1/2$):

(i) exécute le segment A'B'

(ii) exécute le segment C'D'

(iii) exécute le segment E'F'

On a $L(AB) + L(CD) = L(QD)$, ce qui entraîne que la tâche diagonale de la colonne $c_1(x)$ s'exécute par le processeur j à l'instant où elle sera prête et sans perte de temps. De même pour la tâche diagonale de la colonne $d_1(x)$. Le temps

mis pour exécuter cette deuxième phase est $L(s_{1,2})$.

Lemme 3.5. L'algorithme de la seconde phase respecte les contraintes d'ordonnancement.

Puisque l'algorithme de cette seconde phase exécute les tâches par colonnes, alors la contrainte (B) est vérifiée. On prouve tout d'abord que la contrainte (A) est vérifiée par le processeur p-j+1. Soit x tel que $0<x\leq 1/2$ et $j = xp$.:

• durant l'exécution du segment A'B': Quand le processeur p-j+1 se situe sur le niveau N_{n-up}, le plus long chemin exécuté dans la région (III) est égal à la longueur du chemin situé entre les niveaux $N_{n-\sqrt{6}p}$ et N_{n-up} de la colonne $d_2(x)$. C'est à dire, on a exécuté le plus long chemin jusqu'au niveau $N_{n-f(u)p}$ de la région (III). On a alors

$$(3-u^2/2)\, p^2 = (4-f(u)^2)\, p^2 + O(n)$$

Pour tout u tel que $u\geq\sqrt{2}$, on a $f(u)\geq u$. Mais $L(A'B') = L(C'D') \leq TW(d_1(x)) = 3p^2/2 + O(n)$ quand x tend vers 0, donc u varie dans l'intervalle $[\sqrt{3},\sqrt{6}]$. La contrainte (A) est par conséquent respectée.

• durant l'exécution du segment C'D': de la relation $L(A'B') + L(C'D') = L(QD'))$, on conclut que le processeur p-j+1 ne peut pas dépasser le niveau où une tâche du plus long chemin est en cours d'exécution. Alors la contrainte (A) est respectée durant l'exécution du segment C'D'.

• durant l'exécution du segment E'F': quand le processeur p-j+1 atteint le point F', le point R sera aussi atteint. Puisque L(D'R)=L(E'F'), le processeur p-j+1 se situe toujours sur un niveau N_k tel que $k\leq m$ où le niveau N_m est le niveau sur lequel une tâche du plus long chemin est en cours d'exécution.

Pour le processeur j, on prouve que la contrainte (A) est respectée:

• durant l'execution du segment AB: le même argument donné pour le processeur p-j+1 durant l'exécution du segment A'B'.

• durant l'exécution de CD: le même argument donné pour le processeur p-j+1 durant l'exécution du segment C'D'.

• durant l'exécution des segments BE et B'E': quand l'exécution se fait alternativement : une tâche du segment BE puis une tâche du segment B'E', la contrainte (A) est vérifiée: le processeur j exécute deux tâches avant de passer au niveau suivant de chacune des colonnes $c_2(x)$ et $d_2(x)$, et progresse donc verticalement au même rythme que le processeur qui est en train d'exécuter une tâche du plus long chemin. Dans le cas où le processeur j exécute seulement le segment BE, la contrainte (A) est respectée aussi: le même argument donné pour le processeur p-j+1 durant l'exécution du segment A'B'. Le processeur j termine l'exécution du segment B'E' avant que le processeur p-j+1 ne termine l'exécution de la colonne $d_1(x)$.

• durant l'exécution des niveaux situés entre les points F et F': les points F et R sont atteints au même instant (lemme 3.4.) et puisque le niveau où se situe F est plus petit que celui où se situe R alors la contrainte (A) est vérifiée.

A la fin de l'exécution de cette deuxième phase, on resynchronise les processeurs. Le temps d'inactivité d'un processeur est de l'ordre de O(n) ce qui n'affecte pas l'évaluation asymptotique du temps d'exécution.

Troisième phase

On affecte à chaque processeur une colonne de la région (V): le processeur j exécute la colonne n-j y compris la tâche diagonale $T_{n-j,n-j}$.
On prouve facilement que la contrainte (A) est respectée. Supposons que les processeurs abordent l'exécution de la région (V) à l'instant t=0. D'après la définition de la courbe (C), le processeur j atteint le niveau N_{n-j} à l'instant

$$t_{jj} = cp(T_{n-p,n-p}) - j^2.$$

Pour tout i<j, le processeur i atteint le niveau N_{n-j} à l'instant

$$t_{ij} = cp(T_{n-p,n-p}) - i^2 - (j^2/2-i^2/2).$$

Alors $$t_{ij} \geq t_{jj}+TW(T_{n-j,n-j}),$$

et les contraintes d'ordonnancement sont respectées.
La région (V) s'exécute alors en temps $cp(T_{n-p,n-p}) = L(s_{1,3}) = p^2 + O(n)$.
La description de l'algorithme est par conséquent terminée. Le temps d'exécution total est égal à $T_{opt} = L(s_1)$ plus un facteur linéaire. L'algorithme est asymptotiquement optimal, son efficacité asymptotique est égale à

$$e_{\infty,\alpha} = \lim_{n\to\infty} \frac{n^3/3}{(\alpha_0 n)\, n^2} = \frac{1}{3\,\alpha_0} \simeq 0.959$$

4.2. AVEC p=αn PROCESSEURS, α quelconque

Supposons qu'on dispose $p = \alpha n$ processeurs, où $0 \leq \alpha \leq 1$. Si $\alpha \geq \alpha_0$, on utilise le même algorithme qu'avec $\alpha_0 n$ processeurs, lequel est encore asymptotiquement optimal, puisqu'il s'exécute en temps T_{opt}. L'efficacité asymptotique est alors égale à $e_{\infty,\alpha} = 1/(3\alpha)$.

Pour $\alpha \leq \alpha_0$, on a

Lemme 3.6. L'efficacité asymptotique $e_{\infty,\alpha}$ d'un algorithme parallèle avec $p = \alpha n$ processeurs est plus petite que $1/(1+\alpha^3)$.

Durant l'exécution de chaque niveau possédant 2j tâches, seulement j processeurs sont actifs. Ce qui entraîne que

$$TIA = \Sigma_{1 \leq j \leq p} 2j \cdot (p-j) = p^3/3 + O(n^2)$$

le temps libre total de tous les processeurs est:

$$CA \leq p\,T_p - TIA = p\,T_p - p^3/3 + O(n^2)$$

(Preuve similaire à celle faite pour trouver la borne α_0).

La somme totale de toutes les tâches du graphe est égale à $TW = n^3/3 + O(n^2)$, c'est le temps mis T_1 pour exécuter le graphe par un seul processeur. Alors $CA \geq TW$, et on a $p\,T_p \geq T_1 + p^3/3 + O(n^2)$, ce qui entraîne

$$e_{p,\infty} \leq 1/(1+p^3/n^3)$$

Pour $\alpha \leq \alpha_0$, on construit maintenant un algorithme parallèle. On montre que son efficacité asymptotique est égale à la borne donnée par le lemme 3.6.

Soit $\beta = 1/\alpha$. On ajoute simplement la phase de la région (0) composée des niveaux $N_1, N_2, \ldots, N_{n - \beta 0 p}$: voir figure 13.

On ne considère pas la tâche T_{11} de la région (0): elle affecte le temps d'exécution parallèle seulement d'un facteur linéaire qui ne modifie pas le résultat asymptotique. Pour simplifier, on suppose l'existence d'un processeur exécutant les tâches diagonales $T_{22}, T_{33}, \ldots$ dès que possible. De même, l'évaluation asymptotique du résultat n'est pas affectée. Les autres tâches de la région (0) s'exécutent à l'aide de l'algorithme Glouton. Ce dernier exécute les tâches en balayant les lignes de la région (0) de gauche à droite, en commençant à tout instant le maximum de tâches exécutables. Les contraintes d'ordonnancement sont respectées. En effet, un niveau de la région (0) possède plus de 2p tâches ce qui implique qu'un processeur exécute au moins deux tâches par niveau.

On resynchronise tous les processeurs à la fin de cette phase. La région (0) est exécutée avec une efficacité maximale: tous les processeurs sont actifs pendant tout le temps d'exécution. Son temps d'exécution parallèle est donc égal au temps d'exécution séquentielle de toutes les tâches de la région (0) TW(0), divisé par le nombre de processeurs p. Pour les régions (I) jusqu'à (V), on procède comme pour $\alpha = \alpha_0$.

Lemme 3.7. Pour $\alpha \leq \alpha_0$, l'efficacité asymptotique de l'algorithme présenté est égale à $e_{\alpha,\infty} = 1/(1+\alpha^3)$.

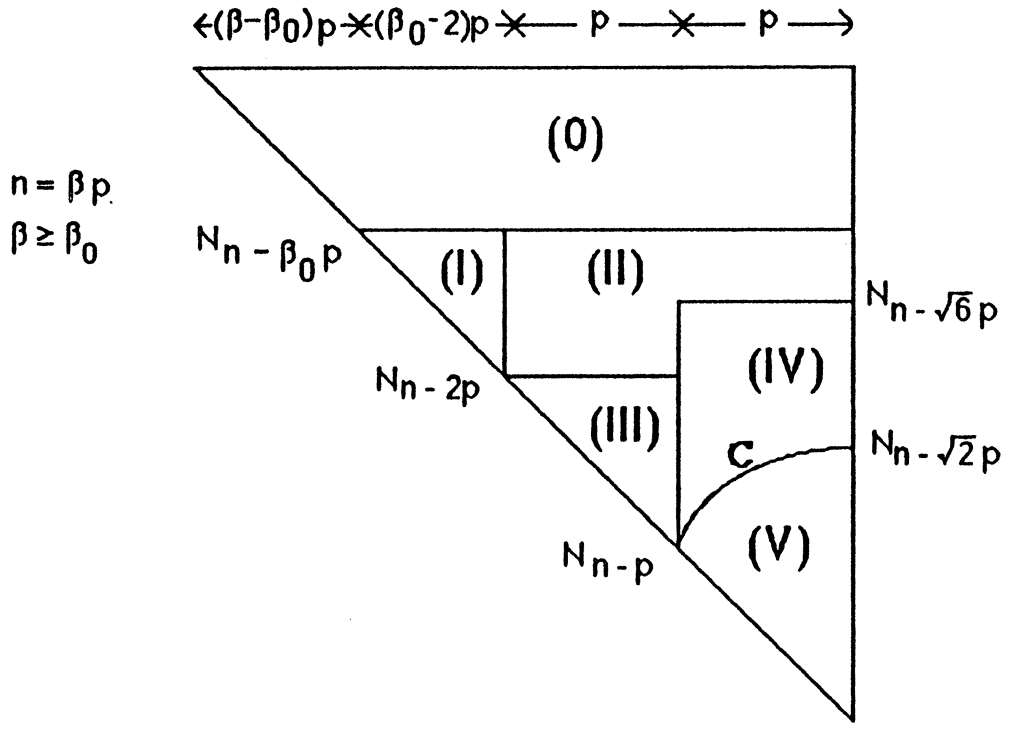


Figure 13 : Partitionnement du graphe quand $p = \alpha n$, $\alpha \leq \alpha_0$

Le temps d'exécution séquentielle est $T_1 = n^3/3 + O(n^2)$. Pour l'exécution parallèle, son temps d'exécution est égale à la somme Σ des surfaces des régions (0), (I), (II), (III) et (IV) divisée par p (l'efficacité asymptotique est égale à 1) plus $cp(T_{n-p,n-p})$, le temps mis pour exécuter la région (V):

$$T_p = \Sigma / p + cp(T_{n-p,n-p}).$$

La surface de la région (V) est égale à $TW(V) = 2p^3/3$ (le calcul est fait à l'aide de la définition de (C)), et $cp(T_{n-p,n-p}) = p^2 + O(p)$. On a

$$\Sigma = T_1 - TW(V)$$

alors

$$T_p = (n^3+p^3) / (3p)$$

et l'efficacité asymptotique $e_{\infty, \alpha}$ voulue est prouvée.

Remarquons qu'il y a continuité de $e_{\infty, \alpha}$ pour $\alpha = \alpha_0$.

4.3. INTERPRETATION GEOMETRIQUE DE α_0

Considérons le partitionnement du graphe de tâches présenté au début. Remarquons que le partage du graphe en régions (0) jusqu'au (IV) a un but seulement technique. Seule la région (V) est vraiment importante. En effet, la

région (V) est la région de surface maximale qui peut être exécutée en temps $L(s_{1,3})$. Avec la notation TIA introduit plus haut, on a

$$p\, L(s_{1,3}) = TW(V) + TIA + O(n^2)$$

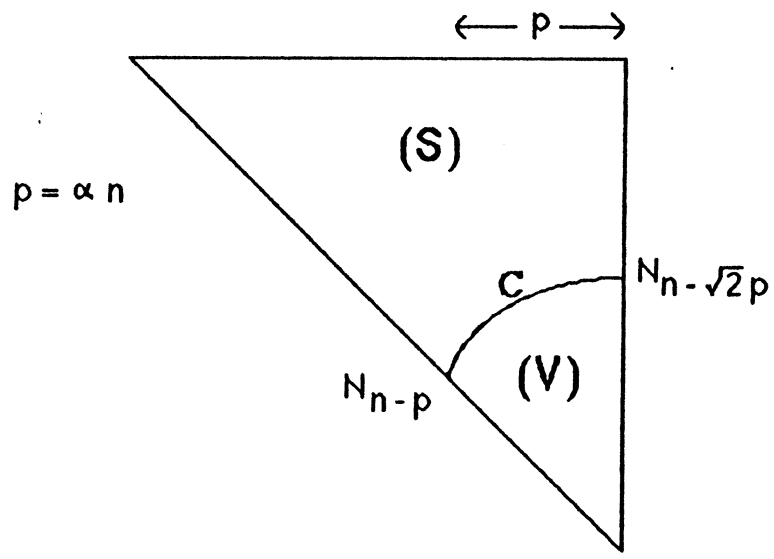


Figure 14 : Interprétation géométrique de α_0

Considérons la figure 14, et soit $p = \alpha n$. On sait que l'exécution du graphe s'effectue en temps $T_{opt} = L(s_1)$ pour $\alpha \geq 1/2$. Pour pouvoir obtenir une exécution parallèle en temps $L(s_1)$, on doit obligatoirement exécuter toutes les tâches de la région (S) en temps $L(s_1 \cap S)$, puisqu'on exécute seulement les tâches de la région (V) en temps $L(s_{1,3})$. Ce qui implique la relation $TW(S) \leq p\, L(s_1 \cap S)$, c'est à dire $n^3 - 2p^3/3 \leq p(n^2 - p^2) + O(n^2)$, soit $3\alpha - \alpha^3 - 1 \geq 0$, ou encore $\alpha \geq \alpha_0$. La caractérisation de α_0 est la suivante:

Proposition 3.1. $\alpha_0 = \min \{ \alpha / TW(S) \leq p\, L(s_1 \cap S) \}$.

5. CONCLUSION

Etant donnée une matrice dense A non singulière de taille nxn et $p = \alpha n$ processeurs, $\alpha \leq 1$, on a défini des algorithmes parallèles pour la décomposition LU de la matrice A avec pivotage partiel sur un ordinateur multiprocesseur. Le premier algorithme parallèle présenté n'est pas optimal mais il est facile à programmer sur un ordinateur parallèle. Le second algorithme parallèle présenté est asymptotiquement optimal pour toutes les valeurs de α ($0 < \alpha \leq 1$), et son efficacité asymptotique est maximale. En particulier, on atteint la borne de [37]

pour le nombre minimum de processeurs exécutant le graphe en temps $T_{opt} = n^2 + O(n)$. Ce qui améliore le temps d'exécution du second algorithme est le fait que tous les processeurs se situent, quand on exécute la tâche $T_{n-p,n-p}$, sur la courbe (C), qui est une équipotentielle du chemin critique. Ce qui n'est pas le cas pour le premier algorithme: à la fin, les processeurs se situent sur au plus deux niveaux consécutifs et le temps total d'inactivité des processeurs n'est pas minimal.

6. RESULTATS DE SIMULATION

Dans ce paragraphe, nous simulons l'algorithme parallèle défini au pragraphe 3.1. de ce chapitre.

6.1. PROGRAMME

```
program algo_2_pas;
 type
  processeur = ^proc;
  proc = record
       suiv, pred : processeur;
       ligne, colonne : integer;
        temps : real;
       end;
  tache = ^tac;
  tac = record
       suiv : tache;
       ligne, colonne : integer;
       chemin : real;
       end;
  coefficient = ^coef;
  coef = record
       ligne, colonne : integer;
       suiv : coefficient;
       end;
  tab = array[1..1000] of real;
  temps_diagonale = array[1..1000] of real;
 var
  i, j, n, m, nombre, diag, l, k, w, niv, t1 : integer;
  tetepr, finpr, pp, qp, p1 : processeur;
  processeur_libre, deb, pas, etape, n1, a : integer;
  temps_courant, alpha1 : real;
  teteex, pt, qt : tache;
  teteat : coefficient;
  t : tab;
  temps_diag : temps_diagonale;
  alpha, temps_seq, efficacite, opt : real;
  sortie : boolean;
```

```
 seqreal, seqparal : real;

{Evaluation de la durée d'une tâche }

function duree (i, j : integer) : integer;
begin
 if i = j then
     duree := n - i + 1
 else
     duree := n - i
end;

{Evaluation de la fonction d'ordonnancement des tâches}

function ch (i, j : integer) : real;
begin
 if (i = j) then
     ch := 1.0 * n * n
 else
     if (i < j) then
          begin
          if (j < n - nombre) or (i < n - 3 * nombre) then
               ch := 1.0 * (n - i) * n + (n - j)
          else
               if (j = n - nombre) and (i >= n - 3 * nombre) then
                    begin
                    if ((i + 3 * nombre  - n) mod 2 = 0) then
                         ch := 1.0 * (n - i - 1) * n + (n - j)
                    else
                         ch := 1.0 * n * n;
                    end;
          end;
 if (etape = 3) or (etape = 4) then
     begin
     if i < j then
          ch := 1.0 * (n - i) * n + (n - j);
     end;
end;

{Placement d'un processeur dans la file d'attente}

procedure place (pp0 : processeur);
 var
  pp1, pp2 : processeur;
  fin : boolean;
begin
 if tetepr = nil then
    begin
```

```
    tetepr := pp0;
    finpr := pp0;
    pp0^.suiv := nil;
    pp0^.pred := nil;
    end
  else
    begin
    if tetepr^.temps <= pp0^.temps then
        begin
        pp0^.suiv := tetepr;
        tetepr^.pred := pp0;
        tetepr := pp0;
        pp0^.pred := nil;
        end
    else
        begin
        if tetepr <> finpr then
            begin
            pp1 := tetepr;
            pp2 := tetepr^.suiv;
            fin := false;
            end
        else
            begin
            finpr := pp0;
            pp0^.pred := tetepr;
            tetepr^.suiv := pp0;
            pp0^.suiv := nil;
            fin := true;
            end;
        while not fin do
            begin
            if pp2 = nil then
                begin
                pp1^.suiv := pp0;
                finpr := pp0;
                fin := true;
                pp0^.pred := pp1;
                pp0^.suiv := pp2;
                end
            else
                begin
                if pp0^.temps >= pp2^.temps then
                    begin
                    pp1^.suiv := pp0;
                    pp0^.suiv := pp2;
                    pp0^.pred := pp1;
                    pp2^.pred := pp0;
```

```
                        fin := true;
                        end
                    else
                        begin
                        if pp2 <> finpr then
                            begin
                            pp1 := pp2;
                            pp2 := pp2^.suiv;
                            end
                        else
                            begin
                            finpr := pp0;
                            pp0^.pred := pp2;
                            pp2^.suiv := pp0;
                            fin := true;
                            end
                        end
                    end
                end
            end
        end
    end;
```

{Affectation d'une tâche à un processeur}

```
procedure affectation;
 var
  qp : tache;
  pp : processeur;
 begin
 processeur_libre := processeur_libre - 1;
 new(pp);
 pp^.ligne := teteex^.ligne;
 pp^.colonne := teteex^.colonne;
 pp^.temps := temps_courant + duree(pp^.ligne, pp^.colonne);
 temps_seq := temps_seq + duree(pp^.ligne, pp^.colonne);
 place(pp);
 qp := teteex;
 teteex := teteex^.suiv;
 qp^.suiv := nil;
  dispose(qp);
end;
```

{Mise d'une tâche dans la liste des tâches exécutables}

```
procedure miseex (i1, j1 : integer);
 var
  p1, p2, pt : tache;
```

```
  fin : boolean;
 begin
 new(pt);
 pt^.ligne := i1;
 pt^.colonne := j1;
 pt^.chemin := ch(i1, j1);
 if teteex = nil then
      begin
      teteex := pt;
      teteex^.suiv := nil;
      end
else
      begin
      if teteex^.chemin <= pt^.chemin then
            begin
            pt^.suiv := teteex;
            teteex := pt;
            end
      else
            begin
            p1 := teteex;
            p2 := teteex^.suiv;
            fin := false;
            while not fin do
                  begin
                  if p2 = nil then
                        begin
                        p1^.suiv := pt;
                        pt^.suiv := nil;
                        fin := true;
                        end
                  else
                        begin
                        if pt^.chemin >= p2^.chemin then
                              begin
                              p1^.suiv := pt;
                              pt^.suiv := p2;
                              fin := true;
                              end
                        else
                              begin
                              p1 := p2;
                              p2 := p2^.suiv;
                              end
                        end
                  end
            end
      end
```

```
 end;
```

{Mise d'une tâche dans une liste des tâches liberées des contraintes verticales}

```
procedure miseat (i2, j2 : integer);
  var
   p : coefficient;
begin
new(p);
p^.ligne := i2;
p^.colonne := j2;
p^.suiv := teteat;
teteat := p;
end;
```

{Transfert des tâches de la liste d'attente vers la liste des tâches} {exécutables c'est à dire liberées des contraintes diagonales}

```
 procedure libeat (ii : integer);
 var
  p, q : coefficient;
 begin
 p := teteat;
 teteat := nil;
while p <> nil do
     begin
     if p^.ligne = ii then
           begin
           miseex(ii, p^.colonne);
           q := p;
           p := p^.suiv;
           q^.suiv := nil;
           dispose(q);
           end
     else
           begin
           q := p^.suiv;
           p^.suiv := teteat;
           teteat := p;
           p := q;
           end
     end
 end;
```

{préparer l'exécution de l'étape 3}

```
 procedure preparation;
```

```
 begin
  if i = j then
      temps_diag[i] := temps_courant;
  if k < nombre then
      begin
      if (i = n - 3 * nombre + 2 * k - 1) and (j = n - nombre) then
            begin
            processeur_libre := processeur_libre - 1;
            t[k] := temps_courant;
            k := k + 1;
            end;
      end
   else
      if k = nombre then
      begin
      if (i = n - nombre) and (j = n - nombre) then
            begin
            processeur_libre := processeur_libre - 1;
            t[k] := temps_courant;
            end
      end
 end;

{Début du programme}

begin

{Initialisation du problème}

 writeln('entrez le pourcentage des processeurs');
 readln(alpha);
 writeln(' Pourcentage des processeurs:   ', alpha : 10 : 8);
 writeln('entrez le nombre d iteration');
 readln(m);
 writeln('debut des essais');
 readln(deb);
 writeln('pas de 1 iteration');
 readln(pas);
 for l := 1 to m do
      begin
       n := deb + l * pas;
       nombre := round(alpha * n);
      temps_seq := 1.0 * (n - 1) * nombre;
       diag := 1;
      temps_courant := 1.0 * (n - 1);
      if n < 3 * nombre + 1 then
            begin
            writeln('Les valeurs de nombre et de n ne verifient pas la condition
```

```
        n>=3*nombre +1');
        a := 5;
        end
else
        if n = 3 * nombre + 1 then
                a := 2
        else
                if n > 3 * nombre + 1 then
                        a := 1;
if a = 1 then
        n1 := n
else
        if a = 2 then
                n1 := 2 * nombre + 1;
new(pt);
teteex := pt;
for j := nombre + 2 to n1 - 1 do
        begin
        pt^.ligne := 1;
        pt^.colonne := j;
        pt^.chemin := ch(1, j);
        new(qt);
        pt^.suiv := qt;
        pt := qt;
        end;
pt^.ligne := 1;
pt^.colonne := n1;
pt^.chemin := ch(1, n1);
pt^.suiv := nil;
new(pp);
tetepr := pp;
pp^.pred := nil;
for j := nombre + 1 downto 3 do
        begin
        pp^.temps := n - 1;
        pp^.ligne := 1;
        pp^.colonne := j;
        new(qp);
        pp^.suiv := qp;
        qp^.pred := pp;
        pp := qp;
        end;
pp^.temps := n - 1;
pp^.ligne := 1;
pp^.colonne := 2;
pp^.suiv := nil;
finpr := pp;
for etape := a to 4 do
```

```
        begin
        sortie := false;
        teteat := nil;

{Initialisation de l'étape 1}

        if etape = 1 then
            begin
            w := n - 3 * nombre + 1;
            processeur_libre := 0;
            end;

{Initialisation de l 'étape 2}

        if etape = 2 then
            begin
            diag := n - 3 * nombre;
            k := 1;
            w := n - nombre + 1;
            processeur_libre:=0;
            if a = 1 then
                begin
                processeur_libre := nombre;
                if not sortie then
                while (processeur_libre > 0) and (teteex <> nil) do
                    affectation;
                end
            end;

{Initialisation de l'étape 4}

    if etape = 4 then
        begin
        diag := n - nombre;
        w := n;
        miseex(n - nombre, n - nombre + 1);
        processeur_libre := nombre;
        if not sortie then
            while (processeur_libre > 0) and (teteex <> nil) do
                affectation;
        end;

{Boucle d'affectation des tâches aux processeurs, étapes 1, 2 et 4}

    if etape <> 3 then
        begin
        while (not sortie) and (tetepr <> nil) do
            begin
```

```
                i := finpr^.ligne;
                j := finpr^.colonne;
                temps_courant := finpr^.temps;
                if etape = 2 then
                    preparation;
                p1 := finpr;
                finpr := finpr^.pred;
                if finpr = nil then
                    tetepr := nil;
                p1^.suiv := nil;
                p1^.pred := nil;
                dispose(p1);
                if (j < n - nombre + 1) or (i < n - 3 * nombre - 1) or
                (i >= n - nombre) then
                    begin
                    if i < j - 1 then
                        if diag > i then
                            miseex(i + 1, j)
                        else
                            miseat(i + 1, j)
                    else
                        if i = j - 1 then
                            begin
                            if j <> w then
                            miseex(j, j)
                            else
                                sortie := true;
                            end
                        else
                            begin
                            diag := diag + 1;
                            libeat(i);
                            end;
                    end;
                    if (i = n - 3 * nombre - 1) and (j > n - nombre) then
                        processeur_libre := processeur_libre - 1;
                    processeur_libre := processeur_libre + 1;
                    if not sortie then
                        while (processeur_libre > 0) and (teteex <> nil) do
                            affectation;
                end;
            end;
```

{Initialisation de l'étape 3}

```
        if etape = 3 then
            begin
            k := 2;
```

```
new(pt);
teteex := pt;
for j := (n - nombre + 2) to n - 1 do
    begin
    pt^.ligne := n - 3 * nombre;
    pt^.colonne := j;
    pt^.chemin := ch(n - 3 * nombre, j);
    new(qt);
    pt^.suiv := qt;
    pt := qt;
    end;
pt^.ligne := n - 3 * nombre;
pt^.colonne := n;
pt^.chemin := ch(n - 3 * nombre, n);
pt^.suiv := nil;
new(pp);
tetepr := pp;
pp^.pred := nil;
pp^.temps := t[1] + 3 * nombre;
pp^.ligne := n - 3 * nombre;
pp^.colonne := n - nombre + 1;
pp^.suiv := nil;
finpr := pp;
temps_seq := temps_seq + 3 * nombre;
niv := n - 3 * nombre - 1;
```

{Boucle d'affectation des tâches aux processeurs, étape 3}

```
while (not sortie) and (tetepr <> nil) do
    begin
    if (finpr^.temps <= t[k]) or (k > nombre) then
        begin
        i := finpr^.ligne;
        j := finpr^.colonne;
        temps_courant := finpr^.temps;
        p1 := finpr;
        finpr := finpr^.pred;
        if finpr = nil then
            tetepr := nil;
        p1^.suiv := nil;
        p1^.pred := nil;
        dispose(p1);
        if j = n then
            begin
            niv := niv + 1;
            libeat(i + 2);
            end;
        if i < j - 2 then
```

```
                    begin
                    if (niv >= i - 1) then
                        miseex(i + 1, j)
                    else
                        miseat(i + 1, j);
                    end;
                processeur_libre := processeur_libre + 1;
                end
            else
                begin
                processeur_libre := processeur_libre + 1;
                temps_courant := t[k];
                k := k + 1;
                end;
            if (i = n - nombre - 1) then
                processeur_libre := processeur_libre - 1;
            if (i = n - nombre - 1) and (j = n) then
                sortie := true;
            if not sortie then
                begin
                while (processeur_libre > 0) and (teteex <> nil) do
                    begin
                    t1 := teteex^.ligne;
                    if temps_diag[t1] <= temps_courant then
                        affectation
                    else
                        begin
                        processeur_libre := processeur_libre-1;
                        writeln('Les contraintes ne sont pas
                        verifiees');
                        sortie := true;
                        end
                    end
                end
            end
        end;

{Bilan}

        if a <> 5 then
            begin
            writeln('Le temps d'exécution parallèle est:' ',temps_courant : 10 : 4);
            writeln('Le temps d'exécution séquentielle est:'   ',temps_seq: 10: 4);
            writeln('Le nombre de processeurs utilisé est:'    ',nombre : 10 : 1);
            seqreal := temps_seq;
            seqparal := temps_courant;
            seqparal := nombre * seqparal;
```

```
            efficacite := seqreal / seqparal;
            alpha1 := nombre / n;
            opt := 1 / (1 + 2 * (alpha1 * alpha1 * alpha1));
            writeln(' l'efficacité théorique est:        ',opt :10 : 8);
            writeln(' l'efficacité pratique est:         ',efficacité :10 : 8);
            end
        end
end.
```

6.2. COMMENTAIRE

En écrivant le programme précédent, nous nous sommes beaucoup inspirés du programme écrit par Jean Claude Konig [34] pour l'algorithme du chemin critique. Les résultats de la simulation de ce dernier sont présentés dans [65]. Avant de commenter le programme ci-dessus et ces procédures, nous précisons la définition de certaines variables du programme.

processeur: un processeur est défini par les indices de la dernière tâche exécutée par ce processeur et par aussi l'instant où l'exécution de cette tâche se termine. Notons cette dernière composante temps. Elle a pour rôle d'ordonner les processeurs.

tache: une tâche est définie par ses indices et son chemin. Ce dernier est une fonction qui ordonne les tâches à exécuter suivant l'algorithme parallèle défini.

temps_courant: c'est l'instant où l'exécution d'une tâche est terminée. On suppose que l'exécution de l'algorithme parallèle débute à l'instant 0.

nombre: c'est le nombre de processeurs utilisés. Il correspond dans nos notations habituelles à p.

Processeur_libre: cette variable a pour rôle de limiter le nombre de processeurs actifs.

Fonction chemin

Cette fonction est définie de l'ensemble des tâches à exécuter vers l'ensemble des réels positifs. Elle affecte les tâches aux processeurs, c'est un moyen de traduire l'algorithme parallèle à exécuter. Le graphe à étudier est partagé en quatre parties notées partie (0), partie (1), partie (2) et partie (3). Elles se composent respectivement des tâches $\{T_{kj},\ 1\leq k\leq n-p-1,\ k\leq j\leq\leq n\}\cup\{T_{n-3p,n-3p}\}$, $\{T_{kj},\ n-3p\leq k\leq n-p-1,\ k<j\leq n-p\}\cup\{T_{kk},\ n-3p<k\leq n-p\}$, $\{T_{kj},\ n-3p\leq k\leq n-p-1,\ k<j\leq n\}$ et $\{T_{kj},\ n-p\leq k\leq n,\ k<j\leq n\}\cup\{T_{kk},\ n-p+1\leq k\leq n-1\}$. Les quatre parties correspondent respectivement aux quatre étapes du programme. A chaque étape, on définit la fonction chemin ch.

Etape1

si i=j ch(i,j) = n^2

si i<j ch(i,j) = (n-i)*n + n-j

Etape 2

si i=j ch(i,j) = n^2

si i<j ch(i,j) =
- (n-i-1)*n + n-j si j=n-p et i-n+3p pair
- n^2 si j=n-p et i-n+3p impair
- (n-i)*n + n-j si non

Etape 3

ch(i,j) = (n-i)*n + n-j

Etape 4

si i=j ch(i,j) = n^2

si i<j ch(i,j) = (n-i)*n + n-j

La fonction ch des étapes 1, 3, 4 assure l'affectation des tâches respectivement des parties (0), (2) et (3) aux processeurs comme l'indique l'algorithme Glouton. En effet, l'exécution de la tâche diagonale débute dès que celle-ci est libérée des contraintes d'ordonnancement: la valeur de la fonction ch au point (j,j) est égale à n^2, valeur qui est supérieure à toutes les autres valeurs de la même fonction. Ce qui assure que la tâche diagonale est en tête de la liste des tâches exécutables quand elle se libère des contraintes d'ordonnancement. Pour i<j, la fonction ch est strictement décroissante par rapport à j, avec i constant. Sur un même niveau, les tâches s'exécutent de gauche à droite: c'est à dire l'exécution d'une tâche T_{ij} aura lieu au plus tôt au même instant que celle de sa précédente située sur le même niveau. Il est clair qu'une tâche située sur un niveau quelconque est affectée avant celle située sur le niveau suivant. Pour l'étape 2, la fonction ch définie traduit l'algorithme construit pour la partie (1) du graphe. En effet, les 2(p-k) -1 premières tâches du premier niveau du bloc B_k et la tâche diagonale du même niveau (le premier niveau du bloc B_k privé de la dernière tâche) s'exécutent par p-k processeurs suivant un algorithme Glouton, c'est pourquoi la fonction ch définie pour ces dernières tâches est la même que celle définie pour les régions (0), (1), (3). Même explication pour l'exécution du niveau suivant du bloc B_k privé de la dernière tâche. Pour les tâches de la dernière colonne de la partie (1),

c'est à dire les tâches $T_{k,n-p}$ où $n-3p \leq k < n-p$, leur affectation aux processeurs par la fonction ch s'effectue comme suit: le chemin de la première tâche d'un bloc quelconque est égal à (n-i-1)*n + n-j. Ce qui entraîne que l'exécution de cette tâche ne débute qu'après la fin de l'exécution des p premières tâches du second niveau du même bloc. La seconde tâche se libère de la contrainte verticale dès que l'exécution de la première se termine et puisque son chemin est n^2, alors elle sera affectée au processeur juste après la fin de l'exécution de la tâche précédente sur la même colonne.

Procédure place

Elle ordonne la liste des p processeurs suivant l'ordre dans lequel ils se libèrent. C'est grâce à la troisième composante (la composante temps) d'un processeur qui est l'instant où l'exécution de la dernière tâche associée à un processeur se termine. Les deux processeurs extrêmes de la liste s'appellent **tetepr** et **finpr**. **tetepr** (respectivement **finpr**) est celui qui possède la plus grande (respectivement la plus petite) longueur. Le temps d'un processeur croit de **finpr** à **tetepr**. Cette procédure permet de placer les processeurs sur une liste suivant l'ordre dans lequel ils se libèrent. La procédure telle qu'elle est écrite place le processeur pp0. On a les différents cas possibles suivants:

1. **tetepr**=nil

La liste ne contient aucun processeur. Dans ce cas **tetepr** et **finpr** prend la valeur de **pp0**.

2. **tetepr** ≠nil

2.1. temps(**tetepr**)≤temps(**pp0**)

Dans ce cas **tetepr** prend la valeur de **pp0** et l'ancien contenu de **tetepr** se place juste après **tetepr**.

2.2. temps(**tetepr**)>temps(**pp0**)

Le processeur **pp0** s'est libéré après celui de **tetepr**. La place de **pp0** se situe après celle du processeur en tête de la liste. Pour cela, on cherche deux processeurs **pr1** et **pr2** de la liste vérifiant:

temps(**pr2**) < temps(**pp0**) ≤ temps(**pr1**)

S'il n'existe pas de processeurs vérifiant l'inégalité ci-dessus, le processeur **pp0** se place à la fin de la liste de processeurs.

Procédure miseex

Les tâches sont ordonnées dans une liste à l'aide d'une fonction chemin (function ch) qui traduit l'algorithme parallèle. La tâche en tête de la liste possède le plus long chemin par rapport à celles qui la précédent sur la même liste. Les tâches

sont alors ordonnées suivant leur chemin décroissant. Démarche similaire à la procédure place.

Procédure miseat

La procédure miseat met en évidence une liste de tâches en attente . Ces dernières sont libérées des contraintes verticales mais pas des contraintes diagonales. Pour les transmettre en exécution, on se sert de la procédure libeat et ceci aura lieu dès que les contraintes sont vérifiées.

Corps du programme

Le programme ne fonctionne que si $n \geq 3p+1$, c'est pourquoi on essaie au début de comparer $3p+1$ et n. Si $n=3p+1$, l'étape 1 n'existe pas, le programme ne commence à tourner qu'à partir de l'étape 2. On crée les listes de processeurs et de tâches: la liste de p processeurs est définie à partir des p premières tâches à exécuter tandis que la liste des tâches, est définie à partir des tâches restantes (respectivement des p tâches suivantes) du niveau N_1 si $n>3p+1$ (respectivement si $n=3p+1$). A titre de vérification seulement, on tient compte des contraintes d'ordonnancement. Une tâche n'est mise en exécution que si les contraintes d'ordonnancement sont respectées: d'une part, la tâche $T_{i+1,j}$ ne se met en exécution que si l'exécution de la tâche T_{ij} est terminée, d'autre part, la variable **diag** doit être supérieure à i. Si cette dernière condition n'est pas vérifiée, la tâche $T_{i+1,j}$ sera affectée à la liste d'attente en attendant la fin de l'exécution de la tâche diagonale correspondante, voir figure 15. A l'étape 2, on sauvegarde les instants où l'exécution des tâches diagonales de la partie (1) est terminée dans le tableau **temps_diagonale** afin de vérifier à l'étape 3 que les contraintes diagonales sont respectées. De même on sauvegarde les instants où les processeurs cessent de fonctionner à l'étape 2 (dans le tableau **tab**). A l'étape 3, les processeurs ne sont pas tous actifs au début de cette étape. Un processeur P_i ne peut fonctionnner dans cette étape que s'il existe un proceseur P_j ($j<i$) en mode de fonctionnement à l'étape 3 vérifiant $tab[i] \geq temps(P_j)$.

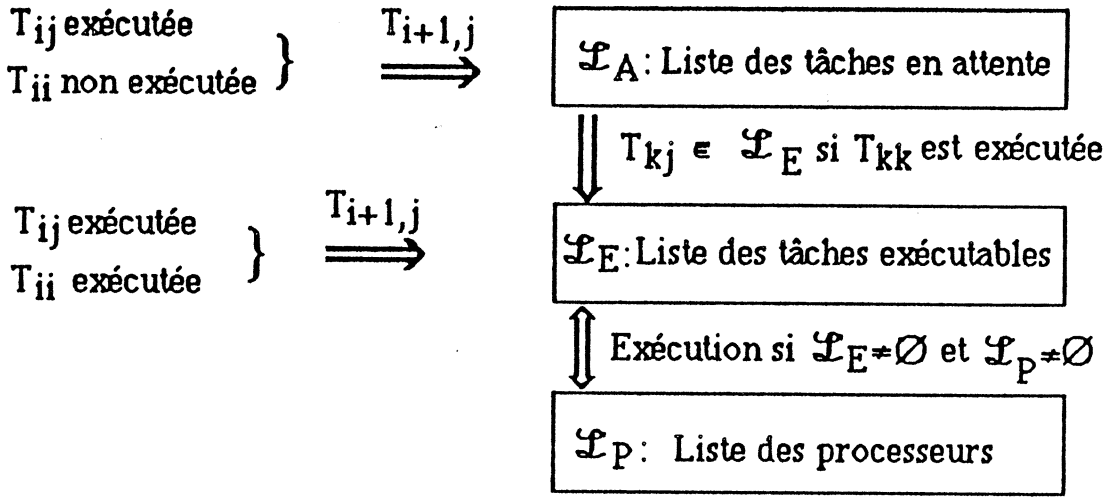


Figure 15 : Fonctionnement du programme

6.3. RESULTATS DE SIMULATION

Le pourcentage de l'erreur relative de l'efficacité est donné par la formule

$$\frac{|e_t - e_e|}{e_t} * 100$$

où |x| est la valeur absolue de x, e_t et e_e sont respectivement les efficacités théorique et expérimentale. Les valeurs de e_e sont données par le programme précédent tandis que

$$e_t = \frac{1}{1 + 2\alpha^3}$$

Pour calculer cette efficacité théorique, nous avons négligé les facteurs en $O(n^2)$ sur le temps séquentiel et O(n) sur le temps parallèle en plus nous n'avons pas tenu compte des temps des synchronisation des processeurs aux niveaux N_{n-3p} et N_{n-p}. La différence entre les deux efficacités est dûe aux termes négligés. D'après les exemples simulés, on peut dire que l'efficacité théorique ne diffère pas de l'efficacité expérimentale: l'erreur est de l'ordre de 10^{-3} si α=0.05, de 10^{-2} si α=0.2. L'erreur relative ne dépasse pas 0.03% pour α=0.05, elle est inférieure à 0.01% si la taille de la matrice est supérieure à 100 (voir figure 16). Dans le cas où α=0.2, elle est limitée à 0.3%. De ces deux exemples, nous concluons que

i) si la taille de la matrice est assez grande, l'erreur diminue: les courbes des figures 16 et 17 montrent que si n>150, l'erreur relative ne dépasse pas 0.05%. Ce qui montre la bonne adéquation du modèle pour des valeurs de n relativement faibles.

ii) si le nombre de processeurs est assez petit par rapport à la taille de la matrice à décomposer, la valeur théorique s'approche plus de la valeur expérimentale: les valeurs de l'efficacité théorique sont plus proches des valeurs expérimentales pour α=0.05 que pour α=0.2. L'écart entre les valeurs théoriques et expérimentales croit avec le nombre de processeurs.

La conclusion ii) est vraie aussi pour la courbe de la figure 18. Jusqu'à 15 processeurs, les valeurs théoriques et expérimentales coincident, tandis que pour un nombre de processeurs proche de 50, l'erreur relative passe à 0.5%.

D'une façon générale, on peut dire que les termes négligés n'affectent pas l'évaluation des performances et les résultats asymptotiques remarquablement concordent avec les résultats expérimentals.

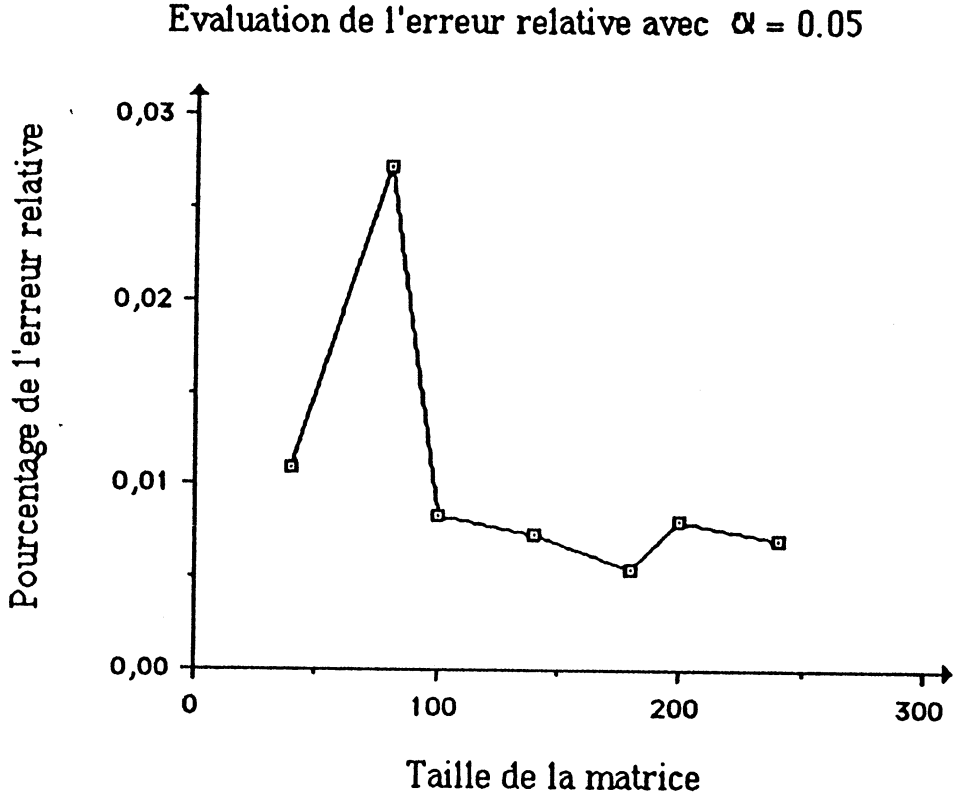


Figure 16

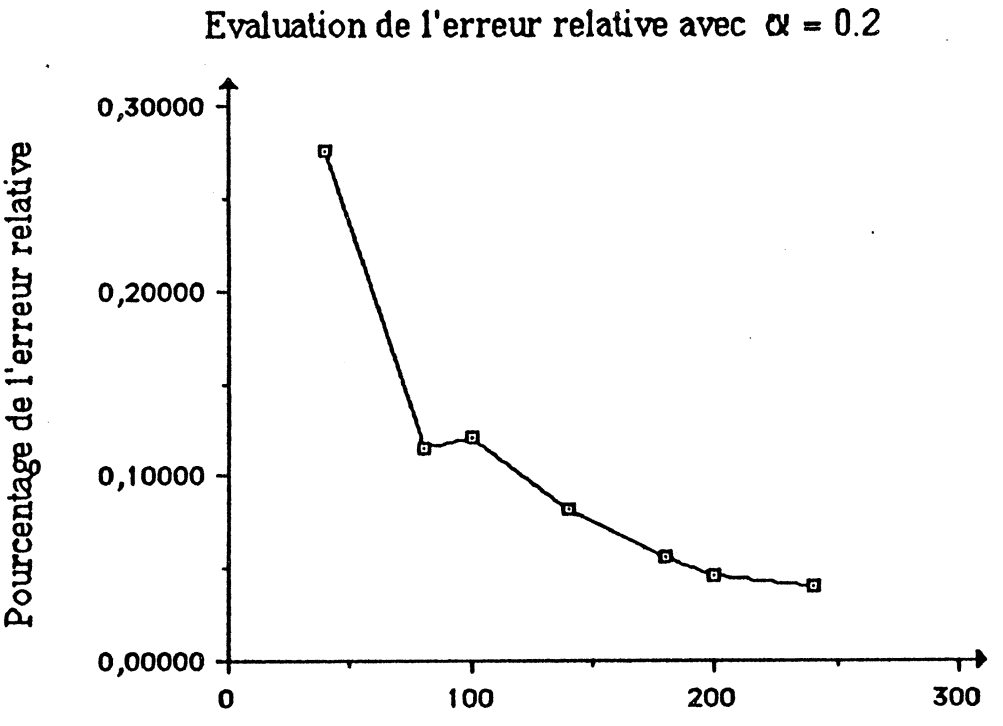


Figure 17

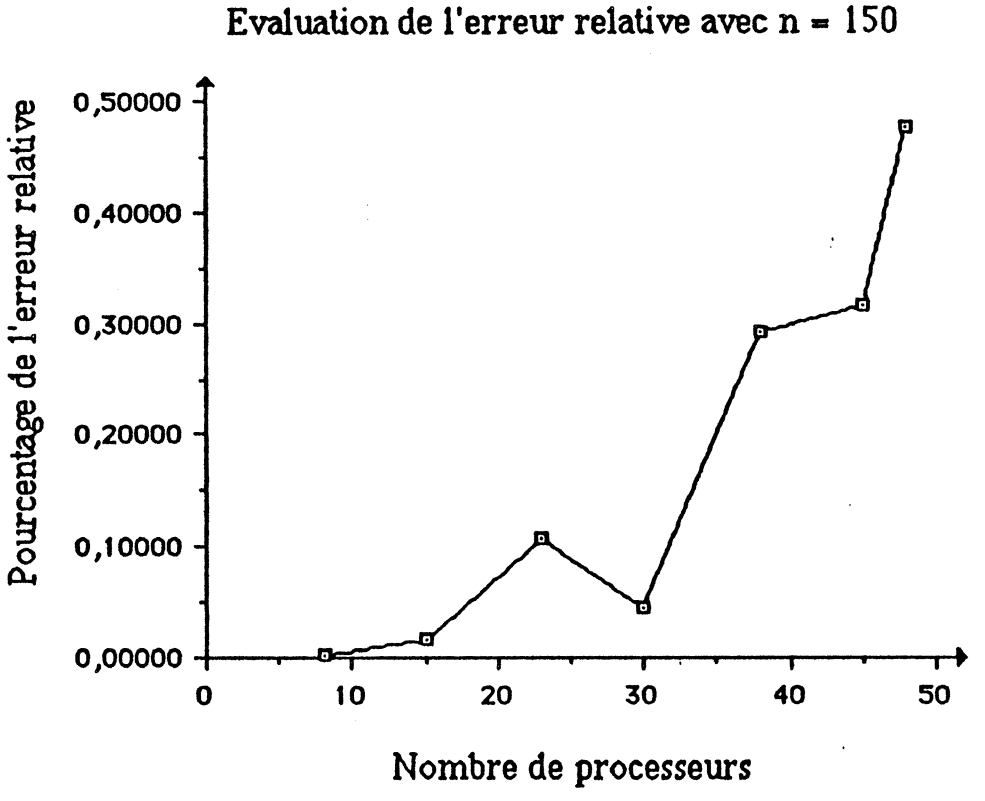


Figure 18

Chapitre IV

INVERSION D'UNE MATRICE EN PARALLELE

1. INTRODUCTION

Dans ce chapitre on parallélise l'algorithme de Rote [51] pour le problème général du chemin algébrique. Cet algorithme permet notamment de calculer l'inverse d'une matrice, de déterminer la fermeture reflexive et transitive d'une relation binaire, et de résoudre le problème des plus courtes distances dans un graphe pondéré. Il comporte trois phases, qui s'interprètent naturellement dans le cas de l'inversion d'une matrice:
- phase 1: décomposition de Gauss d'une matrice en produit de deux matrices triangulaires.
- phase 2: inversion des deux matrices obtenues dans la phase 1.
- phase 3: multiplication des deux matrices inverses.

Nous rappelons l'algorithme de Rote à la section 2, en analysant successivement la parallélisation de chacune des trois phases. Ces dernières sont parallélisées sans duplication puis avec duplication temporaire et enfin avec duplication complète. La version de Rote avec duplication temporaire, nous a conduit à l'étude de nouveaux graphes. Ces derniers possèdent des tâches dont la longueur n'est pas constante par niveau. Nous étudions également la parallélisation de l'algorithme proposé dans [47], qui résoud le problème du chemin algébrique via un schéma de diagonalisation.

Nous rappelons ensuite (section 5) la parallélisation de l'inverse d'une matrice à l'aide des méthodes de Gauss, Jordan et Huard . Enfin, nous comparons les complexités de ces différentes versions.

2. LE PROBLEME DU CHEMIN ALGEBRIQUE

Le concept de chemin algébrique permet la formulation abstraite et unifiée d'une vaste classe de problèmes, englobant l'inversion d'une matrice, le calcul des plus courtes distances dans un graphe pondéré, et la détermination de la fermeture réflexive et transitive d'une relation binaire. Dans un article récent, Rote [51] introduit l'algorithme d'élimination de Gauss-Jordan pour le problème général du chemin algébrique (problème APP, mis pour Algebraic Path Problem). Rote présente une architecture systolique hexagonale de $(n+1)^2$ processeurs élémentaires qui permet de résoudre toute instance du problème APP en temps 7n-2, l'unité étant le temps nécessaire pour réaliser une multiplication suivie d'une addition dans l'algèbre sous-jacente. Nous nous proposons d'étudier ici la parallélisation de l'algorithme de Rote dans le cadre du modèle SIMD/MIMD à mémoire partagée.

2.1. DEFINITION

Le problème APP est défini comme suit [51]: étant donné un graphe pondéré G = (V,E,w), où V est l'ensemble fini des sommets, E l'ensemble des arcs, et w : E -> H une application à poids pris dans un semi-anneau complet (H,(+),(x),*) de zéro (0) et d'unité (1), trouver pour toutes les paires de sommets (i,j) les valeurs

$$d_{ij} = \underset{p \in M_{ij}}{(+)} w(p),$$

où M_{ij} représente l'ensemble de tous les chemins de i à j.

On associe au graphe G = (V,E,w) la matrice d'ordre n, A = (a_{ij}), où a_{ij}=w(i,j) si (i,j)∈E et a_{ij}=0 sinon. On note $M_{ij}^{(k)}$ l'ensemble de tous les chemins de i à j qui ne contiennent comme sommets intermédiaires que des sommets d'indice x avec 1≤x≤k. Ainsi

$$a_{ij}^{(k)} = \underset{p \in M_{ij}^{(k)}}{(+)} w(p),$$

est égal aux valeurs successives de a_{ij} que nous voulons calculer, partant de la valeur initiale $a_{ij}^{(0)}=a_{ij}$ jusqu'à $a_{ij}^{(n)} = d_{ij}$.

2.2. ALGORITHME DE ROTE

L'algorithme général que présente Rote pour la résolution d'une instance quelconque du problème APP utilise seulement les opérations (+), (x) et * du semi-anneau H. Il se divise en trois phases:

```
{ ALGORITHME de ROTE }

{ phase 1 : calcul des a_ij^(j) pour i>j et des a_ij^(i-1) pour i≤j }
pour i := 1 à n
pour j := 1 à n
début
     pour k := 1 à min(i,j)-1
          a_ij^(k) := a_ij^(k-1) (+) a_ik^(k) (x) a_kj^(k-1) ;
(A)  si i=j alors a_ij^(i) := (a_ii^(i-1))* ;
(B)  si i>j alors a_ij^(j) := a_ij^(j-1) (x) a_jj^(j) ;
     fin ;

{ phase 2 : calcul des a_ij^(k) où k=j si i≤j et k=i-1 si i>j }
pour i := 1 à n
pour j := 1 à n
début
     si i<j alors a_ij^(i) := a_ii^(i) (x) a_ij^(i-1) ;
     pour k := min(i,j)+1 à max(i,j)-1
          a_ij^(k) := a_ij^(k-1) (+) a_ik^(k) (x) a_kj^(k-1) ;
(C)  si i<j alors a_ij^(j) := a_ij^(j-1) (x) a_jj^(j) ;
fin ;

{ phase 3 : calcul des a_ij^(n) }
pour i := 1 à n
```

pour j := 1 **à** n
début
    si i>j **alors** $a_{ij}^{(i)} := a_{ii}^{(i)}$ (x) $a_{ij}^{(i-1)}$;
    pour k := max(i,j)+1 **à** n
        $a_{ij}^{(k)} := a_{ij}^{(k-1)}$ (+) $a_{ik}^{(k)}$ (x) $a_{kj}^{(k-1)}$;
fin ;

Nous explicitons brièvement trois instances particulièrement importantes de l'APP et renvoyons à [47, 51] pour plus de précisions:

(i) inversion d'une matrice: A est une matrice réelle, (+) et (x) sont les opérations usuelles sur les nombres réels et l'opération * est définie par (**si** c≠1 **alors** $c^* :=$ 1/(1-c)): l'algorithme correspond ici à l'élimination de Gauss bien connue et délivre en sortie la matrice $(I-A)^{-1}$. Des modifications immédiates sur les lignes (A), (B) et (C) permettent de calculer directement A^{-1}:

si i=j **alors** $a_{ij}^{(i)} := 1 / a_{ii}^{(i-1)}$; (A')
si i>j **alors** $a_{ij}^{(j)} := - a_{ij}^{(j-1)}$ (x) $a_{jj}^{(j)}$; (B')
si i<j **alors** $a_{ij}^{(j)} := - a_{ij}^{(j-1)}$ (x) $a_{jj}^{(j)}$; (C')

(ii) recherche des plus courtes distances dans un graphe pondéré: dans ce cas, les poids a_{ij} sont choisis dans $H = \mathbb{R} \cup \{-\infty,+\infty\}$ (en particulier $a_{ij}=+\infty$ si l'arc (i,j) n'existe pas), (+) est le miminum sur H, (x) est l'addition sur les réels étendue à H (en particulier $-\infty$ (x) $+\infty = +\infty$), et * correspond à (**si** c≥0 **alors** $c^* := 0$ **sinon** $c^* := -\infty$)

(iii) calcul de la fermeture réflexive et transitive d'une relation binaire: les a_{ij} deviennent des variables logiques, (+) et (x) sont respectivement l'addition et la multiplication booléenne, et * consiste en une mise à "vrai" ($c^* := 1$ pour tout c).

Dans ce qui suit, nous étudions la parallélisation de chacune des trois phases de l'algorithme précédent. Notons que dans le cas de l'inversion d'une matrice réelle A, ces trois phases ont une interprétation naturelle. La première phase correspond en effet au calcul de la décomposition LU de A, si ce n'est que les éléments diagonaux $a_{ii}^{(i-1)}$ de U ont déjà été remplacés par leurs inverses $a_{ii}^{(i)}=1/a_{ii}^{(i-1)}$. Dans la deuxième phase, on calcule des inverses L^{-1} et U^{-1} de L et U, et le produit U^{-1} L^{-1} s'effectue au cours de la dernière phase.

3. PARALLELISATION SANS DUPLICATION

3.1. PARALLELISATION DE LA PHASE 1

Nous écrivons la première phase de l'algorithme de Rote sous la forme suivante:

pour i := 1 à n
pour j := 1 à n
début
  exécuter $T_{i,j}$:
    < **pour** k := 1 à min(i,j)-1
      $a_{ij}^{(k)} := a_{ij}^{(k-1)}$ (+) $a_{ik}^{(k)}$ (x) $a_{kj}^{(k-1)}$;
    si i=j **alors** $a_{ij}^{(i)} := (a_{ii}^{(i-1)})^*$;
    si i>j **alors** $a_{ij}^{(j)} := a_{ij}^{(j-1)}$ (x) $a_{jj}^{(j)}$; >
fin ;

La tache T_{ij} s'exécute en 2(i-1) unités de temps si i<j, et en 2(j-1)+1 unités de temps si i≥j. Les contraintes d'ordonnancement sont les suivantes:

(A)	$T_{ij} << T_{i,j+1}$	$1 \le j \le i-1$	$3 \le i \le n$
(B)	$T_{i,i-1} << T_{i,j}$	$i \le j \le n$	$2 \le i \le n$
(C)	$T_{i,i} << T_{j,i}$	$i+1 \le j \le n$	$1 \le i \le n$
(D)	$T_{i,j} << T_{i+1,j}$	$2 \le i$	$j \le n$

L'allure du graphe des tâches est donnée par la figure 1. La figure 2 présente l'étape k de l'algorithme. Il est intéressant de noter que l'exécution parallèle de l'algorithme nous fait passer d'une forme IJK à une forme KIJ (avec les notations de [15]).

Le temps séquentiel nécessaire pour exécuter la phase 1 est $2n^3/3 + O(n^2)$. Le plus long chemin du graphe est { $T_{11}, T_{21}, T_{22}, \ldots, T_{j,j}, T_{j+1,j}, \ldots, T_{n,n-1}, T_{n,n}$}. Le temps d'exécution de tout algorithme est minoré par T_{opt}. Pour cet algorithme $T_{opt} = 2n^2 + O(n)$. D'autre part, le diamètre du graphe est D=n-1, et il très facile de construire un algorithme s'exécutant en temps T_{opt} avec n-1 processeurs: l'efficacité asymptotique est alors $e_{n-1,\infty} = 1/3$.

D'une manière générale, le graphe des tâches (voir figures 1 et 2) correspond à la forme IJK de l'élimination de Gauss qui a été étudiée en détail dans [8]. Nous renvoyons au chapitre II pour la description d'un algorithme s'exécutant avec $p = \alpha n$ processeurs, où α est un nombre arbitraire dans]0,1[.
Les résultats sont résumés dans la

Proposition 4.1. (i) $T_{opt} = 2n^2 + O(n)$ et $e_{n-1,\infty} = 1/3$

(ii) Pour $p = \alpha n$ avec $\alpha \in]0,1[$, il existe un algorithme d'efficacité asymptotique $e_{p,\infty} = 1 / (-\alpha^3 + 3\alpha^2 + 1)$.

Seul l'algorithme trivial avec n-1 processeurs est optimal. Dans le cas général où l'on dispose de $p=\alpha n$ processeurs avec $\alpha \in]0,1[$, la construction d'un algorithme optimal est un problème ouvert.

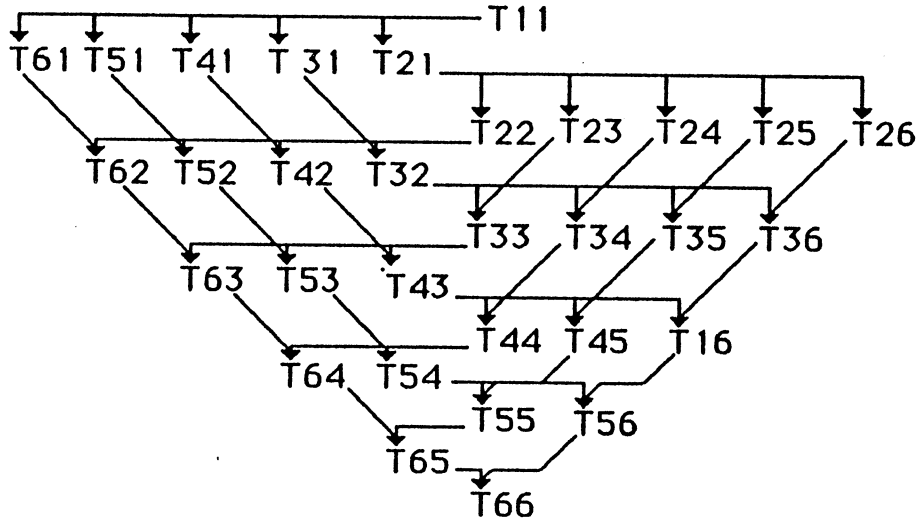


Figure 1: Graphe d'ordonnancement pour la phase 1 de l'algorithme de Rote pour une matrice de taille 6x6.

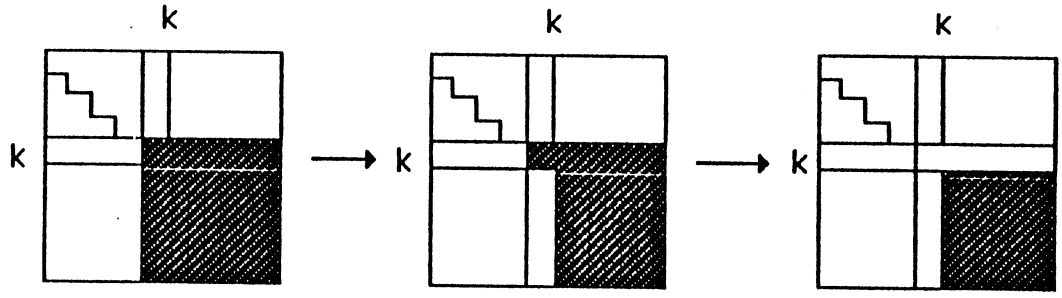


Figure 2: Exécution en parallèle de la phase 1 de l'algorithme de Rote.

3.2. PARALLELISATION DE LA PHASE 2

La deuxième phase de l'algorithme de Rote s'écrit:

pour i := 1 **à** n
pour j := 1 **à** n
début
    si i<j **alors** $a_{ij}^{(i)} := a_{ii}^{(i)}$ (x) $a_{ij}^{(i-1)}$;
    pour k := min(i,j)+1 **à** max(i,j)-1
        $a_{ij}^{(k)} := a_{ij}^{(k-1)}$ (+) $a_{ik}^{(k)}$ (x) $a_{kj}^{(k-1)}$;
    si i<j **alors** $a_{ij}^{(j)} := a_{ij}^{(j-1)}$ (x) $a_{jj}^{(j)}$;
fin ;

Les calculs relatifs aux couples (i,j) avec i>j sont indépendants de ceux relatifs aux couples (i,j) avec i≤j. Cela s'explique aisément par l'interprétation proposée en termes matriciels: pour i>j les calculs correspondent à l'inversion d'une matrice triangulaire inférieure dont la diagonale est l'unité:

pour i := 1 **à** n
pour j := 1 **à** i-2
début
exécuter $T_{i,j}$:
< **pour** k := j+1 **à** i-1
$a_{ij}^{(k)} := a_{ij}^{(k-1)}$ (+) $a_{ik}^{(k)}$ (x) $a_{kj}^{(k-1)}$; >
fin ;

La longueur d'une tâche $T_{i,j}$ est 2(i-j-1) unités de temps.

Si i≤j les calculs correspondent à l'inversion d'une matrice triangulaire supérieure:

pour i :=1 **à** n
pour j := i+1 **à** n
début
exécuter $R_{i,j}$:
< $a_{ij}^{(i)} := a_{ii}^{(i)}$ (x) $a_{ij}^{(i-1)}$;
pour k := i+1 **à** j-1
$a_{ij}^{(k)} := a_{ij}^{(k-1)}$ (+) $a_{ik}^{(k)}$ (x) $a_{kj}^{(k-1)}$
$a_{ij}^{(j)} := a_{ij}^{(j-1)}$ (x) $a_{jj}^{(j)}$; >
fin ;

Une tâche $R_{i,j}$ s'exécute en 2(j-i) unités de temps. Séquentiellement, chacun des deux algorithmes a un coût de $n^3/3+O(n^2)$ unités de temps.

3.2.1. Graphes et contraintes

L'algorithme correspond à l'inversion d'une matrice triangulaire inférieure à diagonale unité sans duplication de la matrice par stockage annexe.

Les contraintes d'ordonnancement sont les suivantes:

(A) $T_{i,j} << T_{i+1,j}$ pour $3 \le i \le n$ $1 \le j \le i-2$

(B) $T_{i,j} << T_{i,j+1}$ pour $4 \le i \le n$ $1 \le j \le i-2$.

Le graphe a l'allure suivante:

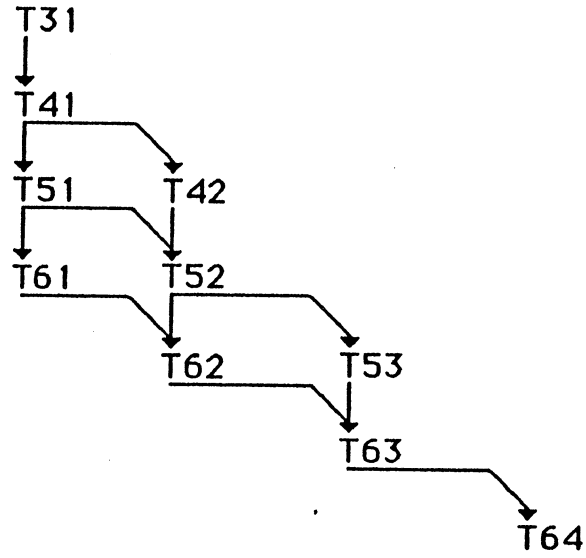


Figure 3 : Graphe d'inversion d'une matrice triangulaire inférieure à diagonale unité de taille 6x6 sans la dupliquer.

L'algorithme procède comme indiqué figure 4.

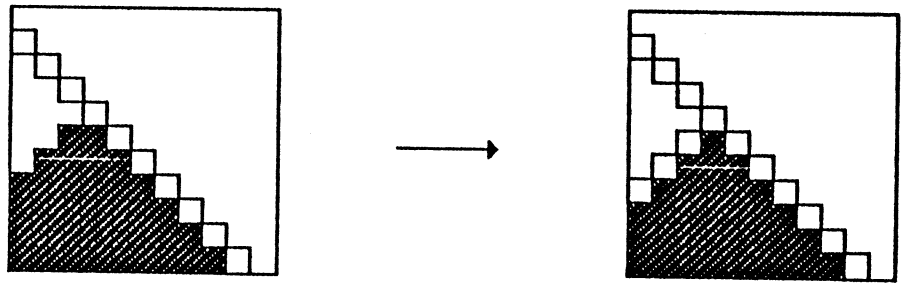


Figure 4: Inversion en parallèle d'une matrice triangulaire inférieure à diagonale unité.

L'algorithme correspond à l'inversion d'une matrice triangulaire supérieure sans duplication de la matrice par stockage annexe. Les contraintes d'ordonnancement sont:

(A) $R_{i,j} << R_{i,j+1}$ pour $1 \leq i \leq n-1$ et $i+1 \leq j \leq n$,

(B) $R_{i,j} << R_{i+1,j}$ pour $1 \leq i \leq j-1$ et $2 \leq j \leq n$.

L'allure du graphe est:

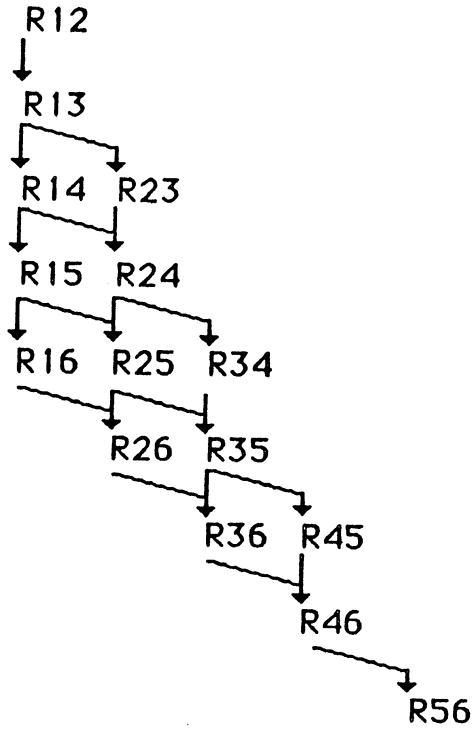


Figure 5: Graphe d'inversion d'une matrice triangulaire supérieure de taille 6x6 sans la dupliquer.

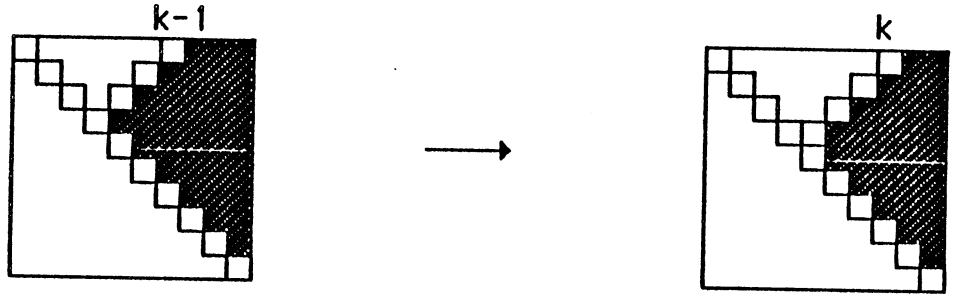


Figure 6: Inversion en parallèle d'une matrice triangulaire supérieure sans la dupliquer.

On remarque que les deux graphes obtenus sont identiques, à l'exception des tâches diagonales supplémentaires pour le second. Nous nous restreignons donc à l'inversion d'une matrice triangulaire supérieure.

3.2.2. Inversion d'une matrice triangulaire supérieure

i) détermination de T_{opt}

Le plus long chemin du graphe est $\{R_{12},R_{13},R_{14},...,R_{1n},R_{2n},...,R_{n,n-1}\}$ d'où la valeur

$$T_{opt} = \sum_{i=2}^{n} 2(i-1) + \sum_{j=2}^{n-1} 2(n-j) = 2n^2 + O(n)$$

Il est évident de construire un algorithme s'exécutant en temps T_{opt} avec

$\lceil n/2 \rceil$ processeurs puisque le diamètre du graphe est inféreur ou égal à $\lceil n/2 \rceil$, l'efficacité asymptotique est alors

$$e_{\lceil n/2 \rceil,\infty} = (n^3/3)/n^3 = 1/3.$$

où $\lceil x \rceil$est la partie entière supérieure du réel x.

ii) cas où $p = \alpha n$ avec $\alpha \in]0,1/2]$

Les tâches d'une diagonale $j \geq 2$ sont les tâches $R_{i,j}$ avec $1 \leq i \leq j-1$. Nous présentons un algorithme dont l'exécution s'effectue par diagonales. On dispose de p processeurs notés $P_1, P_2, \ldots, P_p$. Le processeur P_k exécute les diagonales k+1, k+1+p, k+1+2p,..., k+1+mp où $m \leq (n-p-1)/p$. L'exécution de la diagonale i+1 ne peut commencer que si celle de la tâche $R_{1,i}$ est terminée. Notons T_i, $\mathcal{C}_i$ et $\tau_{i,p}$ les temps mis pour exécuter respectivement la diagonale i, la diagonale i privée de la tâche $R_{1,i}$ et les p-1 tâches successives de la première colonne à partir de $R_{1,i+1}$. Si $\mathcal{C}_i \leq \tau_{i,p}$ c'est à dire $i \leq \beta = (1 + 2p + \sqrt{8p^2-8p+9})/2$, l'exécution de la diagonale j avec $2 \leq j \leq \beta+p$ débute à l'instant δ_j où $\delta_j = \Sigma_{2 \leq l \leq j-1} Ex(R_{1,i})$. Dans le cas contraire, c'est à dire $\mathcal{C}_i > \tau_{i,p}$, ce qui correspond à $i > \beta+p$, l'exécution de la diagonale j où j=i+p commence à l'instant δ_j où $\delta_j = \delta_i + T_i$.

Montrons que les contraintes d'ordonnancement sont respectées :

- les contraintes (B) sont verifiées puique l'exécution s'effectue par diagonales

- les raisons suivantes assurent que les contraintes (A) sont toujours vérifiées:

1) Le nombre de tâches situées sur la diagonale i+1 est supérieur au nombre de tâches situées sur la diagonale i.

2) L'exécution de la tâche $R_{1,i+1}$ commence après la fin de l'exécution de la tâche $R_{1,i}$.

3) La tâche $R_{i,j+1}$, qui est située sur la même ligne que la tâche $R_{i+1,j}$ et sur la diagonale suivante, possède un temps d'exécution supérieur, c'est à dire $Ex(R_{i,j+1}) > Ex(R_{i+1,j})$.

Ces trois raisons assurent que l'exécution des tâches $R_{i+1,j}$ débute après la fin de l'exécution des tâches $R_{i,j}$.

EXECUTION EN TEMPS T_{opt}

Pour que cet algorithme s'exécute en temps T_{opt} il faut que le temps mis pour exécuter la diagonale i privée de la tâche $R_{1,i}$ soit inférieur ou égal au temps mis pour exécuter les tâches $R_{1,i+1}$, $R_{1,i+2}$, ..., $R_{1,i+p}$ de la première colonne. On a

$$T_i = \sum_{j=1}^{i-1} 2(i - j) = (i-1)i$$

$$\mathcal{T}_i = T_i - 2(i - 1) = (i -3)i$$

$$\tau_{i,p} = \sum_{k=i+1}^{i+p-1} 2(k-1) = (p-1)(2i+p-2)$$

Si $\mathcal{T}_i \leq \tau_{i,p}$ pour tout $i \leq n-p$, l'algorithme s'exécute en temps T_{opt}.

$$\mathcal{T}_i - \tau_{i,p} = i^2 - (1 + 2p)i - p^2 + 3p - 2$$

$$\text{Si } i \in [2,\beta] \text{ où } \beta = \frac{1 + 2p + \sqrt{8p^2 - 8p + 9}}{2} \text{ on a} \qquad \mathcal{T}_i - \tau_{i,p} \leq 0.$$

Pour avoir cette relation vraie pour tout $i \leq n-p$ il faut que $\beta \geq n-p$, ce qui entraîne

$$p \in [p_1, \frac{n}{2}] \quad \text{où} \quad p_1 = \frac{2(n - 2) - \sqrt{2n(n - 3)}}{2}$$

Si $p = \alpha n$ et pour n assez grand on a $\alpha \in [(2-\sqrt{2})/2, 1/2]$, l'efficacité asymptotique est donnée par $e_{\alpha n,\infty} = 1/(6\alpha)$.

Proposition 4.2. i) $T_{opt} = 2n^2 + O(n)$

ii) Il existe un algorithme s'exécutant en temps T_{opt} avec αn processeurs où $\alpha \in [(2-\sqrt{2})/2, 1/2[$

iii) L'efficacité asymptotique est donnée par

$$e_{\alpha n,\infty} = 1/(6\alpha)$$

iv) Pour $\alpha = (2-\sqrt{2})/2$, l'efficacité asymptotique est égale à $(2+\sqrt{2})/6$.

EXECUTION AVEC αn PROCESSEURS $(0 \le \alpha \le (2-\sqrt{2})/2)$

Le temps mis pour exécuter l'algorithme décrit précédemment est donné par la somme du temps optimal t_1 mis pour exécuter les diagonales 2, 3, ..., $\lfloor \beta \rfloor$-1 (on note $\lfloor \beta \rfloor$ la partie entière inférieure de β), et du temps t_2 mis pour exécuter les diagonales restantes. Soit $k=(n-\lfloor \beta \rfloor)/p$, on calcule le temps mis pour exécuter respectivement les diagonales $\lfloor \beta \rfloor$, $\lfloor \beta \rfloor+p$, $\lfloor \beta \rfloor+2p$,..., $\lfloor \beta \rfloor+kp$, et le temps total est $T_p= t_1+t_2$ où

$$t_1 = \sum_{i=2}^{\lfloor \beta \rfloor - 1} 2(i-1) = (\lfloor \beta \rfloor - 1)(\lfloor \beta \rfloor - 2)$$

$$t_2 = \sum_{j=0}^{k} (\lfloor \beta \rfloor + jp - 1)(\lfloor \beta \rfloor + jp) = (k+1)\lfloor \beta \rfloor^2 + k(k+1)p\lfloor \beta \rfloor + \frac{k(k+1)(2k+1)}{6}p^2 + O(n)$$

Pour $p = \alpha n$, $\beta = (1+\sqrt{2})\alpha n$, $k = \dfrac{1-(1+\sqrt{2})\alpha}{\alpha}$ et n assez grand on a

$$T_p = \frac{(12+7\sqrt{2})\alpha^3 + \alpha^2 + 3\alpha + 2}{6\alpha} n^2 + O(n)$$

L'efficacité asymptotique est donnée par :

$$e_{\alpha n,\infty} = \frac{2}{(12+7\sqrt{2})\alpha^3 + \alpha^2 + 3\alpha + 2}$$

Le tableau suivant illustre l'efficacité obtenue:

α	0,29	0,2	0,1	0,01
$e_{\alpha n,\infty}$	0,57	0,71	0,86	0,98

Proposition 4.3. Pour $p = \alpha n$ $\alpha \in]0,(2-\sqrt{2})/2[$, le temps mis pour exécuter l'algorithme est

$$T_p = \frac{(12+7\sqrt{2})\alpha^3 + \alpha^2 + 3\alpha + 2}{6\alpha} n^2 + O(n)$$

avec une efficacité asymptotique égale à

$$e_{\alpha n,\infty} = \frac{2}{(12+7\sqrt{2})\alpha^3 + \alpha^2 + 3\alpha + 2}$$

3.2.3. Autre version

En permutant les boucles i et k dans l'algorithme de l'inversion d'une matrice triangulaire inférieure à diagonale unité présenté au début de cette section, on obtient la décomposition en tâches suivante:

pour k := 2 **à** n-1
pour i := k+1 **à** n
début
    exécuter $T_{k,i}$:
    < **pour** j := 1 **à** k-1
        $a_{ij}^{(k)} := a_{ij}^{(k-1)}$ (+) $a_{ik}^{(k)}$ (x) $a_{kj}^{(k-1)}$; >
fin ;

La longueur de la tâche T_{ki} est 2(k-1) unités de temps, Le nombre total d'opérations arithmétiques est de $n^3/3+O(n)$. Les contraintes de précédence sont:

$$T_{k,k+1} << T_{k+1,i} \qquad k+1 \le i \le n \quad 1 \le k \le n-1$$
$$T_{ki} << T_{k+1,i} \qquad k+1 \le i \le n \quad 1 \le k \le n-1$$

L'allure du graphe d'ordonnancement est donné par la figure 7. C'est un graphe triangulaire. Nous renvoyons au chapitre II de cette thèse pour la description détaillée d'un algorithme s'exécutant avec $p=\alpha n$ processeurs, où α est nombre arbitraire dans]0,1].

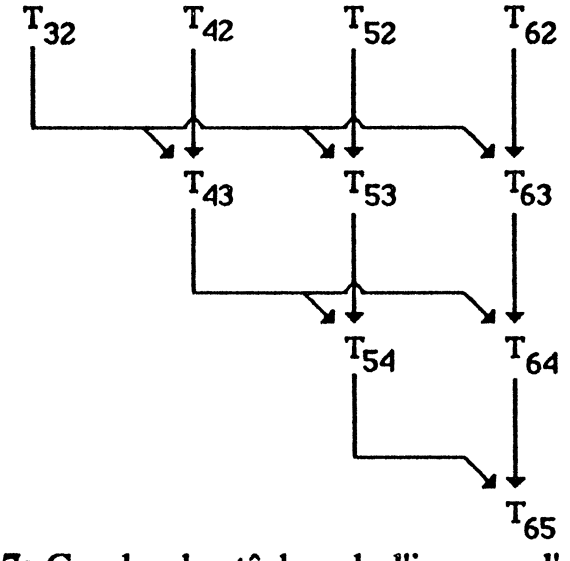


Figure 7: Graphe des tâches de l'inverse d'une matrice triangulaire de taille 6x6 à diagonale l'unité.

Les résultats sont résumés dans la

Proposition 4.4. i) $T_{opt}=n^2 + O(n)$ et $e_{n-1,\infty}=1/3$

ii) Pour $p=\alpha n$ avec $\alpha \in]0,1[$, il existe un algorithme d'efficacité asmyptotique $e_{p,\infty}=1/(1+2\alpha)$.

De même pour l'algorithme de l'inversion d'une matrice triangulaire supérieure, la décomposition en tâches indépendantes est la suivante:

pour k:=1 **à** n
début
  exécuter $R_{k,k}$:
  <pour i:=1 à k-1
    $a_{i,k}^{(k)} := a_{i,k}^{(k-1)}$ (x) $a_{kk}^{(k)}$>
  pour j := k+1 **à** n
    exécuter $R_{k,j}$:
    < $a_{k,j}^{(k)} := a_{k,k}^{(k)}$ (x) $a_{k,j}^{(k-1)}$;
    pour i:=1 **à** k-1
      $a_{ij}^{(k)} := a_{ij}^{(k-1)}$ (+) $a_{ik}^{(k)}$ (x) $a_{kj}^{(k-1)}$ >
fin ;

Les tâches R_{kk} n'existent pas si la diagonale de la matrice triangulaire à inverser est l'unité. Les longueurs des tâches R_{kk} et R_{kj} $(k+1\leq j\leq n)$ sont respectivement k-1 et 2k-1. Les contraintes d'ordonnnancement sont les suivantes:

R_{kk}	<<	R_{kj}	$2\leq k\leq n$	$k+1\leq j\leq n$
R_{kj}	<<	$R_{k+1,j}$	$1\leq k\leq n-1$	$k+2\leq j\leq n$

Le graphe d'ordonnancement est un graphe 2-pas. Le plus long chemin est donné par les tâches R_{12}, R_{22}, R_{23}, R_{33}, ..., $R_{n-1,n}$, R_{nn}. Ce qui entraîne que le temps T_{opt} pour exécuter ce graphe est égal à $3n^2/2+O(n)$. Une étude complète est faite sur ce type de graphe dans les chapitres précédents. En combinant les deux graphes : c'est à dire on intercale chaque niveau du graphe triangulaire entre deux niveaux du graphe 2-pas. Plus précisement, le niveau N_k du graphe de l'inverse d'une matrice triangulaire inférieure à diagonale l'unité s'exécute en parallèle avec la tâche R_{kk}. L'ordonnancement est donné par la figure suivante :

R_{12}	R_{13}	R_{14}	R_{15}
R_{22}	T_{32}	T_{42}	T_{52}
	R_{23}	R_{24}	R_{25}
	R_{33}	T_{43}	T_{53}
		R_{34}	R_{35}
		R_{44}	T_{54}
			R_{55}

Figure 8: Ordonnancement des tâches de la phase 2 de l'algorithme de Rote pour une matrice de taille 5x5.

Pour avoir des longueurs homogènes sur un même niveau, on suppose que la longueur d'une tâche diagonale R_{kk} est de l'ordre de 2(k-1); cela n'affecte pas l'évaluation asymptotique. Le temps T_{opt} devient alors $2n^2+O(n)$ contre $4n^2+O(n)$ pour la première version. L'algorithme Glouton est l'algorithme efficace pour exécuter ce type de graphe. Son efficacité est égal à $1/(1+3\alpha^2-\alpha^3)$ pour $\alpha\in]0,1]$.

Proposition 4.5. i) Il existe un algorithme exécutant avec n-1 processeurs la phase 2 de l'algorithme de Rote en parallèle en temps minimal

$$T_{opt}=2n^2+O(n)$$

avec une efficacité asymptotique

$$e_{n-1,\infty}=1/3$$

ii) Il existe un algorithme exécutant la phase 2 de l'algorithme de Rote en parallèle avec $p=\alpha n$ processeurs où $\alpha\in]0,1[$ avec une efficacité asymptotique

$$e_{p,\infty}= 1/(1+3\alpha^2-\alpha^3).$$

3.3. PARALLELISATION DE LA PHASE 3

L'algorithme de Rote [51] pour cette phase est:

pour i := 1 **à** n
  pour j := 1 **à** n
    exécuter $T_{i,j}$
    < si i>j alors $c_{i,j}^{(i)} := c_{i,i}^{(i)} * c_{i,j}^{(i-1)}$;
    pour k := max(i,j) +1 **à** n **faire**
      $c_{i,j}^{(k)} := c_{i,j}^{(k-1)} + c_{i,k}^{(k)} * c_{k,j}^{(k-1)}$>.

Le temps d'exécution d'une tâche $T_{i,j}$ est 2(n-i)+1 unités de temps si i>j, il est 2(n-j) unités de temps si i≤j. Séquentiellement, le temps mis pour exécuter cet algorithme est $2n^3/3 + O(n^2)$.

Les contraintes d'ordonnancement sont :

(A) $T_{i,j} << T_{i+1,j}$ pour $1 \le i \le n$ et $1 \le j \le i$

(B) $T_{i,j} << T_{i,j+1}$ pour $1 \le i \le n$ et $1 \le j \le n-1$

(C) $T_{i,j} << T_{j+1,j}$ pour $1 \le i \le j-1$ et $2 \le j \le n-1$

Le graphe d'ordonnancement est décrit à la figure 9.

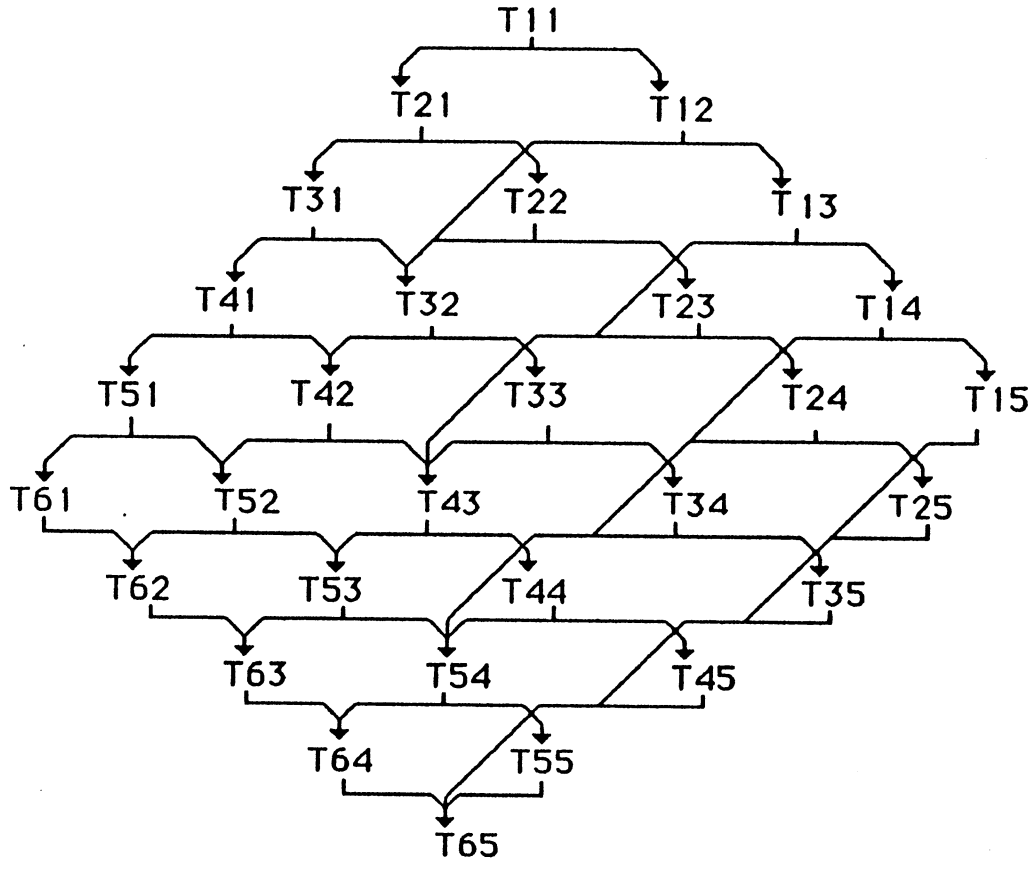


Figure 9: Graphe de multiplication en parallèle de deux matrices triangulaires de tailles 6x6 sans duplication.

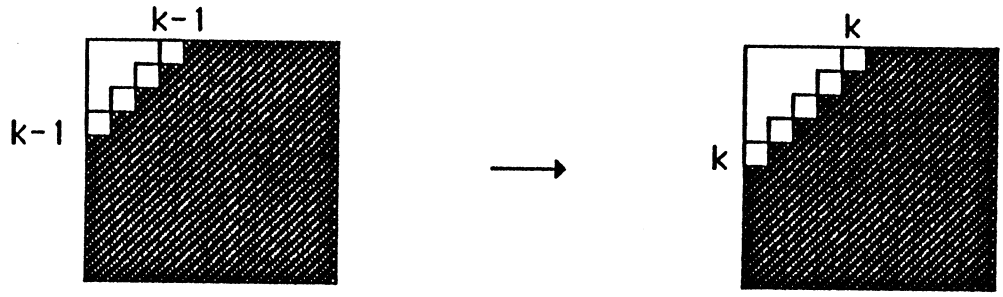


Figure 10 : Multiplication en parallèle de deux matrice triangulaires.

i) détermination de T_{opt}

Le plus long chemin est $\{T_{11},T_{21},T_{22},T_{32},\ldots,T_{n-1,n-1},T_{n,n-1}\}$, d'où T_{opt} est égal à

$$T_{opt} = \sum_{i=2}^{n} (2(n-i)+1) + \sum_{j=1}^{n-1} 2(n-j) = 2n^2 + O(n).$$

L'efficacité asymptotique avec n-1 processeurs est:

$$e_{n-1,\infty} = (2n^3/3)/2n^3 = 1/3.$$

ii) cas où $p=\alpha n$ où $\alpha \in]0,1[$

L'ALGORITHME PROPOSE

Le graphe d'ordonnancement se compose des diagonales de pente +1 et des diagonales de pente -1. Une diagonale j de pente +1 est constituée des tâches $T_{i,j}$ avec $j \leq i \leq n$, et une diagonale i de pente -1 est constituée des tâches $T_{i,j}$ avec $i<j<n$. On dispose de $p=2q$ processeurs notés P_1^+, P_2^+, ..., P_q^+, P_1^-, P_2^-, ..., P_q^-. L'exécution s'effectue par diagonale, les processeurs P_k^+ exécutent les diagonales de pente +1, les autres exécutent les diagonales de pente -1. Le processeur P_k^ε où $\varepsilon \in \{+,-\}$ exécute les diagonales k, k+q, k+2q,...,k+mq où $m \leq (n-q-r)/q$, si $\varepsilon=-$ on a $r=1$ et si $\varepsilon=+$ on a $r=0$. L'exécution de la tâche $T_{j,j}$ doit débuter après la fin de l'exécution de la tâche $T_{j,j-1}$ ce qui entraîne que l'exécution de la tâche $T_{i,j+1}$ ne peut débuter qu'après la fin de l'exécution de la tâche $T_{i,j}$. Les contraintes (B) sont vérifiées pour la raison précédente et du fait que l'exécution s'effectue par diagonales. Cette dernière raison assure aussi que les contraintes (A) sont vérifiées. Quand un processeur P_k^+ a terminé l'exécution de la tâche diagonale $T_{i,i}$, il passe à la tâche suivante de la diagonale i de pente +1. Au même instant, le processeur P_k^- commence l'exécution de la diagonale i de pente -1. Ce qui assure que les contraintes (C) sont vérifiées.

EXECUTION EN TEMPS T_{opt}

L'exécution de la diagonale i de pente +1 sera commencée par l'exécution de la tâche diagonale T_{ii} à l'instant δ_i^+ où $\delta_i^+ = \Sigma_{1 \leq m < i-1} [Ex(T_{m,m}) + Ex(T_{m+1,m})]$ tandis que l'exécution de la diagonale i de pente -1 débute juste après la fin de T_{ii}, c'est à dire à l'instant $\delta_i^- = \delta_i^+ + Ex(T_{ii})$.

Pour que l'exécution s'effectue en temps T_{opt}, les deux conditions suivantes doivent être vérifiées:

a) Le temps T_i^- mis pour exécuter la diagonale i de pente -1 doit être inférieur ou égal au temps $Ex(A_i)$ mis pour exécuter l'ensemble des tâches $A_i = \{T_{i+1,i}, T_{i+1,i+1}, T_{i+2,i+1}, \cdots, T_{i+q,i+q-1}, T_{i+q,i+q}\}$ pour tout i tel que $1 \leq i \leq n-p$.

b) le temps T_j^+ mis pour exécuter la diagonale j de pente +1 privée de la tâche diagonale T_{jj} doit être inférieur ou égal au temps mis $Ex(B_j)$ pour exécuter l'ensemble des tâches $B_j = \{T_{j+1,j}, T_{j+1,j+1}, T_{j+2,j+1}, \cdots, T_{j+q-1,j+q-1},$

$T_{j+q,j+q-1}$ } pour tout j tel que $1 \leq j \leq n-p$.

Pour que la condition a) soit vérifiée il faut que $i \in [\lambda, \xi]$ où

$$\lambda = \frac{2n - 4q + 1 - \sqrt{8q^2 - 4q + 1}}{2} \qquad \lambda \leqslant 1$$

et

$$\xi = \frac{2n - 4q + 1 + \sqrt{8q^2 - 4q + 1}}{2} \qquad \xi \geqslant n - p$$

$\xi \geq n-p$ est toujours vrai, pour que $\lambda \leq 1$ il faut $q \in [(2-\sqrt{2})n/2, n/2[$ pour n assez grand.
Pour que la condition b) soit vérifiée il faut que $j \in [\nu, \mu]$ où

$$\nu = n - 2q - \sqrt{2q^2 - 2q} \qquad \nu \leqslant 1$$

et

$$\mu = n - 2q + \sqrt{2q^2 - 2q} \qquad \mu \geqslant n - p$$

$\mu \geq n-p$ est toujours vrai, pour que $\nu \leq 1$ il faut $q \in [(4-\sqrt{14})n/2, n/2[$ pour n assez grand .
Les deux conditions a) et b) sont vérifiées simultanément si $q \in [(2-\sqrt{2})n/2, n/2[$, c'est à dire $p \in [(2-\sqrt{2})n, n[$, soit encore $\alpha \in [2-\sqrt{2}, 1[$, et l'efficacité asymptotique est

$$e_{\alpha n,\infty} = 1/(3\alpha).$$

Proposition 4.6. i) $T_{opt} = 2n^2 + O(n)$

ii) L'efficacité asymptotique avec n-1 processeurs est $e_{n-1,\infty} = 1/3$

iii) Il existe un algorithme s'exécutant en temps T_{opt} avec $p = 2q = \alpha n$ processeurs où $\alpha \in [2-\sqrt{2}, 1[$

iv) L'efficacité asymptotique avec αn processeurs où $\alpha \in [2-\sqrt{2}, 1[$ est $e_{\alpha n,\infty} = 1/(3\alpha)$

v) Pour $\alpha = 2-\sqrt{2}$, l'efficacité asymptotique est maximale; elle est égale à $(2+\sqrt{2})/6$.

EXECUTION AVEC αn PROCESSEURS où $\alpha \in]0,2-\sqrt{2}[$

L'exécution des 2q premières colonnes de pente $\varepsilon 1$ débute aux instants $\varphi_i^{\varepsilon}=\delta_i^{\varepsilon}$ avec $i \leq q$ et $\varepsilon \in \{-,+\}$. L'exécution de la diagonale i=j+q de pente $\varepsilon 1$ avec $q<i \leq \nu+q-1$ commence à l'instant $\varphi_i^{\varepsilon}=\varphi_j^{\varepsilon}+T_j^{\varepsilon}$ où $\varepsilon \in \{-,+\}$, T_j^{+} est le temps mis pour exécuter la diagonale j de pente +1 et $T_j^{-}=T_j^{+}+Ex(T_{ii})$. Pour i tel que $i \geq \nu+q$, l'exécution de la diagonale i de pente $\varepsilon 1$, où $\varepsilon \in \{-,+\}$, débute à l'instant $\varphi_i^{\varepsilon}=\varphi_r^{-\varepsilon}+Ex(T_{i,r})$ où r=i-1 si $\varepsilon=+$, r=i si $\varepsilon=-$. Le temps total pour l'exécution de l'algorithme est $T_p=\varphi_n^{+}+1$, pour $p=2q=\alpha n$ où $\alpha \in]0,2-\sqrt{2}[$

Il vient:

$$T_p = \frac{(-40+13\sqrt{2})\alpha^3 + 2(39-8\sqrt{2})\alpha^2 - 4\alpha + 16}{24\alpha} n^2 + O(n)$$

L'efficacité asymptotique est alors:

$$e_{\alpha n,\infty} = \frac{16}{(-40+13\sqrt{2})\alpha^3 + 2(39-8\sqrt{2})\alpha^2 - 4\alpha + 16}$$

Le tableau suivant illustre l'efficacité obtenue:

α	0,58	0,4	0,2	0,1
$e_{\alpha n,\infty}$	0,57	0,71	0,86	0,99

Proposition 4.8. Si on dispose de $p=2q=\alpha n$ où $\alpha \in]0,2-\sqrt{2}[$, l'algorithme s'exécute en temps

$$T_p = \frac{(-40+13\sqrt{2})\alpha^3 + 2(39-8\sqrt{2})\alpha^2 - 4\alpha + 16}{24\alpha} n^2 + O(n)$$

avec une efficacité asymptotique

$$e_{\alpha n,\infty} = \frac{16}{(-40+13\sqrt{2})\alpha^3 + 2(39-8\sqrt{2})\alpha^2 - 4\alpha + 16}$$

D'après cette étude, on conclut que l'inversion d'une matrice de taille nxn à l'aide de l'algorithme de Rote s'exécute en parallèle en temps $T_{opt}=6n^2+O(n)$ unités de temps, en supposant qu'on dispose n-1 processeurs.

3.4. UN ALGORITHME EQUIVALENT

Robert et Trystram [47] ont proposé l'algorithme suivant pour résoudre une instance arbitraire du problème APP:

{ ALGORITHME 2 }
pour k := 1 **à** n
début
$a_{kk}^{(k)} := (a_{kk}^{(k-1)})^*$
pour i := 1 **à** n, i ≠ k
$a_{ik}^{(k)} := a_{ik}^{(k-1)}$ (x) $a_{kk}^{(k)}$;
pour j := 1 **à** n, i ≠ k
début
pour i := 1 **à** n, i ≠ k
$a_{ij}^{(k)} := a_{ij}^{(k-1)}$ (+) $a_{ik}^{(k)}$ (x) $a_{kj}^{(k-1)}$;
$a_{kj}^{(k)} := a_{kk}^{(k)}$ (x) $a_{kj}^{(k-1)}$;
fin ;
fin ;

Le lemme suivant prouve l'équivalence des algorithmes 1 et 2:

Lemme 4.1. [47] Pour une donnée initiale $A^0 = (a_{ij}^{(0)})$ d'ordre n, l'algorithme 2 délivre en sortie la matrice $A^n = (a_{ij}^{(n)}) = D$ solution du problème APP.

La décomposition de cette algorithme en tâches est la suivante:

pour k := 1 **à** n
exécuter $T_{k,k}$
< $a_{kk}^{(k)} := (a_{kk}^{(k-1)})^*$
pour i := 1 **à** n, i ≠ k
$a_{ik}^{(k)} := a_{ik}^{(k-1)}$ (x) $a_{kk}^{(k)}$;>
pour j := 1 **à** n, i ≠ k
exécuter $T_{k,j}$
< **pour** i := 1 **à** n, i ≠ k
$a_{ij}^{(k)} := a_{ij}^{(k-1)}$ (+) $a_{ik}^{(k)}$ (x) $a_{kj}^{(k-1)}$;
$a_{kj}^{(k)} := a_{kk}^{(k)}$ (x) $a_{kj}^{(k-1)}$;>.

La longueur d'une tâche T_{ij} (i≠j) est 3(n-1) unités de temps, elle est égale à n pour toute tâche diagonale T_{ii}.

Le graphe des tâches est représenté à la figure 11.

i) calcul de T_{opt}

Le plus long chemin est $\{T_{11},T_{12},T_{22},T_{23},...,T_{n,n},T_{n,1}\}$ d'où

$$T_{opt} = 3n^2 + O(n).$$

Remarquons que le temps T_{opt} de cette version est la moitié que celle de la précédente.
L'efficacité asymptotique avec n-1 processeurs est :

$$e_{n-1,\infty} = 2n^3/3n^3 = 2/3.$$

ii) cas où $p=\lceil \alpha n \rceil$ où $\alpha \in]0,2/3]$

Le graphe donné par la figure 11, est étudié par [11] pour la résolution des systèmes linéaires par la méthode de Jordan par lignes.

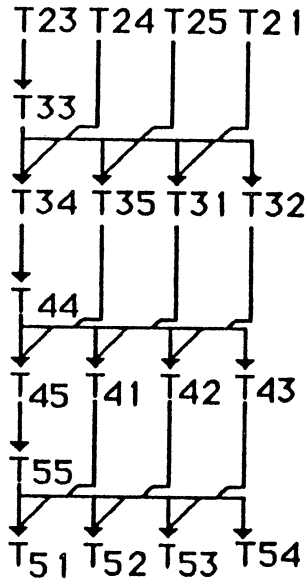


Figure 11: Graphe d'inversion d'une matrice de taille 5x5

Posons p=2q+1, tout d'abord, on affecte à un processeur les tâches du plus long chemin $\{T_{11},T_{12},T_{22},T_{23},\ldots,T_{n,n},T_{n,1}\}$, puis on répartit les tâches restantes en q blocks constitués chacun de r=(n-2)/q colonnes consécutives et on affecte leur exécution à un couple de processeurs, par exemple, les processeurs 2 et 3 exécutent les tâches des colonnes 1 à r. Plus généralement, les processeurs 2i et 2i+1 auront en charge l'exécution des tâches des colonnes (i-1)r+1, (i-1)r+2, ..., ir. Pour p=2n/3, ce qui correspond à chaque bloc est constitué de 3 colonnes consécutives et $T_{2n/3}$ est asymptotiquement égal à T_{opt} avec une efficacité 1; pour $p=\alpha n$, le temps nécessaire pour l'exécution de l'algorithme est $T_p=s*T_{opt}$ où s est le nombre de triplet de colonnes situées dans un bloc de r colonnes, s=(1/3)(2n/p)=2n/3p donc

$$T_p = 3n^2.(2n/3p)$$
$$T_p = 2n^2/\alpha \qquad \text{pour } p=\lceil \alpha n \rceil$$

Proposition 4.9. i) $T_{opt}=3n^2+O(n)$

ii) $e_{n-1,\infty}=2/3$

iii) Pour $\alpha \in]0,2/3]$, il existe un algorithme asymptotiquement optimal d'efficacité 1, s'exécutant en temps $2n^2/\alpha$ avec αn processeurs

iv) La valeur minimale de α permettant de réaliser un algorithme en T_{opt} est $\alpha = 2/3$

v) Pour $p \geq 2n/3$, $p=\alpha n$, l'efficacité asymptotique est $e_{p,\infty}=2/(3\alpha)$.

4. PARALLELISATION AVEC DUPLICATION TEMPORAIRE

Les phases 2 et 3 de l'algorithme de Rote peuvent être parallélisées avec duplication temporaire de certaines données. Dans ce cas, des contraintes d'ordonnancement se suppriment, d'autres apparaîssent. Nous avons plus de liberté dans la manipulation de certaines données, d'où une amélioration des algorithmes parallèles: diminution du temps optimal T_{opt} , augmentation de l'efficacité.

La duplication temporaire s'effectue à l'aide d'un vecteur V de longueur fixe déclaré au début du programme. Pour modifier un élément a_{ij} de la matrice A, les accumulations se font dans une composante de V au lieu de se faire sur a_{ij}. Ceci a pour effet de laisser la liberté aux autres processeurs d'accéder à l'ancienne valeur de a_{ij}.

Dans ce contexte, on répète l'étude des phases 2 et 3 de l'algorithme de Rote qui correspondent respectivement dans le cas de l'inversion d'une matrice à

- l'inversion des deux matrices triangulaires obtenues dans la phase 1

- multiplication des deux matrices inverses.

4.1. PARALLELISATION DE LA PHASE 2

Si l'on dispose d'un vecteur V de longueur n, l'algorithme de l'inversion d'une matrice triangulaire supérieure sera:

```
pour  i :=1 à n-1
début
    exécuter  R_{i,i+1}:
    < V_i := a_ii^(i) (x) a_{i,i+1}^(i-1) ;
    V_i := V_i (x) a_{i+1,i+1}^(i+1); >
    pour j := i+2 à n+1
```

```
début
    exécuter  R_{i,j}:
    < a_{ij-1}^{(j-1)} := V_i ;
    V_i := a_{ii}^{(i)} (x) a_{ij}^{(i-1)} ;
    pour k := i+1 à j-1
        V_i := V_i (+) a_{ik}^{(k)} (x) a_{kj}^{(k-1)}
    V_i := V_i (x) a_{jj}^{(j)} ; >
fin ;
fin ;
```

Le temps d'exécution d'une tâche R_{ij} est égal à 2(j-i). Il est le même que dans le cas sans duplication, puisque la duplication d'une colonne a un coût nul. Pour cette dernière raison aussi, les tâches $R_{i,n+1}$ ($1 \le i \le n-1$) possèdent un coût nul.

Les contraintes d'ordonnancement sont :

(A) $R_{ij} << R_{i,j+1}$ $\quad 1 \le i \le n-1 \quad i+1 \le j \le n+1$

(B) $R_{ij} << R_{k,j+1}$ $\quad 2 \le j \le n \quad 1 \le i \le k-1 \quad 1 \le k \le j-1$

La première contrainte correspond à la contrainte (A) du paragraphe 3.4.1., cela est dû à l'utilisation des valeurs calculées dans R_{ij} pour exécuter $R_{i,j+1}$. Pour k tel que $1 \le i \le k-1$, la contrainte (B) ci-dessus s'explique par le fait d'utiliser le contenu de a_{ij}. Si après avoir terminé l'exécution de R_{kj}, un processeur commence l'exécution de la tâche $R_{k,j+1}$ du niveau suivant N_{j+1}, a_{kj} prend la valeur de V_k et l'exécution de $a_{k,j+1}$ se lance en s'accumulant sur V_k. Par la suite, l'ancienne valeur de a_{kj} n'est plus stockée. L'exécution de toutes les tâches R_{ij} ($i<k$) doit être terminée, car elle utilisent l'ancien contenu de a_{kj}.
L'allure du graphe d'ordonnancement est présentée à la figure 12.
Le plus long chemin du graphe est $\{R_{12}, R_{13}, ..., R_{1n}\}$ d'où la valeur

$$T_{opt} = \sum_{i=2}^{n} 2(i-1) = n^2 + O(n)$$

Il est évident de construire un algorithme s'exécutant en T_{opt} avec n-1processeurs. L'efficacité asymptotique de cet algorithme est égale à $(n^3/3)/n^3 = 1/3$.

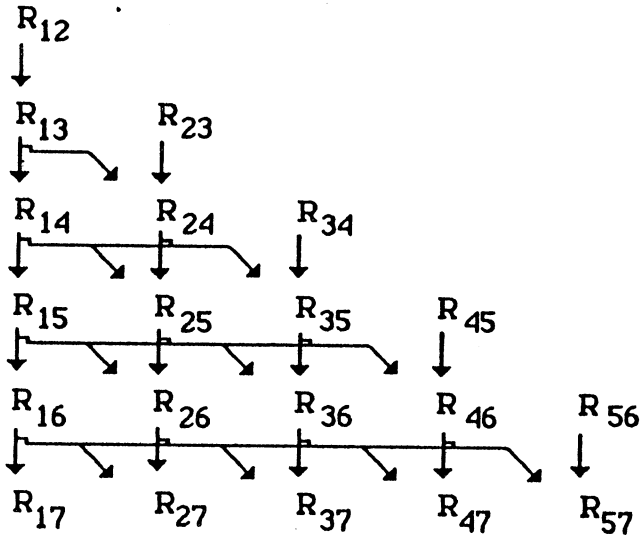


Figure 12: Graphe d'inversion d'une matrice triangulaire supérieure de taille 6x6 avec duplication temporaire.

L'exécution parallèle d'un tel graphe n'a pas encore été étudiée. La différence de ce type de graphe avec ceux étudiés réside dans le fait que les tâches d'un même niveau n'ont plus la même longueur. Sur un niveau quelconque N_k, la longueur des tâches est décroissante de gauche à droite. Le taux de décroissance est constant, il est égal à deux unités de temps.

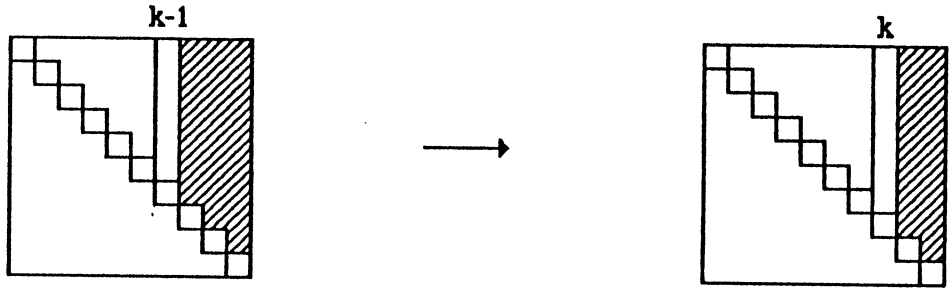


Figure 13: Inversion en parallèle d'une matrice triangulaire supérieure avec duplication temporaire.

Etant donné p processeurs notés P_1, P_2, ..., P_p, on définit l'algorithme suivant: cet algorithme possède deux phases dont la première est l'exécution des niveaux compris entre N_1 et N_{2p-1}. L'algorithme exécute les tâches du graphe d'ordonnancement de gauche à droite, en commençant à tout instant le maximum de tâches exécutables. La différence entre les deux phases réside dans le passage à un niveau suivant avant que l'exécution des tâches du niveau précédent ne soit terminée.

Première phase

Dans la première phase, le processeur P_i exécute les tâches des colonnes i, 2p-i+1. L'algorithme procède niveau par niveau, mais le commencement d'un niveau quelconque n'a lieu qu'après la fin du niveau précédent, c'est à dire juste après la fin de l'exécution de la tâche de la colonne 1 par le processeur P_1. On a $Ex(R_{1j}) \geq Ex(R_{ij}) + Ex(R_{2p-i+1,j})$ pour tout j tel que $1 \leq j \leq 2p-1$ ce qui assure que les contraintes sont vérifiées. Le temps d'exécution dans cette phase est optimal, il est égal à

$$\tau_1 = \sum_{j=2}^{2p-1} 2(j-1) = 4p^2 + O(p)$$

Les processeurs arrivent au même temps pour la:

Seconde phase

On considère maintenant la seconde phase de l'algorithme : du niveau N_{2p} jusqu'au dernier niveau N_n. Dans cette deuxième phase l'algorithme est d'efficacité 1. Les processeurs opèrent suivant l'algorithme Glouton niveau par niveau et de gauche à droite. Pour plus de précision, l'algorithme exécute les tâches de cette phase dans l'ordre partiel suivant :

$$R_{1,2p} \leq\leq R_{2,2p} \leq\leq \ldots \leq\leq R_{2p-1,2p} \leq\leq \ldots \leq\leq \ldots \leq\leq$$
$$R_{1,k} \leq\leq \ldots \leq\leq R_{k-1,k} \leq\leq \ldots \leq\leq R_{1,n} \leq\leq \ldots \leq\leq R_{n-1,n}$$

où $T \leq\leq T'$ signifie que l'exécution de la tâche T' ne peut débuter avant celle de la tâche T mais la simultanéité est possible.
Les tâches $R_{i,n+1}$ sont exécutées après que l'exécution de toutes les autres tâches est terminée.

Lemme 4.2. Le début de l'exécution de deux tâches consécutives $R_{2mp+j,k+1}$ et $R_{2mp+j+1,k+1}$ (respectivement $R_{1+2mp-j,k+1}$ et $R_{2mp-j,k+1}$) où $1 \leq j \leq p-1$, se fait avec au plus un déphasage de deux unités de temps. Les tâches $R_{2mp+j,k+1}$ et $R_{1+2mp-j,k+1}$ si elles existent s'exécutent par $P_{\sigma(j)}$ où σ une permutation de l'ensemble $\{1, 2, \ldots, p\}$.

Preuve

Par récurrence sur les niveaux N_k.
En arrivant au niveau N_{2p} les processeurs se synchronisent naturellement, ils finissent aussi l'exécution de ce niveau en même temps: la durée d'activité de tous les processeurs est constante puisque la longueur des tâches situées sur un même niveau décroit de gauche à droite, de manière constante. L'exécution de deux

tâches consécutives des p dernières tâches du niveau N_{2p} débute avec au plus un déphasage de deux unités de temps. Le lemme est vrai pour le niveau N_{2p}. On suppose qu'il est vrai pour le niveau N_{k-1} et on le démontre pour le niveau N_k. D'après l'hypothèse de récurrence, le début de l'exécution des deux tâches consécutives $R_{j,k+1}$et $R_{j+1,k+1}(1\leq j\leq p-1)$ se fait au plus avec un déphasage de deux unités de temps. Puisque l'écart des longueurs de ces deux tâches est deux unités de temps, on est dans l'un des deux cas suivants à la fin de l'exécution de ces deux tâches :

1) Les deux processeurs se synchronisent si le début de l'exécution des tâches $R_{j,k+1}$et $R_{j+1,k+1}(1\leq j\leq p-1)$ est décalé de deux unités de temps.

2) L'exécution de la tâche $R_{j+1,k+1}$ se termine avant deux unités de temps que celle de la tâche $R_{j,k+1}$si le début de l'exécution de ces deux dernières tâches est simultané.

Ce qui entraîne que le processeur $P_{\sigma(j+1)}$ termine son travail dans le premier bloc (formé des p premières tâches) au plus tard en même temps que $P_{\sigma(j)}$. Deux tâches consécutives du bloc suivant formé des tâches $R_{1+2p-j,k+1}$ $(1\leq j\leq p)$ débutent avec au plus un décalage de deux unités de temps. Le processeur $P_{\sigma(j)}$ exécute la tâche située juste après celle associée au processeur $P_{\sigma(j+1)}$. Autrement dit, les processeurs $P_{\sigma(p)}$, $P_{\sigma(p-1)}$, ..., $P_{\sigma(1)}$ exécutent respectivement les tâches $R_{p+1,k+1}$, $R_{p+2,k+1}$, ...,$R_{2p,k+1}$. Puisque l'écart entre les longueurs de deux tâches consécutives est constante à de deux unités de temps, les processeurs $P_{\sigma(j)}$ et $P_{\sigma(j+1)}$ passent du deuxième bloc formé par les tâches $R_{1+2p-j,k+1}$ $(1\leq j\leq p)$ au troisième bloc formé par les tâches $R_{2p+j,k+1}$ $(1\leq j\leq p)$ avec au plus un déphasage de deux unités de temps. Ils débutent respectivement l'exécution des tâches $R_{2p+j,k+1}$ et $R_{2p+j+1,k+1}$. D'une façon générale, supposons que les processeurs $P_{\sigma(j)}$ et $P_{\sigma(j+1)}$ terminent l'exécution des tâches associées, d'un bloc de tâches de la forme $R_{2mp+i,k+1}$ou de tâches de la forme $R_{1+2mp-i,k+1}$, avec au plus un déphasage de deux unités de temps. A la fin de l'exécution de ces tâches, les processeurs $P_{\sigma(j)}$ et $P_{\sigma(j+1)}$ passent à l'exécution des deux premières tâches non encore commencées du bloc suivant. A la fin de l'exécution de ces deux dernières tâches ou bien les processeurs se synchronisent s'ils étaient décalés, ou bien ils se décalent s'ils étaient synchronisés.

Plus formellement :

Les tâches du bloc à exécuter sont de la forme $R_{2mp+i,k+1}$ (ou $R_{1+2mp-i,k+1}$)

1- Si $P_{\sigma(j)}$ et $P_{\sigma(j+1)}$ terminent au même instant l'exécution des deux dernières tâches associées dans le niveau précédent. Alors $P_{\sigma(j)}$ et $P_{\sigma(j+1)}$ abordent simultanément et respectivement l'exécution des tâches $R_{2mp+j,k+1}$ et $R_{2mp+j+1,k+1}$(ou $R_{2mp-j+1,k+1}$ et $R_{2mp-j,k+1}$). Le processeur $P_{\sigma(j)}$ se retarde (ou s'avance) de deux unités de temps par rapport au processeur $P_{\sigma(j+1)}$. Ce qui autorise $P_{\sigma(j+1)}$ (ou $P_{\sigma(j)}$) à aborder l'exécution la première tâche non encore commencée du bloc suivant qui est $R_{2(m+1)p-j,k+1}$ (ou $R_{2(m+1)p+j,k+1}$). La tâche suivante, $R_{1+2(m+1)p-j,k+1}$ (ou $R_{2(m+1)p+j,k+1}$), est associée au processeur $P_{\sigma(j)}$ (ou $P_{\sigma(j+1)}$).

2- Si $P_{\sigma(j)}$ et $P_{\sigma(j+1)}$ terminent l'exécution des deux dernières tâches associées dans le niveau précédent avec un décalage de deux unités de temps. Alors $P_{\sigma(j)}$ et $P_{\sigma(j+1)}$ abordent avec un déphasage de deux unités de temps respectivement l'exécution des tâches $R_{2mp+j,k+1}$ et $R_{2mp+j+1,k+1}$ (ou $R_{1+2mp-j,k+1}$ et $R_{2mp-j,k+1}$). A la fin de cette exécution les processeurs $P_{\sigma(j)}$ et $P_{\sigma(j+1)}$ se synchronisent . Ce qui autorise le début simultané de l'exécution des deux premières tâches non encore commencées du bloc suivant qui sont $R_{1+2(m+1)p-j,k+1}$et $R_{2(m+1)p-j,k+1}$(ou $R_{2(m+1)p+j,k+1}$et $R_{1+2(m+1)p+j,k+1}$).

Les contraintes d'ordonnancement sont vérifiées puisque chaque processeur exécute au plus une seule tâche d'un bloc de tâches de la forme $R_{2mp+j,k}$ ou $R_{1+2mp-j,k}$($1 \leq j \leq p$). L'exécution des p premières tâches du niveau N_k se termine avant la fin de l'exécution du bloc suivant formé de p tâches du même niveau. Ce qui entraîne que le début de l'exécution des p premières tâches du niveau suivant est possible par les p processeurs. Les tâches du niveau N_{k+1} situées après ce premier bloc sont aussi libérées des contraintes d'ordonnancement car les p processeurs se situent sur le niveau N_{k+1} avant la fin de l'exécution du premier bloc.

Proposition 4.10. L'algorithme respecte les contraintes d'ordonnancement.

Pour évaluer le temps d'exécution de cette phase, nous notons qu'à partir du niveau N_{2p} tous les p processeurs seront actifs sans interruption : en effet il y a plus de 2p tâches par niveau ce qui assure l'exécution d'au moins de deux tâches par un seul processeur (en d'autres termes l'algorithme dans cette phase possède une efficacité asymptotique 1). Nous obtenons le temps d'exécution de cette phase :

$$\tau_2 = \frac{\frac{n^3}{3} - \sum_{i=1}^{2p-1} \sum_{j=i+1}^{2p} Ex(R_{ij})}{p} = \frac{n^3 - 8p^3}{3p} + O(n)$$

Le temps total est

$$T_p = \tau_1 + \tau_2 = \frac{n^3 + 4p^3}{3p} + O(n)$$

Pour $p = \lceil \alpha n \rceil \alpha \in]0,1]$ $\quad T_p = \frac{4\alpha^3 + 1}{3\alpha} n^2 + O(n)$ $\quad$ et $\quad e_{\alpha n,\infty} = \frac{1}{4\alpha^3 + 1}$

Si $\alpha \geq 1/2$, l'exécution s'effectue en temps T_{opt}.

Si $\alpha = 1/2$, l'efficacité asymptotique $e_{\lceil n/2 \rceil,\infty} = 2/3$.

Nous allons détailler l'exécution de l'algorithme présenté sur un exemple. Choisissant n=9 et p=3, les processeurs opèrent selon le schéma suivant (ij dans la colonne P_s signifie que le processeur P_s effectue la tâche R_{ij} après l'exécution de celle située au dessus). On ne tient pas compte de l'exécution des tâches $R_{i,n+1}$ $(1 \leq i \leq n-1)$:

	P_1	P_2	P_3
niveau N_1	12	-	-
niveau N_2	13	23	-
niveau N_3	14	24	34
niveau N_4	15	25	35
			45
niveau N_5	16	26	36
		56	46
niveau N_6	17	27	37
	67	57	47
niveaux N_7 et N_8	18	28	38
	68	58	48
	78	19	29
	39	69	59
	49	79	89

Proposition 4.11. i) Pour $p=\lceil \alpha n \rceil (\alpha \in [1/2,1])$, il existe un algorithme s'exécutant en temps

$$T_{opt}=n^2 + O(n)$$

avec une efficacité asymptotique égale à

$$e_{\alpha n,\infty}=1/3\alpha$$

ii) Pour $p=\lceil \alpha n \rceil$ $(\alpha \in [0,1/2])$, il existe un algorithme s'exécutant en temps

$$T_p = \frac{4\alpha^3 + 1}{3\alpha} n^2 + O(n)$$

avec une efficacité asymptotique égale à

$$e_{p,\infty} = \frac{1}{4\alpha^3 + 1}$$

iii) Si $\alpha=1/2$, l'exécution s'effectue en temps T_{opt} avec une efficacité asymptotique égale $e_{\lceil n/2 \rceil,\infty}=2/3$.

4.2. PARALLELISATION DE LA PHASE 3

L'objet de la phase 3 est la multiplication de deux matrices triangulaires. Les contraintes d'ordonnancement sont dûes à l'utilisation des anciennes valeurs de a_{ij} ce qui nous oblige à calculer les cœfficients de la matrice produit dans un ordre bien déterminé. On répète la modification faite pour la deuxième phase à cette troisième phase, en utilisant un vecteur V de longueur n-1.

L'algorithme devient :

```
pour  j := 1 à n-1
début
      exécuter T1,j
      < Vj:=c1,j(j);
      pour k :=j +1 à n faire
            Vj := Vj + ci,k(k) * ck,j(k-1) ; >.
      pour  i := 2 à n
      début
            exécuter Ti,j
            < ci-1,j(n) :=Vj
            si i>j alors Vj:= ci,i(i) * ci,j(i-1) ;
            sinon alors Vj:=ci,j(j);
            pour k := max(i,j) +1 à n faire
                  Vj := Vj + ci,k(k) * ck,j(k-1) ; >.
      fin ;
```

exécuter $T_{n+1,j}$
$< c_{n,j}^{(n)} := V_j ;>$
fin ;

Puisque la duplication d'une colonne a un coût nul, la longueur d'une tâche $T_{i,j}$ est égale à 2(n-i)+1 unités de temps si i>j, il est 2(n-j) unités de temps si i≤j. On suppose pour i>j que la longueur d'une tâche est 2(n-i) au lieu de 2(n-i)+1. Cela n'affecte pas l'évaluation asymptotique des performances. Séquentiellement, le temps mis pour exécuter cet algorithme est $2n^3/3 + O(n^2)$. Pour cette dernière raison, la longueur des tâches $T_{n+1,j}$ (1≤j≤n-1) est nulle.
Les contraintes d'ordonnancement sont:

(A) $T_{i,j} << T_{i+1,j}$ pour $1 \le i \le n$ $1 \le j \le n-1$
(B) $T_{i,j} << T_{i+1,k}$ pour $1 \le j \le n-1$ $j+1 \le k \le n$ $1 \le i \le n$

On remarque que sur un niveau N_k, les k premières tâches possèdent le même temps d'exécution tandis que le temps d'exécution des tâches restantes du même niveau décroit d'une tâche à la suivante de deux unités de temps. L'allure du graphe d'ordonnancement est présentée à la figure 14.

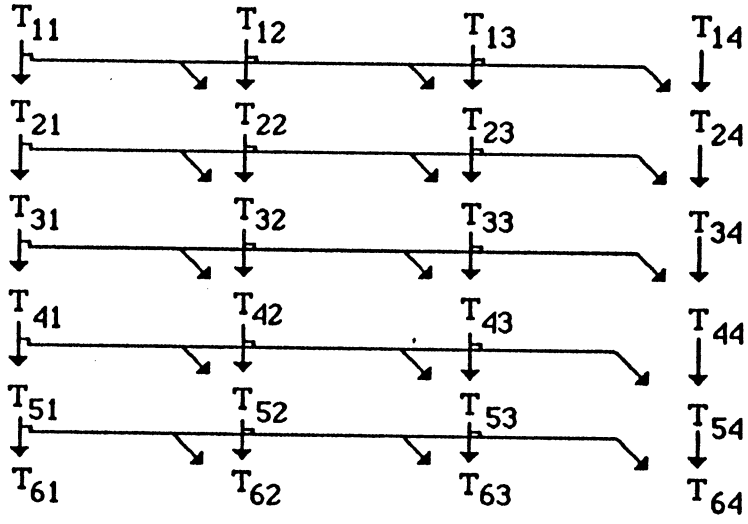


Figure 14: Graphe de multiplication de deux matrices triangulaires de tailles 5x5 avec duplication temporaire.

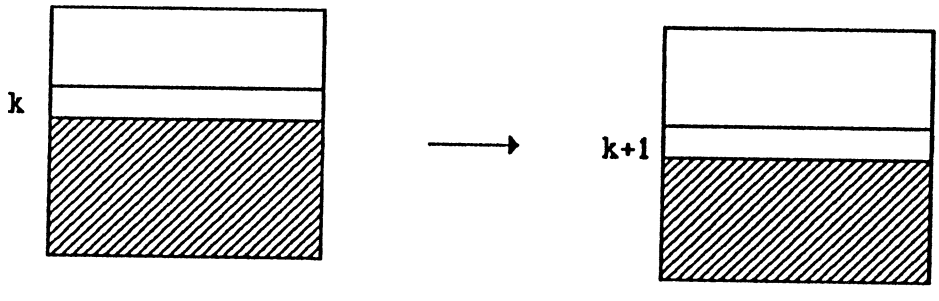


Figure 15: Multiplication de deux matrices triangulaires en parallèle avec duplication temporaire.

Le plus long chemin est constitué par $\{T_{11}, T_{21}, ..., T_{n1}\}$, sa longueur est égale à $T_{opt}=n^2+O(n)$. Il est facile de construire un algorithme s'exécutant en temps T_{opt} avec n-1 processeurs d'où l'efficacité asymptotique est évalueé à $(2n^3/3)/n^3= 2/3$.

Si les tâches possèdent une longueur constante par niveau, il est évident de construire un algorithme asymptotiquement optimal avec $p=\alpha n$ processeurs. La difficulté réside dans le fait que les tâches d'un même niveau n'ont pas la même longueur. Sur le niveau N_k, les k premières tâches possèdent une longueur constante à 2(n-k). Cette longueur décroit d'une tâche à la suivante (de gauche à droite) de deux unités de temps jusqu'à devenir 2 unités de temps pour la dernière tâche de chaque niveau.

On définit un algorithme parallèle avec p processeurs ($p=\alpha n$, $\alpha \in]0,1]$) notés $P_1, P_2,..., P_p$. L'algorithme associe à chaque processeur une ou deux colonnes du graphe pour $\alpha \in [1/2,1]$. Il fonctionne de gauche à droite en exécutant à tout instant le maximum de tâches exécutables pour $\alpha \in]0,1/2]$: dès qu'un processeur se libère, il passe à l'exécution de la première tâche non encore commencée, en respectant les contraintes d'ordonnancement.

Cas où $\alpha \in [1/2,1]$

L'algorithme procède niveau par niveau, mais le commencement d'un niveau quelconque n'aura lieu qu'après la fin de l'exécution de toutes les tâches du niveau précédent. Le processeur P_i exécute les tâches de la colonne i si i vérifie $1 \le i \le 2p-n+1$, il exécute les tâches des colonnes i et 2p-i+1 si $i \ge 2p-n+2$. Les contraintes d'ordonnancement sont vérifiées grâce à la synchronisation des processeurs au passage à chaque niveau.
Le temps mis par un processeur pour exécuter les tâches correspondantes dans un niveau N_j tel que $j \le 2p-n$ est toujours inférieur ou égal au temps mis par le processeur P_1 pour exécuter sa tâche de la première colonne. En effet, d'une part, $Ex(T_{j1}) \ge Ex(T_{ji})$ si $1 \le i \le 2p-n+1$, d'autre part, $Ex(T_{j1}) \ge Ex(T_{ji}) + Ex(T_{j,2p-i+1})$ si $i \ge 2p-n+2$. Si $j > 2p-n$, le processeur retardataire sur le niveau N_j est le processeur P_j exécutant les tâches T_{jj} et $T_{j,2p-j+1}$. On a d'une part, $Ex(T_{jj}) + Ex(T_{j,2p-j+1}) = Ex(T_{ji}) + Ex(T_{j,2p-i+1})$ pour tout i et j vérifiant $j<i<n$ et $j>2p-n$, d'autre part, $Ex(T_{jj})=Ex(T_{ji})$ pour i vérifiant $i \le j$.
Le temps mis pour exécuter l'algorithme est égal à

$$T_p = \sum_{i=1}^{n-1} Ex(T_{ii}) + \sum_{j=2p-n+1}^{n-1} Ex(T_{j,2p-j+1}) = 5n^2 - 4p^2 + O(n)$$

Pour $\alpha \in [\frac{1}{2}, 1]$ $\quad T_p = (-4\alpha^2 + 5)n^2 + O(n) \quad e_{p,\infty} = \dfrac{2}{3\alpha(-4\alpha^2 + 5)}$

Proposition 4.12. i) Pour $\alpha=1$, il existe un algorithme s'exécutant en temps T_{opt} avec une efficacité asymptotique égale à 2/3.

ii) Pour $\alpha\in[1/2,1[$, il existe un algorithme s'exécutant en temps

$$T_p = (-4\alpha^2 + 5)n^2 + O(n)$$

avec une efficacité asymptotique

$$e_{p,\infty} = \frac{2}{3\alpha(-4\alpha^2 + 5)}$$

Cas où $\alpha\in]0,1/2]$

L'algorithme balaye le graphe des tâches niveau par niveau et de gauche à droite. Pour plus de précision, l'algorithme exécute les tâches de cette phase dans l'ordre partiel suivant :

$$T_{11} \leq\leq \ldots \leq\leq T_{1,n-1} \leq\leq T_{21} \leq\leq \ldots \leq\leq T_{k,n-1} \leq\leq \ldots \leq\leq T_{n,n-1}$$

où $T\leq\leq T'$ signifie que l'exécution de la tâche T' ne peut débuter avant celle de la tâche T mais la simultanéité est possible.
L'exécution d'un niveau quelconque peut commencer même si l'exécution du niveau précédent n'est pas encore terminée. Les processeurs qui exécutent les premières tâches d'un niveau N_k tel que $n-p+2\leq k\leq n-1$ ne doivent pas aborder l'exécution d'autres tâches du même niveau, s'il existe encore au moins un processeur terminant l'exécution du niveau N_{k-1}. Ce temps libre dû au passage des processeurs d'un niveau à un autre n'affecte pas l'évaluation asymptotique. En effet, la longueur maximale d'une tâche du niveau N_{k-1} est 2(n-k-1), il ne diffère que de 2 unités de temps de la longueur d'une tâche des p premières tâches d'un niveau N_k. Ces dernières possèdent la même longueur, elle est égale à 2(n-k). Entre les niveaux N_{n-p+2} et N_{n-1} on a O(p) niveaux, donc le temps total d'inactivité des processeurs est de l'ordre de $O(p^2)$. Le temps d'exécution parallèle n'est donc pas affecté. On néglige l'exécution des tâches du niveau N_{n+1}. Soit n-1=ap+r où $0\leq r<p$ la division euclidienne du nombre de colonnes du graphe par celui des processeurs. On note B_1^k, B_2^k, ..., B_a^k, B_{a+1}^k les blocs du niveau N_k dont chacun des a premiers est formé de p tâches. Si $r\neq 0$, le dernier bloc B_{a+1}^k existe et est formé de r tâches.

Lemme 4.3. Le passage consécutif de deux processeurs d'un bloc au suivant possède au plus un déphasage de deux unités de temps.

Preuve

Par récurrence sur les niveaux N_i.

Pour i=1
Sur le niveau N_1, on commence les premières tâches du premier bloc au même instant. Le processeur P_i exécute la tâche T_{1i} ($1 \leq i \leq p$). Puisque la longueur des tâches du premier niveau décroit de deux unités de temps d'une tâche à la suivante alors le processeur P_p, exécutant la dernière tâche du premier bloc, finit le premier et aborde l'exécution de la tâche $T_{1,p+1}$: la première tâche du deuxième bloc. Après deux unités de temps, le processeur P_{p-1} aborde l'exécution de la tâche $T_{1,p+2}$. Deux unités de temps après le passage du processeur P_i au bloc $B_2{}^1$, le processeur P_{i-1} débute l'exécution de sa tâche associée dans le bloc $B_2{}^1$. Les processeurs se synchronisent naturellement après la fin de l'exécution du second bloc. Ils abordent l'exécution du bloc suivant en vérifiant notre hypothèse : le passage consécutif de deux processeurs possède au plus un déphasage de deux unités de temps. Ils se synchronisent aussi après avoir effectué les deux blocs suivants $B_3{}^1$ et $B_4{}^1$. L'exécution se poursuit par bloc de p tâches et à la fin de chaque bloc $B_i{}^1$ avec i pair, les processeurs se synchronisent. Ce processus se termine à un instant donné où on commence l'exécution des r dernières tâches. L'exécution du niveau N_2 débute au plus tard, après le début de l'exécution de la dernière tâche du niveau N_1. Les processeurs arrivent au niveau N_2 soit du bloc $B_a{}^1$ soit du bloc $B_{a+1}{}^1$. Le déphasage maximal entre deux processeurs passant successivement au niveau N_2 ne dépasse pas 2 unités de temps. Le lemme est donc vrai pour le niveau N_1. On suppose qu'il est vrai pour le niveau N_{k-1} et on le démontre pour le niveau N_k.
D'après l'hypothèse de récurrence, les processeurs arrivent sur le niveau N_k avec au plus un déphasage de deux unités de temps. Deux processeurs exécutant deux tâches consécutives ou bien ils se synchronisent ou bien ils se décalent de deux unités de temps:

1- Ils se synchronisent s'ils étaient déphasés avant le début de l'exécution des deux tâches consécutives du bloc $B_1{}^k$, si ces deux dernières n'ont pas la même longueur ou bien si les deux tâches possèdent la même longueur et si les processeurs arrivent au même instant au niveau N_k.

2- Ils se décalent de deux unités de temps s'ils étaient synchronisés et si les deux tâches correspondantes ont des longueurs différentes ou bien ils arrivent sur le niveau N_k avec un déphasage de deux unités de temps et si les deux tâches correspondantes ont la même longueur.
Les k premières tâches du niveau N_k ont la même longueur. L'exécution des $\lfloor k/p \rfloor$ premiers blocs du niveau N_k ne modifie pas le déphasage des processeurs. Les tâches du bloc $B_{\lfloor k/p \rfloor+1}{}^k$ possèdent l'une des propriétés suivantes:

i) la longueur d'une tâche décroit de deux unités de temps par rapport à celle de sa précédente
ii) les premières tâches possèdent la même longueur tandis que les autres ont

des longueurs décroissantes.

Si $B_{\lfloor k/p \rfloor+1}{}^{k}$ vérifie la propriété i)

Deux processeurs passant consécutivement au bloc $B_{\lfloor k/p \rfloor+2}{}^{k}$ ont au plus un déphasage de deux unités de temps. En effet, supposons que les processeurs $P_{\sigma(1)}$, $P_{\sigma(2)}$, $P_{\sigma(3)}$, ..., $P_{\sigma(m)}$ exécutent respectivement les tâches du bloc $B_{\lfloor k/p \rfloor+1}{}^{k}$: $T_{k,ip+1}$, $T_{k,ip+2}$, ..., $T_{k,m}$ avec $i= \lfloor k/p \rfloor$, $m=r$ si $\lfloor k/p \rfloor = a$ ou $m=(\lfloor k/p \rfloor +1)p-1$ si $1 \leq \lfloor k/p \rfloor < a$ et σ une permutation de {1, 2, ..., p}. Soient $P_{\sigma(u)}$ et $P_{\sigma(v)}$ deux processeurs passant consécutivement au bloc suivant $B_{\lfloor k/p \rfloor+2}{}^{k}$. $P_{\sigma(u)}$ passe le premier. Les processeurs $P_{\sigma(u+1)}$, $P_{\sigma(u+2)}$, ..., $P_{\sigma(p)}$ terminent l'exécution des tâches correspondantes du bloc $B_{\lfloor k/p \rfloor +1}{}^{k}$ au plus tard en même temps que $P_{\sigma(u)}$: d'une part la longueur des tâches décroit, d'autre part, le déphasage maximal entre le début de l'exécution de deux tâches consécutives ne dépasse pas deux unités de temps. Ce qui entraîne que $v<u$ et par la suite on choisit v de telle façon que $P_{\sigma(v+1)}$ quitte le bloc $B_{\lfloor k/p \rfloor+1}{}^{k}$ en même temps que $P_{\sigma(u)}$: c'est à dire $P_{\sigma(v+1)}$ passe au bloc $B_{\lfloor k/p \rfloor+2}{}^{k}$ deux unités de temps avant $P_{\sigma(v)}$. Ce qui entraîne que $P_{\sigma(v)}$ et $P_{\sigma(v+1)}$ débutent l'exécution de leur tâche correspondante du bloc $B_{\lfloor k/p \rfloor+1}{}^{k}$ au même instant. Puisque la longueur des tâches décroit de deux unités de temps, le passage au bloc suivant $B_{\lfloor k/p \rfloor+2}{}^{k}$ des deux processeurs $P_{\sigma(v)}$ et $P_{\sigma(v+1)}$ se fait avec un déphasage de deux unités de temps. Par la suite, le déphasage entre le passage des deux processeurs $P_{\sigma(v)}$ et $P_{\sigma(u)}$ est de deux unités de temps. La même démonstration est valable pour les autres blocs possédant la même propriété.

Si $B_{\lfloor k/p \rfloor+1}{}^{k}$ vérifie la propriété ii)

Les premières tâches de ce bloc ont une longueur constante. Le déphasage des processeurs exécutant ces dernières ne varie pas. D'après l'hypothèse de récurrence, il ne dépasse pas 2 unités de temps pour deux processeurs exécutant deux tâches consécutives. Le passage de deux processeurs, exécutant deux tâches de même longueurs, au bloc suivant se fait avec au plus un déphasage de deux unités de temps. Les autres processeurs exécutent les tâches restantes du bloc $B_{\lfloor k/p \rfloor+1}{}^{k}$. La longueur de ces dernières est décroissante : c'est le cas i). Pour la continuité de passage au bloc suivant entre ces derniers processeurs et ceux exécutant les tâches $B_{\lfloor k/p \rfloor+1}{}^{k}$ de longueur constante, c'est aussi le cas i) qui se présente entre deux processeurs : celui qui exécute la dernière tâche possédant la même longueur que ses précédentes et le processeur exécutant la première tâche située après cette dernière. Pour les blocs suivants, ils possèdent la propriété i), la

démonstration pour le cas i) est toujours valable. Par conséquent le lemme est vérifié pour tout niveau N_k $1 \leq k \leq n-1$.

Lemme 4.4. Sur tout niveau N_k tel que $1 \leq k \leq n-p+1$, tout processeur exécute au plus une tâche par bloc B_i^k.

Preuve

L'exécution des blocs suivant du niveau N_k: B_1^k, B_2^k, ..., B_{a-1}^k vérifie le lemme: la longueur d'une tâche de ces blocs est supérieure à 2(p-1) unités de temps et le déphasage maximal entre le premier processeur abordant le bloc B_i^k et le dernier processeur arrivant au même bloc ne dépasse pas 2(p-1) (lemme 4.3.). Pour les blocs B_a^k et B_{a+1}^k , on a trois situations possibles:

i) les tâches de B_a^k et une partie des tâches de B_{a+1}^k ont la même longueur

ii) les premières tâches de B_a^k ont la même longueur

iii) toutes les tâches de B_a^k et B_{a+1}^k ont des longueurs différentes.

La longueur des tâches qui ont la même longueur est nécessairement supérieure à 2(p-1) car $n \geq 2p$ et k vérifie $1 \leq k \leq n-p+1$. Donc le cas i) vérifie le lemme. Pour la même raison, la partie possédant une longueur constante du cas ii) vérifie le lemme. Pour la partie restante de B_a^k, soit la tâche T_{ku} du bloc B_a^k dont la longueur décroit par rapport à la longueur de celle qui la précéde $T_{k,u-1}$. Le temps d'exécution de T_{ku} est égal à 2(n-u). L'exécution de T_{ku} ne peut être terminée avant celle des tâches $T_{k,u+1}$, $T_{k,u+2}$, ..., $T_{k,ap}$ ne commence: toutes les deux unités de temps au moins un processeur arrive au bloc B_a^k (lemme 4.3.), ce qui entraîne que dans au plus 2(ap-u) unités de temps l'exécution de $T_{k,ap}$ commence. Pour le bloc B_{a+1}^k et le cas iii) même raisonnement : la longueur des tâches décroit et tout processeur exécutant une tâche de l'un des deux blocs ne peut pas finir son exécution avant que l'exécution des tâches suivantes du même bloc ne commence.

Lemme 4.5. L'algorithme présenté respecte les contraintes d'ordonnancement.

Preuve

Les contraintes d'ordonnancement sont vérifiées :

1- Pour les niveaux N_k tels que $1 \leq k \leq n-p+1$

D'après le lemme 4.4., les contraintes d'ordonnancement sont vérifiées: un processeur exécute au plus une seule tâche de chacun des blocs B_i^k $(1 \leq i \leq a+1)$. Après avoir terminé l'exécution du bloc B_i^k, les p processeurs passent à l'exécution du bloc suivant du même niveau. Aucune tâche du bloc suivant B_{i+1}^k ne se termine avant la fin de l'exécution du bloc B_i^k. Ce qui entraîne que les tâches du bloc B_i^{k+1} se libèrent des contraintes d'ordonnancement.

2- Pour les niveaux N_k tels que $n-p+2 \leq k \leq n-1$

Par définition de l'algorithme, les p processeurs exécutent les p premières tâches de chaque niveau. Donc les n-p-1 autres tâches du même niveau sont libérées des contraintes d'ordonnancement. L'exécution des p premières tâches respecte les contraintes d'ordonnancement. En effet, si q processeurs passent d'un niveau au suivant, ils peuvent débuter l'exécution des q premières tâches du niveau correspondant: ces derniers processeurs ont déjà exécuté des tâches du premier bloc du niveau précédent et comme les p premières tâches ont la même longueur ($n \geq 2p$ et k vérifie $n-p+2 \leq k \leq n-1$) alors les processeurs finissent l'exécution de ces dernières dans l'ordre où ils les ont commencées.

Les processeurs sont actifs pendant toute la durée de l'exécution. Le temps mis est égal à T_1/p. L'algorithme dans ce cas est toujours asymptotiquement optimal.

Proposition 4.13. Pour $\alpha \in [0,1/2]$, il existe un algorithme s'exécutant en temps $T_p = T_1/p$ avec une efficacité asymptotique égale à 1.

Nous allons détailler l'exécution de l'algorithme présenté sur un exemple. Choisissant n=13 et p=4, les processeurs opèrent selon le schéma suivant (ij dans la colonne P_s signifie que le processeur P_s effectue la tâche R_{ij} après l'exécution de celle située au dessus). On ne tient pas compte de l'exécution des tâches $T_{i,n+1}$ $(1 \leq i \leq n-1)$:

P_1	P_2	P_3	P_4
11	12	13	14
18	17	16	15
19	1,10	1,11	1,12
24	23	22	21
27	28	26	25
29	2,11	2,10	2,12
34	31	32	33
35	36	37	38
3,12	3,11	3,10	39
41	42	43	44
45	46	47	48
49	4,10	4,11	4,12
54	53	52	51
58	57	56	55
59	5,10	5,11	5,12
64	63	62	61
68	67	66	65
6,12	6,11	6,10	69
71	72	73	74
75	76	77	78
79	7,10	7,11	7,12

84	83	82	81
88	87	86	85
8,12	8,11	89	8,10
91	92	94	93
95	96	98	97
99	9,10	9,12	9,11
10,4	10,3	10,1	10,2
10,8	10,7	10,5	10,6
10,12	10,11	10,9	10,8
11,1	11,2	11,3	11,4
11,5	11,6	11,7	11,8
11,9	11,10	11,11	11,12
12,2	12,3	12,4	12,1
12,6	12,7	12,8	12,5
12,10	12,11	12,12	12,9

Dans l'exemple, on vérifie qu'un processeur n'exécute pas deux tâches d'un même bloc.

5. PRALLELISATION DE LA PHASE 2 AVEC DUPLICATION COMPLETE

En se servant d'une matrice $V=(b_{ij})_{1\leq i,j\leq n}$ pour stocker les éléments calculés de l'inversion de $U=(a_{ij})_{1\leq i,j\leq n}$, la partition en tâches de l'algorithme ci-dessus devient

pour i :=1 **à** n
début
  exécuter $R_{i,i}$:
  < $b_{ii}^{(i)}:=a_{ii}^{(i)}$;>
  pour j := i+1 **à** n
  début
    exécuter $R_{i,j}$:
    <$a_{ij}^{(i)} := b_{ii}^{(i)}$ (x) $a_{ij}^{(i-1)}$;
    pour k := i+1 **à** j-1
      $a_{ij}^{(k)} := a_{ij}^{(k-1)}$ (+) $a_{ik}^{(k)}$ (x) $a_{kj}^{(k-1)}$
    $b_{ij}^{(j)} := a_{ij}^{(j-1)}$ (x) $b_{jj}^{(j)}$; >
  fin ;
fin ;

Le temps d'exécution de la tâche $R_{i,j}$ est 2(j-i) unités de temps. Puisque la duplication d'une colonne a un coût nul, on néglige le temps mis pour exécuter les tâches $R_{i,i}$ ($1\leq i\leq n$).

Les contraintes d'ordonnancement sont

$$R_{ij} << R_{i,j+1} \qquad 1 \leq i \leq n-1 \quad i+1 \leq j \leq n$$

Le graphe d'ordonnancement a l'allure suivante:

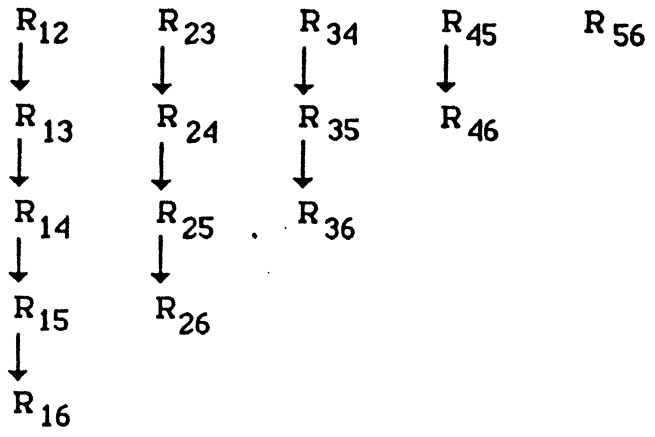


Figure 16: Graphe d'ordonnancement de l'inversion d'une matrice triangulaire avec duplication (matrice de taille 6x6).

Le plus long chemin du graphe est $\{R_{12}, R_{13}, ..., R_{1n}\}$ d'où la valeur

$$T_{opt} = \Sigma_{2 \leq i \leq n} \; 2(i-1) = n^2+O(n)$$

Il est évident de construire un algorithme s'exécutant en T_{opt} avec n-1 processeurs. L'efficacité asymptotique de cet algorithme est égale à $(n^3/3)/n^3 = 1/3$.

Etant donné p processeurs notés P_1, P_2, P_3, ..., P_p, pour définir un algorithme parallèle, on partage le graphe d'ordonnancement en deux parties A et B (voir figure 17). L'exécution de la première partie s'effectue au début, dès qu'un proceseur se libère dans cette partie, il passe à l'exécution des tâches de la partie B. La partie A (respectivement B) est constituée par les p premières (respectivement n-p+1 dernières) colonnes. Le commencement de l'exécution de la partie B aura lieu dès la fin du niveau N_{n-p} de la première partie. Dans la partie A, chaque processeur est affecté à une colonne. A partir du niveau N_{n-p}, un processeur se libère à la fin de chaque niveau de cette première partie. Il sera affecté au pool Q des processeurs effectuant les tâches de la seconde partie. L'exécution de la partie B s'effectue par l'algorithme Glouton G diagonale par diagonale: l'algorithme G exécute les tâches de la partie B en balayant les diagonales de la seconde partie de bas en haut, en commençant à tout instant le maximum de tâches exécutables (une diagonale d_j de B est formée par l'ensemble des tâches T_{ij} où i=p+1, p+2, ..., j-1 où $j \geq p+2$). Si à un instant donné q processeurs ($q \leq p$) sont disponibles, l'algorithme G les affecte à l'exécution des q tâches suivantes. Plus précisement, l'algorithme G exécute les tâches de la partie B dans l'ordre partiel suivant:

$$T_{p+1,p+2} \leq\leq T_{p+1,p+3} \leq\leq T_{p+2,p+3} \leq\leq T_{p+1,p+4} \leq\leq \ldots \leq\leq$$
$$\ldots \leq\leq T_{n-2,n-1} \leq\leq T_{p+1,n} \leq\leq T_{p+2,n} \leq\leq \ldots \leq\leq T_{n-2,n} \leq\leq T_{n-1,n}.$$

Un processeur passant du pool P au pool Q ne peut débuter l'exécution d'une tâche de la partie B que s'il existe un autre processeur du même pool abordant l'exécution d'une autre tâche. C'est à dire le nouveau processeur arrivant au pool Q doit se synchroniser avec au moins un processeur du même pool. Cela n'affecte pas l'évaluation asymptotique.

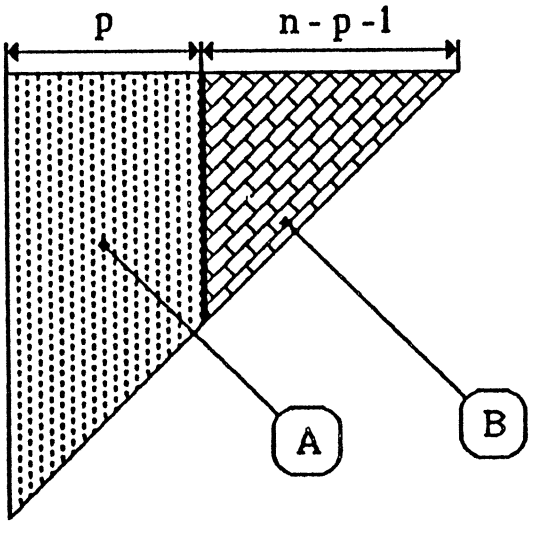


Figure 17: Partage du graphe d'ordonnancement

Plutôt que d'expliciter formellement l'algorithme dans le cas général, nous allons détailler son exécution sur un exemple (voir figure 18). En choisissant n=10 et p=3, nous avons l'ordonnancement suivant avec la notation suivante:

(s)[t] : signifie que le processeur P_s débute l'exécution de la tâche à l'instant t.

On remarque que si on démontre que l'algorithme présenté s'exécute en T_p pour p=n/3, alors l'exécution avec p processeurs tels que p≥n/3 s'effectue en T_p aussi. En effet, la partie A dans tous les cas possède seulement p colonnes tandis que la partie B, elle possède n-p-1 colonnes. Pour n fixé, en augmentant le nombre de processeurs p, le nombre de colonnes formant la partie B diminue. Par conséquent, le temps mis pour exécuter B diminue si p augmente tout en laissant la valeur de n constante. Autrement dit, l'exécution de la partie B en disposant p processeurs p>n/3, est plus rapide qu'en disposant p processeurs avec p=n/3 tout en gardant n constante. Par contre, le temps d'exécution de la partie A est constant.

Lemme 4.6. Pour p≤n/3, le nombre de tâches situées sur la diagonale en cours d'exécution de la partie B est supérieur ou égal à 2q-1 où q est le nombre de processeurs du pool Q.

Pour établir le lemme, il suffit de comparer pour j= 2q-1, la somme

$$S_1(j) = \Sigma_{2 \le i \le j} (i^2-1) = j(j+1)(2j+1)/6 + j - 3$$

des temps des diagonales de B: $d_1, d_2, \ldots, d_j$ avec la somme

$$S_2(q) = (n-p)q(q+1) + q(q+1)(2q+1)/3 + q$$

des temps d'activité des q processeurs dans la partie B ($1 \le q \le p$). L'inégalité $S_1(2q-1) \le S_2(q)$, toujours vérifiée pour tout $p \le n/3$, assure que le lemme est vérifié.

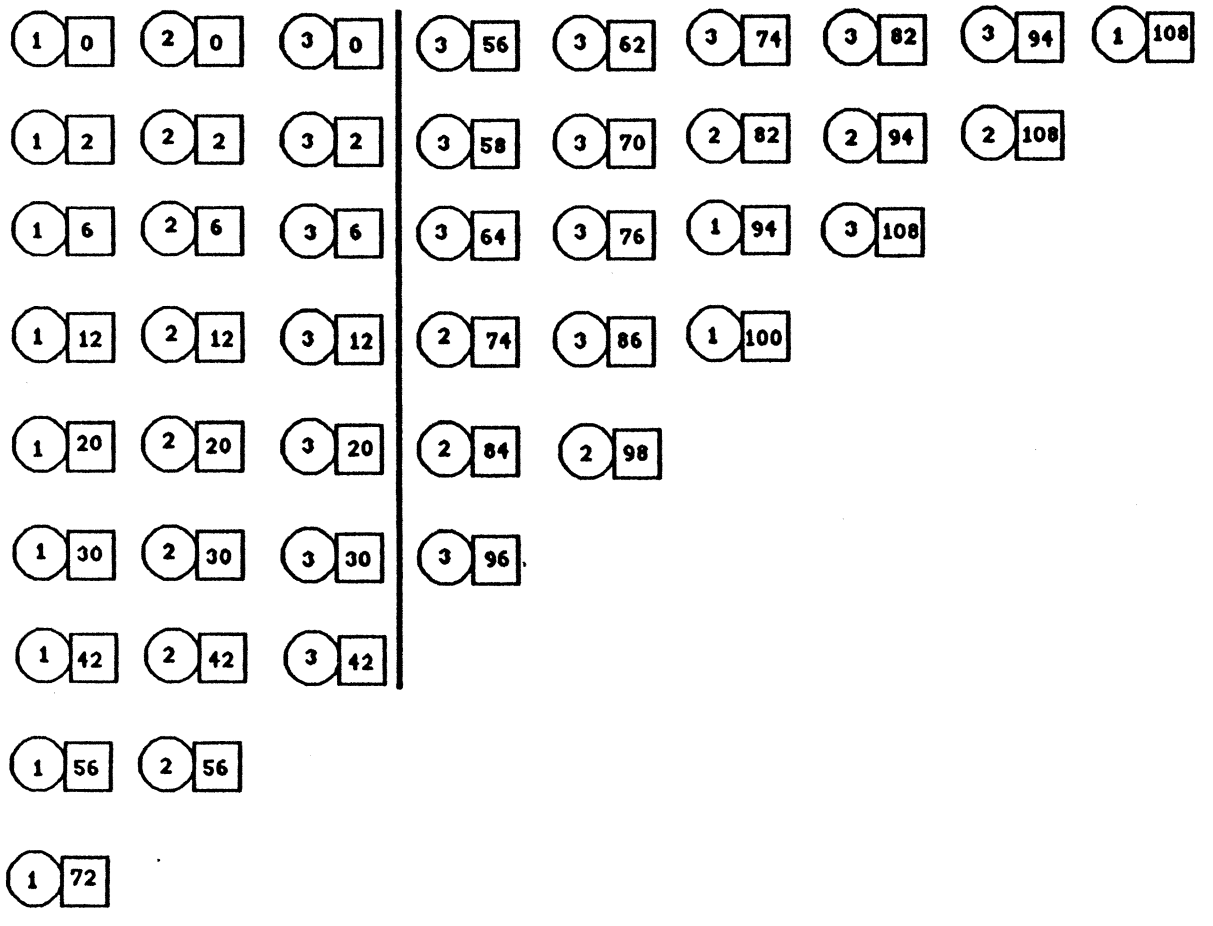


Figure 18

Lemme 4.7. L'algorithme respecte les contraintes d'ordonnancement.

Dans la partie A, le lemme est vérifié puisque l'exécution se fait par colonne. Pour démontrer le lemme dans la partie B, remarquons que la longueur des tâches d'une même diagonale décroit d'une tâche à la suivante de deux unités de temps dans l'ordre de l'algorithme Glouton. D'après le lemme 4.6. précédent, on démontre que les contraintes d'ordonnancement sont respectées(preuve similaire à celle de la proposition 4.10. de ce chapitre).

Pour évaluer le temps d'exécution de l'algorithme, nous notons que tous les processeurs sont actifs sans interruption pour $p \leq n/3$: en effet, dans la partie B le nombre de tâches situées sur la diagonales en cours d'exécution est supérieur à 2q-1 où q est le nombre de processeurs du pool Q (en d'autres termes l'algorithme possède une efficacité asymptotique 1). Nous obtenons le temps d'exécution de l'algorithme pour $p \leq n/3$ $T_p=T_s/p$ où T_s est le temps d'exécution séquentielle. Pour $p \geq n/3$, le temps mis est $T_{opt}=n^2 + O(n)$.

Proposition 4.14. i) L'algorithme présenté est asymptotiquement optimal

ii) Pour $p \geq n/3$, il exécute le graphe des tâches de l'inversion d'une matrice avec duplication complète en temps T_{opt} avec une efficacité asymptotique $e_{p,\infty}=1/3\alpha$

iii) Pour $p<n/3$, il exécute le graphe des tâches de l'inversion d'une matrice avec duplication complète en temps T_s/p avec une efficacité asymptotique égale à 1.

6. COMPARAISON DES DIFFERENTES VERSIONS

6.1. RAPPELS

Soit A une matrice carrée d'ordre n . Rappelons que si B est l'inverse de A alors on a $AB=I_n$ (I_n est la matrice identité d'ordre n). Ecrivons B comme l'ensemble de ses vecteurs colonnes :

$$[\, B_{.1}\ B_{.2}\ \ldots\ B_{.j}\ \ldots\ B_{.n}\,]$$

Rappelons aussi que AB peut s'écrire

$$[\, AB_{.1}\ AB_{.2}\ \ldots\ AB_{.j}\ \ldots\ AB_{.n}\,]$$

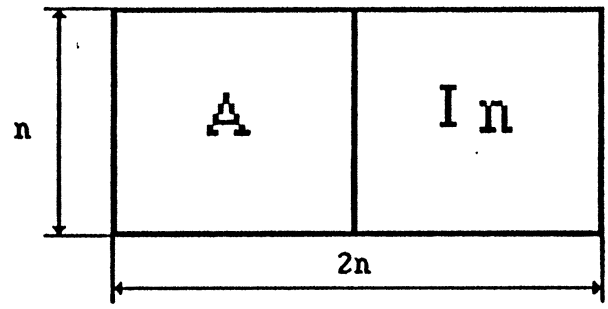
La matrice B est donc l'inverse de A si et seulement si

$$[\, AB_{.1}\ AB_{.2}\ \ldots\ AB_{.j}\ \ldots\ AB_{.n}\,]=[\, e_1\ e_2\ \ldots\ e_j\ \ldots\ e_n]$$

où e_j est le $j^{ème}$ vecteur de la base canoniqe.
C'est à dire si et seulement si

$$AB_{.j} = e_j \quad \text{pour} \qquad j=1, 2, \ldots, n.$$

Calculer l'inverse de A, c'est donc résoudre les n systèmes linéaires ci dessus. Ces derniers ont la même matrice, on peut donc les résoudre en parallèle par la méthode de triangularistion. Alors on a le tableau suivant :



Dans ce qui suit, nous présentons brièvement les différentes méthodes pour résoudre n systèmes linéaires possédant la même matrice. Pour plus de détails, nous renvoyons le lecteur au chapitre II de cette thèse et les travaux de Cosnard, Robert et Trystram [9, 10].

6.1.1. Inversion d'une matrice par la méthode de Gauss

Inverser la matrice A à l'aide de la méthode d'élimination de Gauss c'est triangulariser A en répercutant les transformations sur chacune des vecteurs e_1, e_2, ..., e_n et résoudre les n systèmes triangulaires. Ces derniers sont indépendants deux à deux. Par conséquent, leur exécution en parallèle est possible avec p processeurs. Pour la triangularisation en parallèle d'une matrice à l'aide de la méthode d'élimination de Gauss, sept versions parallèles de cet algorithme ont été étudiées au chapitre II. Pour obtenir le graphe des tâches d'une version il suffit d'ajouter au graphe d'ordonnancement les tâches correspondantes au second membre. Initialement, ce dernier est la matrice unité. On ne considère pas les tâches de la partie supérieure nulle de cette matrice puisqu'elle ne sera pas modifiée. Le graphe des versions (KJI), (JKI) se transforme en un graphe possédant n niveaux, chacun d'eux contient n tâches de même longueur: à chaque niveau une tâche de la matrice A disparaît, une autre de la matrice I_n apparaît. Le graphe triangulaire (1) de la forme (KJI) modifiée subit la même transformation. Le nombre tâches U_{ij} par niveau (respectivement T_{ij}) de la version (IJK) (respectivement réduction de Doolittle (1)) n'est pas modifié par rapport à celui de l'élimination de Gauss. Il devient constant à n par niveau pour les tâches T_{ij} (respectivement U_{ij}). De même pour les graphes des deux versions modifiées (IJK) modifiée et réduction de Doolittle (2). Dans tous les cas, la longueur d'une tâche reste égale à la longueur de la même tâche dans le cas de la méthode d'élimination de Gauss. Le temps séquentiel est donc égal à $n^3 + O(n^2)$. Les algorithmes parallèles construit au chapitre II sont encore valables pour exécuter les graphes décrit ci-dessus. Ces algorithmes sont basés sur l'algorithme Glouton

[8, 9,10] qui exécute les tâches du graphe de gauche à droite, en commençant à tout instant le maximum de tâches exécutables. On évalue le temps d'exécution parallèle si on dispose de $p=\alpha n$ processeurs avec $\alpha \in]0,1]$. Pour les versions (KJI) et (JKI) le temps mis pour décomposer la matrice A et modifier I_n est $T_{opt} = 2n^2 + O(n)$ si $\alpha \geq 1/2$: en effet chaque niveau possède n tâches de même longueur. Il est égal à $n^3/p + O(n)$ si $\alpha \leq 1/2$: les processeurs sont actifs pendant tout le temps de l'exécution puisqu'on a plus de 2p tâches par niveau. La décomposition à l'aide des versions (IJK) et réduction de Doolittle (1) nécessite $n^3/p + O(n)$ unités de temps si $\alpha \leq 1/2$, si on dispose n processeurs, elle s'effectue en $T_{opt}= 2n^2 + O(n)$. La décomposition d'une matrice dense et la modification du second membre en se servant d'une des versions (IJK) modifiée ou réduction de Doolittle (2) s'effectue dans un temps minimal $T_{opt} = n^2 + O(n)$ si on dispose 2n -2 processeurs, si $\alpha \in]0,1]$, le temps mis est $T_p = n^3/p + O(n)$. Pour les autres versions, les processeurs sont tous actifs pendant tout le temps de l'exécution, le temps parallèle mis est donc égal à $n^3/p + O(n)$ pour tout $\alpha \in]0,1]$. Pour obtenir l'inverse d'une matrice, on exécute l'un des graphes précédents et en plus on résoud n systèmes linéaires triangulaires. Pour résoudre les n systèmes triangulaires en parallèle, on affecte simplement $\lceil n/p \rceil$ systèmes à chaque processeur. Le temps d'exécution parallèle des n systèmes triangulaires est donc $\lceil n/p \rceil * n^2 + O(n)$. Si nous disposons $O(n^2)$ processeurs, il est de l'ordre de O(n) unités de temps [40]: c'est négligeable devant le temps mis pour décomposer la matrice et le temps mis pour inverser une matrice à l'aide de la méthode de Gauss devient celui de la décomposition de la matrice. Pour p=n, le temps d'exécution parallèle pour la résolution de n systèmes triangulaires est $n^2+O(n)$. Pour inverser une matrice dense à l'aide de la méthode de Gauss, le temps parallèle mis à l'aide une des versions (JKI) et (KJI) est égal à $(2 + \lceil n/p \rceil)n^2 + O(n)$ si $\alpha \geq 1/2$, il est égal à $(\lceil n/p \rceil + n/p)n^2 + O(n)$ pour toutes les versions si $\alpha \leq 1/2$. Dans le cas où $\alpha \geq 1/2$ et en se servant de l'une des versions (IJK) modifiée, réduction de Doolittle (2) ou (KJI) modifiée il est égal à $(\lceil n/p \rceil + n/p)n^2 + O(n)$.

6.1.2. Inversion d'une matrice par la méthode de Jordan

D'après l'etude faite par Cosnard, Robert et Trystram [9, 10], la résolution d'un système linéaire par la méthode de Jordan possède deux versions parallèles une par lignes et l'autre par colonnes. Pour résoudre n systèmes linéaires possédant la même matrice, Les mêmes versions sont encore valables. Le graphe de précédence de la version par lignes pour résoudre n systèmes linéaires est identique à celui de la même version pour résoudre un seul système linéaire. C'est le graphe présenté à la figure 11 de ce chapitre. La seule différence réside dans la longueur d'une tâche T_{ij}. Dans notre cas, elle est égale à 2(2n-i) si $i \neq j$, 2n-i si i=j. Pour la version par colonnes, son graphe de précédence est composé d'un graphe triangulaire pour la diagonalisation de la matrice et d'un graphe rectangulaire nxn pour la modification du second membre. Le nombre de tâches décroit par niveau:

le niveau N_1 du graphe contient 2n-1 tâches tandis que le dernier niveau N_n contient n tâches. La longueur d'une tâche quelconque est égale à 2n-1 unités de temps. Cosnard, Robert et Trystram [9, 10] ont construit des algorithmes pour ces deux versions dans le cas de la résolution d'un seul système linéaire. Ces algorithmes sont encore valables dans notre cas. Le temps optimal de l'algorithme de la version par lignes est $9n^2/2 + O(n)$, il est égal à $2n^2 + O(n)$ pour celui de la version par colonnes. Le premier algorithme a besoin seulement de $\lceil 2n/3 \rceil$ processeurs tandis que le second nécessite 2n-1 processeurs. En disposant $p=\alpha n$ processeurs, le temps mis pour exécuter l'algorithme de l'une des versions est $3n^2/\alpha + O(n)$ ($\alpha \in]0,2/3]$ pour la version par lignes, si la version est par colonnes $\alpha \in]0,1]$).

6.1.3. Inversion d'une matrice par la méthode de Huard

Pour résoudre un système linéaire à l'aide de la méthode de Huard, deux versions parallèles sont possibles: version colonnes - lignes et version colonnes - colonnes [9,10]. Le graphe des tâches de la version colonnes - colonnes est similaire à celui de l'inversion d'une matrice par la méthode de Jordan version par colonnes. La longueur d'une tâche T_{ij} est égale à 4i-1 unités de temps. Pour la version colonnes - lignes, il est préférable de consulter les travaux [9,10] afin d'avoir une idée correcte. Nous nous contentons dans cette présentation de donner les temps optimaux des deux versions citées. Le temps optimal de chacune des versions est égal à $2n^2 + O(n)$. Pour exécuter l'algorithme de Cosnard, Robert et Trystram, il faut 2n-2 processeurs pour la version colonnes - lignes, 2n-1 pour la version colonnes - colonnes.

6.2. INVERSION D'UNE MATRICE TRIANGULAIRE

Il est très intéressant de remarquer comment l'efficacité des algorithmes parallèles peut être améliorée en autorisant le stockage de certaines variables. Le tableau 1 illustre les performances optimales des différentes versions parallèles de l'inverse d'une matrice triangulaire. Il est clair que la version avec duplication complète possède des performances meilleures que celles des autres versions.

6.3. INVERSION D'UNE MATRICE DENSE

Le temps d'exécution séquentiel pour inverser une matrice dense à l'aide de la méthode de Gauss est $2n^3 + O(n^2)$: c'est la somme des deux temps de triangularisation de la matrice et résolution de n systèmes linéaires. Il est égal au temps séquentiel pour les versions de Rote. Pour les méthode de Jordan et de Huard il est égal respectivement à $3n^3 + O(n^2)$ et $8n^3/3 + O(n^2)$. Séquentiellement, Inverser une matrice à l'aide de l'un des algorithmes de Gauss ou de Rote est meilleur que l'algorithme de Jordan ou de Huard au point de vue opérations arithmétiques. Les temps optimaux d'exécution parallèle de l'algorithme de Rote sans duplication, avec duplication temporaire sont respectivement $6n^2+O(n)$, $4n^2+O(n)$. Pour la version présentée par Robert et

Trystram [47], le temps parallèle optimal est égal à $3n^2+O(n)$. Pour le calcul de T_{opt} de l'inversion d'une matrice à l'aide d'une version de Gauss, on néglige la résolution des systèmes linéaires pour l'inversion à l'aide d'une version de Gauss puisqu'ils s'exécutent en temps O(n) avec $O(n^2)$ processeurs. Dans ce cas, l'inverse d'une matrice à l'aide d'une version parallèle de Gauss est meilleure par rapport à toute autre version: le temps optimal mis pour inverser une matrice est $n^2+O(n)$ en utilisant l'une des versions (IJK) modifiée, réduction de Doolittle ou (KJI) modifiée, il est égal à $2n^2+O(n)$ pour les autres versions de Gauss. Le tableau 2 illustre les temps optimaux T_{opt} et $T_{opt}(p)$ des différentes versions de l'inversion d'une matrice. $T_{opt}(p)$ de l'une des versions (IJK) modifiée ou réduction de Doolittle (2) est meilleur par rapport à celui de la version (KJI) modifiée pour la raison que le nombre de processeurs est deux fois plus grand dans le premier cas que dans le second cas: la résolution des n systèmes triangulaires est plus rapide dans le premier cas que dans le second cas. La version de Rote sans duplication est la version la moins performante. Le nombre de processeurs p indiqué dans le tableau 2 n'est pas égal à p_{opt} pour toutes les versions.

Les versions parallèles	p	T_{opt}	$e_{p,\infty}$
Sans duplication	$\frac{2-\sqrt{2}}{2}n$	$2n^2+O(n)$	$\frac{2+\sqrt{2}}{6}=$ 0.56...
Avec duplication temporaire	$n/2$	$n^2+O(n)$	$2/3=$ 0.666...
Avec duplication	$n/3$	$n^2+O(n)$	1

Tableau 1

Inversion à l'aide des méthodes	p	$T_{opt}(p)$	T_{opt}
(JKI) ou (KJI)	$\lceil n/2 \rceil$	$4n^2 + O(n)$	$2n^2 + O(n)$
(IJK) modifiée ou réduc. de Doo. (2)	2n	$3n^2/2 + O(n)$	$n^2 + O(n)$
(KJI) modifiée	n	$2n^2 + O(n)$	$n^2 + O(n)$
(IJK) ou réduction de Doolittle (1)	n	$3n^2 + O(n)$	$2n^2 + O(n)$
Rote sans duplication	n - 1	$6n^2 + O(n)$	$6n^2 + O(n)$
Rote avec duplication tempo.	n - 1	$4n^2 + O(n)$	$4n^2 + O(n)$
Robert et Trystram	$\lceil 2n/3 \rceil$	$3n^2 + O(n)$	$3n^2 + O(n)$
Jordan par lignes	$\lceil 2n/3 \rceil$	$9n^2/2 + O(n)$	$9n^2/2 + O(n)$
Jordan par colonnes	2n - 1	$2n^2 + O(n)$	$2n^2 + O(n)$
Huard colonnes -lignes	2n - 2	$2n^2 + O(n)$	$2n^2 + O(n)$
Huard colonnes -colonnes	2n - 1	$2n^2 + O(n)$	$2n^2 + O(n)$

Tableau 2

CONCLUSION

Dans le chapitre II, à partir d'un algorithme séquentiel, nous pouvons avoir plusieurs versions parallèles. Pour certaines versions parallèles correspondent d'autres versions modifiées. Ces dernières possédent des résutats de complexités meilleurs que ceux des versions initiales. Prenons comme exemple la décomposition de Cholesky. Nous avons montré qu'il est possible d'avoir un graphe de tâches triangulaire plutôt que le graphe 2-pas présenté par Cosnard et Coll. [8]. Ce dernier possède un temps optimal plus grand que le graphe triangulaire et jusqu'à maintenant, nous ne savons pas un algorithme optimal exécutant les différents cas de ce graphe. Et même ce dernier existe, il ne peut pas être meilleur que celui du graphe triangulaire.

Le chapitre III met en évidence un algorithme asymptotiquement optimal pour la méthode d'élimination de Gauss avec pivotage partiel. Il est difficile de construire un algorithme optimal pour le graphe 2-pas. Quand les tâches diagonales et la tâche non diagonale d'un même niveau possèdent la même longueur, nous avons présenté un algorithme asymptotiquement optimal. Cette analyse s'étend au cas où $Ex(T_{kk})=a(n-k)+b$ et $Ex(T_{kj})=c(n-k)+d$, $j\neq k$. De plus, on trouvera dans [40] une construction d'un algorithme asymptotiquement optimal pour le cas

$$Ex(T_{kk})=a$$
$$Ex(T_{kj})=b$$

qui modélise la résolution itérative d'un système linéaire par la méthode de Gauss-Seidel.

Nous montrons dans le chapitre IV qu'avec duplication temporaire nous pouvons premièrement se débarrasser de l'hypothèse duplication complète et deuxièment améliorer les complexités des phases 2 et 3 de l'algorithme de Rote. De même pour la duplication complète des éléments à transformer, nous constatons l'amélioration des résultats de complexité de la version duplication temporaire. Les algorithmes parallèles qui sont définis pour les trois phases de l'algorithme de Rote dans le cas de l'inversion d'une matrice dense sont aussi valables pour la détermination de la fermeture reflexive et transitive d'une relation binaire et la résolution du problème des plus courtes distances dans un graphe pondéré. La classe d'algorithmes parallèles définis pour la version de Robert et Trystram possède une excellente implémentation sur une architecture parallèle. Enfin, en négligeant le temps de la résolution des n systèmes triangulaires, l'étude théorique montre que l'inversion d'une matrice en parallèle par une des versions parallèles de la méthode d'élimination de Gauss est meilleure au point de vue temps d'exécution optimal T_{opt} que celle de toutes les versions de l'algorithme de Rote et les versions de Jordan et Huard.

Nous avons considéré des systèmes où le coût d'une tâche opèrant sur q données est une fonction affine de q: $\alpha q+\beta$
où α et β peuvent prendre compte

- de l'arithmétique
- des communications.

Une construction importante: le coût d'une tâche doit rester indépendant du numéro de processeur qui l'exécute, et ceci exclut les systèmes à mémoire distribuée. Si on veut vraiment modéliser une architecture avec un grand nombre de processeurs, seule une solution distribuée semble réaliste. Un modèle reste à construire.

A tout le moins, même si une analyse avec $p=\alpha n$ processeurs, où n tend vers l'infini, sur une mémoire partagée n'est cependant pas réaliste. Les résultats de complexité sont des critères qui permettent de sélectionner les meilleurs algorithmes parallèles lors d'une implémentation.

REFERENCES

[1] H.M. AHMED, M.MORF et J.M.DELOSME, Highly concurrent computing structures for matrix arithmetic and signal processing, IEEE Comptuer 15,1, (1982), 65-82

[2] M. AUGUIN, F. BOERI, Une architecture pour le calcul vectoriel et parallèle OPSILA, in "Future trends in computing", Ed. CHENIN et al. Masson-Wiley, (1985), 222-226

[3] M. AUGUIN, F. BOERI, Efficient multiprocessor architecture for digital signal processing, ICASSP (1982), 675-679

[4] M. AUGUIN, F. BOERI, Réseaux d'interconnexion et leurs commandes asynchrones, dans " Parallélisme, Communication et Synchronisation", Ed. J.P VERJUS et G. ROUCAIROL, 489-517

[5] L. N. BHUYAN, Interconnection networks for parallel and distributed processing, IEEE computer, June (1987), 9-12

[6] E.G. COFFMAN Jr., P.J. DENNING, Operating system theory, Prentice Hall, Englewood Cliffs, NJ, (1973)

[7] E.G. COFFMAN Jr., Computer & job-shop scheduling theory, John Wiley and Sons, New York, 1976

[8] M. COSNARD, M. MARRAKCHI, Y. ROBERT, D. TRYSTRAM, Gauss elimination algorithms for MIMD computers, Proc. CONPAR 86, W. Händler et al. eds., Lecture Notes in Computer Science 237, Springer Verlag (1986), 247-254

[9] M. COSNARD, Y. ROBERT, D. TRYSTRAM, Résolution parallèle de systèmes linéaires denses par diagonalisation, Bulletin EDF, C, 2, (1986), 67-87

[10] M. COSNARD, Y. ROBERT, D. TRYSTRAM, Comparaison des méthodes parallèles de diagonalisation pour la résolution de systèmes linéaires denses, C. R. Acad. Sc. Paris 301, I, 16, (1985), 781-784

[11] M. COSNARD, Y. ROBERT, Algorithmique parallèle: une étude de complexité, Techniques et Science Informatique 6, 2 (1987), 115-125

[12] M. COSNARD, D. TRYSTRAM, Communication complexity of gaussian elimination on MIMD shared memory computers, rapport 3, Ecole Centrale des Arts et Manufactures, Paris

[13] L. CSANKY, Fast parallel matrix inversion algorithms, SIAM Review 5, 4, (1976,) 618-623

[14] Z. CVETANOIV, The effect of problem partitioning, allocation, and granularity on the performa)nce of multiple-processeur systems, IEEE Transactions on Computers, 36, 4, April (1987, 421-432

[15] J.J. DONGARRA, F.G. GUSTAVSON, A. KARP, Implementing linear algebra algorithms for dense matrices on a vector pipeline machine, SIAM Review 26, 1,

(1984), 91-112

[16] C. EISENBEIS, J. EHREL, Expériences sur le calculateur vectoriel ST100, in "Future trends in computing", Ed. CHENIN et al. Masson-Wiley, (1985), 250-254

[17] J. ERHEL, A. LICHNEWSKY, F. THOMASSET, Vectorizing finite element methods, in "Future trends in computing , Ed. CHENIN et al. Masson-Wiley, (1985), 255-269

[18] J. ERHEL, W. JALBY, A. LICHNEWSKY, F. THOMASSET, Quelques progrès en calcul parallèle et vectoriel, 6ème colloque international sur les méthodes de calcul scientifique et technique, (1983), Glowinski et Lions eds

[19] M. FEILMEIER, Parallel computers - Parallel mathematics, IMACS North Holland, (1977)

[20] J.A. FISCHER et J.J. O'DONNELL, VLIW machines: multiprocessors we can actually program", COMPCON (1984), 299-305

[21] M.J. FLYNN, Very high-speed computing systems, Proc. IEEE 54 (1966), 1901-1909

[22] C. FRABOUL et N. HIFDI, LESTAP: expression et mise en oeuvre d'applications parallèles, R. R. 1/3192, ONERA-CERT, Toulouse

[23] D.D. GAJSKI, J.K. PEIR, Essential issues in multiprocessors systems, IEEE Computer, June (1985), 9-27

[24] W.M. GENTELMAN et H.T KUNG, Matrix triangularisation by systolic arrays, Conf.S.P.I.E.Real-time Signal Processing IV 298,(1981), 19-26

[25] G. H. GOLUB, C. F. VAN LOAN, Matrix computation, The Johns Hopkins University Press, (1983)

[26] D. HELLER, A survey of parallel algorithms in numerical linear algebra, SIAM Review 20, (1978), 740-777

[27] R.W. HOCKNEY, C.R. JESSHOPE, Parallel computers: architectures, programming and algorithms, Adam Helger, Bristol, (1981)

[28] W.HOFMANN, Gaussian Elimination Algorithms on a vector computer, Report 85-10,Dept of Mathematics,University of Amesterdam,(1985)

[29] K. HWANG, F. BRIGGS, Parallel processing and computer architecture, MC Graw Hill, (1984)

[30] D.J. KUCK, The structure of computers and computations: volume one, Wiley & Sons (1978)

[31] H.T. KUNG, Why systolic architectures, IEEE Computer 15 (1982), 37-46

[32] S.P. KUMAR, Parallel algorithms for solving linear equations on MIMD computers, PhD. Thesis, Washington State University,' (1982)

[33] S.P. KUMAR, J.S. KOWALIK, Parallel factorization of a positive definite matrix on an MIMD computer, Proc. Int. Conf. ICCD 84 (1984), 410-416

[34] J.C. KONIG, Communication personnelle

[35] J. LENFANT, Mémoires parallèles et réseaux d'interconnexion, TSI 1,2 (1982), 135-142

[36] A. LICHNEWSKI, F. THOMASSET, Techniques de base sur l'exploitation automatique du parallélisme dans les programmes, Rapport de Recherche INRIA n° 460, Décembre (1985)

[37] R.E. LORD, J.S. KOWALIK, S.P. KUMAR, Solving linear algebraic equations on an MIMD computer, J. ACM 30, 1, (1983), 103-117

[38] B.NETA, H.M.TAI, LU Factorisation on parallel computers, Com.&Maths.with Appls. 11,6,(1985), 573-579

[39] M. MARRAKCHI, Y. ROBERT, Un algorithme parallèle pour la méthode de Gauss. C.R. Acad. Sc. Paris 303, série I (1986), 425-429

[40] M. MARRAKCHI, Y. ROBERT, Optimal sheduling algorithms for parallel iterative methods on multiprocessor systems, rapport 693, Janvier (1988), IMAG, Laboratoire TIM3, Grenoble

[41] M. MARRAKCHI, Y. ROBERT, Optimal algorithms for gaussian elimination on a MIMD computer, rapport technique 34, Janvier (1988), IMAG, Laboratoire TIM3, Grenoble

[42] N.M. MISSIRILS, F. TAJAFERIS, Parallel matrix factorizations on shared memory MIMD computer, 1st International Conference (1987), E.N. HOUSTIS et al. Eds., Lecture notes in computer science 297, Springer-Verlag, 926-938

[43] D.A. PADUA, D.J. KUCK et D.L. LAWRIE, High speed multiprocessor and compilation techniques, IEEE Trans. Comp. 29 (1980), 763-776

[44] D.C. POLYCHRONOPOULOS, U. BANERJEE, Processor allocation for horizontal and vertical parallelism and related speedup bounds, IEEE computer vol C-36, Avril (1987), 410-420

[45] D.C. POLYCHRONOPOULOS, D.J. KUCK, Guided self-schduling: A practical scheduling scheme for parallel supercomputers, IEEE computer vol C-36, December (1987)

[46] P. QUINTON, Les hyper-ordinateurs, La recherche, 167, Juin (1985), 740-749

[47] Y.ROBERT, D.TRYSTRAM, An orthogonal systolic array for the algebraic path problem, Computing 39, (1987), 199-199

[48] Y. ROBERT, D. TRYSTRAM, Optimal scheduling algorithms for parallel Gaussian elimination, Proc. Int. Symp. High Performance Computer Systems, Paris, December 14-16, (1987), E.Gelenbe editor, to be published by North Holland

[49] Y. ROBERT, P. SGUAZZERO, The LU decomposition algorithm and its efficient FORTRAN implementation on the IBM 3090 vector multiprocessor, rapport de recherche, IMAG, Laboratoire TIM3, Grenoble

[50] Y.ROBERT, Algorithmique parallèle: réseaux d'automates, architectures systoliques, machines SIMD & MIMD, Thèse d'Etat, Université de Grenoble, (1986)

[51] G.ROTE, A systolic array algorithm for the algebraic path problem (short paths; matrix inversion), Computing 34 ,(1985), 191-219

[52] G.ROTE, Communication privée

[53] Y. SAAD, Data communication in parallel architectures, Report DCS/461, Yale University, (1986)

[54] Y. SAAD, Communication complexity of the Gaussian elimination algorithm on multiprocessors, Report DCS/348, Yale University, (1985)

[55] A.SAMEH et D.J.KUCK, On stable parallel linear system solvers, J.ACM.25,(1978), 81-91

[56] A. SAMEH, An overview of parallel algorithms, Bulletin EDF, (1983), 129-134

[57] U. SCHENDEL, Introduction to Numerical Methods for Parallel Computers, Ellis Horwood Series, J. Wiley & Sons, New York, (1984)

[58] J. SCHWARZ, A taxonomic table of parallel computers, based on 55 designs, Courant Institute, N. Y. U. (1983)

[59] P. SPITERI, Parallel asynchronous algorithms for solving Boundary value problems, in "Parallel algorithms & architectures", Ed. M. COSNARD, P. QUINTON, Y. ROBERT, M. TCHUNTE, North Holland, (1986), 73-84

[60] A. STAFYLOPATIS, A. DRIGAS, On the processing time of parallel linear system solver, 1st International Conference (1987), E.N. HOUSTIS et al. Eds., Lecture notes in computer science 297, Springer-Verlag, 994-1010

[61] H.S. STONE, Parallel Computers, In "Introduction to computer architecture", Science Res. Associates, Inc, (1983), 318-374

[62] H.S. STONE, Problems of parallel computation, Proc. Conf. "Complexity of sequential and parallel numerical algorithms", J.F. Traub ed., Academic Press (1973)

[63] C. TIMSIT, J.L. THOMAS, M. SCHMITT, Le supercalculateur ISIS, in "Future trends in computing", Ed. CHENIN et al. Masson-Wiley, (1985), 283-293

[64] P.C. TRELEAVEN, D.R. BROWNBRIDGE, R.P. HOPKINS, Data-driven and demand-driven computer architecture, ACM Comp. Surveys14 (1982), 93-143

[65] D. TRYSTRAM, Quelques résultats de complexité en algorithmique parallèle et systolique, Thèse de l'INPG, Université de Grenoble, Avril 1988

[66] M. VELDHORST, A note on Gaussian elimination with partial pivoting on an MIMD computer, T. R. RUU-CS-84-14, University of Utrecht, The Netherlands (1984)

[67] P. WILSON, OCCAM architecture eases system design - part 1, Computer design (1983), 107-115

[68] O.WING et J.W.HUANG, A Computation Model of Parallel Solution of Linear Equations IEEE.T.C. 29, 7, (1980), 632-638

AUTORISATION DE SOUTENANCE

VU les dispositions de l'article 15 Titre III de l'arrêté du 5 Juillet 1984 relatif aux études doctorales

VU les rapports de présentation de

M. Michel COSNARD

M. Pierre SPITERI

Monsieur MARRAKCHI Mounir

est autorisé à présenter une thèse en soutenance en vue de l'obtention du diplôme de DOCTEUR de l'INSTITUT NATIONAL POLYTECHNIQUE de GRENOBLE, spécialité MATHEMATIQUES APPLIQUEES.

Fait à Grenoble, le 20 Juin 1988

Georges LESPINARD
Président
de l'Institut National Polytechnique
de Grenoble

Résumé

Dans cette thèse, nous nous intéressons à la parallélisation de quelques algorithmes d'algèbre linéaire à l'aide du formalisme du graphe des tâches. Au chapitre I, après avoir décrit brièvement les architectures SIMD et MIMD, nous introduisons les notions de tâche et de graphe des tâches. Aux chapitres II et III, nous étudions la parallélisation des méthodes de triangularisation d'une matrice dense. Diverses versions parallèles sont étudiées et, pour chaque algorithme parallèle construit, nous évaluons le temps d'exécution et l'efficacité. Au chapitre IV, la parallélisation de l'inversion d'une matrice dense est étudiée et des résultats de complexité sont présentés. Durant notre travail, nous nous sommes fixés comme objectif la recherche d'algorithmes optimaux et de résultats de complexité.

Mots clés

Algorithmes parallèles, graphe de tâches, algèbre linéaire, complexité.