Résolution des modèles markoviens sur machines à mémoires distribuées

Abderezak Touzene

▶ **To cite this version:**

Abderezak Touzene. Résolution des modèles markoviens sur machines à mémoires distribuées. Modélisation et simulation. Institut National Polytechnique de Grenoble - INPG, 1992. Français. NNT : . tel-00341389

HAL Id: tel-00341389
https://theses.hal.science/tel-00341389

Submitted on 25 Nov 2008

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

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

Tu 16439

THESE

présentée par

Abderezak Touzene

pour obtenir le grade de **DOCTEUR**

de l'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

(Arrêté ministériel du 23 Novembre 1988)

Spécialité : INFORMATIQUE

==========

RESOLUTION DES MODELES MARKOVIENS SUR MACHINES A MEMOIRES DISTRIBUEES

==========

Date de soutenance : 21 Septembre 1992

Composition de jury :	D. Trystram	Président
	B. Plateau	Directeur de la thèse
	G. Bernard	Rapporteur
	J. C. Konig	Rapporteur
	J. M. Vincent	Examinateur

Thèse préparée au sein du **Laboratoire de Génie Informatique.**

A mon père,

à ma mère,

A mes frères et sœurs,

REMERCIEMENTS

Tout d'abord, je tiens à remercier Brigitte Plateau qui, avec ses conseils ainsi que ses critiques constructives, a bien dirigé mon travail. Cette thèse doit beaucoup à son esprit de rigueur, à son entière disponibilité et surtout à sa patience.

Je remercie Denis Trystram pour l'honneur qu'il me fait en présidant le jury de cette thèse.

Mes remerciements vont également à Guy Bernard et Jean-Claude Konig pour leur acceptation d'être rapporteurs de cette thèse.

Je remercie Jean Marc Vincent qui a accepté de participer au jury.

Que tous les membres du laboratoire LGI et ceux du laboratoire LMC qui ont participé de prés ou de loin à la réalisation de ce travail soient vivement remerciés.

J'adresse mes remerciements à Claire Dicrescenzo et Joelle Prevost de l'équipe technique du laboratoire LMC pour leur aide omniprésente et très appréciable.

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

Enfin, je tiens à exprimer toute ma reconnaissance aux différentes personnes qui ont participé, depuis mon enfance, dans ma formation intellectuelle et scientifique.

Table des matières

1 Présentation de la machine (MIMD) cible MEGANODE **5**

1.1 Introduction 5

1.2 Description du MEGANODE à 128 Transputers 6

1.2.1 Vue générale 6

1.2.2 Réseau d'interconnexion pour Transputers 7

1.3 Modèle de communication pour le MEGANODE 9

1.3.1 Communication à l'intérieur d'un même tandem-node 9

1.3.2 Communication entre deux Transputers appartenant à deux tandem-nodes différents 12

1.3.3 Communication bi-directionnelle 13

1.3.4 Communication sur les liens en parallèle 15

1.4 Performance de l'unité de calcul du Transputer (T800/T414) 16

1.5 Taux de recouvrement des communications par du calcul 17

1.6 Conclusion 19

2 Communications dans les machines à mémoires distribuées **21**

2.1 Introduction 21

2.2 Principaux facteurs qui influent sur le coût de communication 22

2.2.1 Mode de communication 23

2.2.1.1 Communication par commutation de message 23

2.2.1.2 Communication par commutation de circuit 23

2.2.1.3 Comparaison du coût de communication les différents modes 24

2.2.2 Topologie du réseau de communication 26

2.2.3 Type de communications intensive 26

2.2.3.1 Diffusion et multidiffusion (échange total) 27

2.2.3.2 Accumulation et échange total avec accumulation 27

2.2.3.3 Distribution, rassemblement et échange total personnalisé 27

2.2.3.4 Dualité entre les types de communication intensives . . . 28

2.3 Echange total et échange total avec accumulation sur l'anneau 30

2.3.1 Echange total sur l'anneau 30

2.3.2 Echange total avec accumulation sur l'anneau 31

2.4 Echange total sur une topologie torique 33

2.4.1 Procédure d'échange total sur un tore $(\sqrt{p} \times \sqrt{p})$ avec $\sqrt{p}$ impair 34

2.4.1.1 Notation 34

2.4.1.2 Position du problème 35

2.4.1.3 Exemple d'arbre de recouvrement F_{NB} du tore $(\sqrt{p} \times \sqrt{p})$ avec $\sqrt{p}$ impair 36

2.4.1.4 Procédure d'échange total optimal pour le tore $(\sqrt{p} \times \sqrt{p})$ avec $\sqrt{p}$ impair 40

2.4.2 Procédure d'échange total sur un tore $(\sqrt{p} \times \sqrt{p})$ avec $\sqrt{p}$ pair . 42

2.4.2.1 Exemple d'arbre de recouvrement F_{NB}^{E} pour un tore $(\sqrt{p} \times \sqrt{p})$ avec $\sqrt{p}$ pair 43

2.4.2.2 Procédure d'échange total sur un tore ($\sqrt{p} \times \sqrt{p}$) avec $\sqrt{p}$ pair 46

2.4.3 Implémentation de l'échange total 48

2.4.4 Conclusion sur l'échange total sur le tore 51

2.5 Echange total avec accumulation sur le tore 51

2.5.1 Principe de l'échange total avec accumulation sur le tore 51

2.5.2 Procédure d'accumulation en utilisant les arbres de recouvrement 51

2.5.3 Procédure d'échange total avec accumulation dans le tore 52

2.5.4 Implémentation de l'échange total avec accumulation 53

2.6 Procédure de communication échange partiel personnalisé sur le tore . . . 55

2.6.1 Principe de l'échange partiel personnalisé sur le tore 55

2.6.2 Procédure de l'échange partiel personnalisé 56

2.6.3 Remarques sur la procédure de l'échange partiel personnalisé . . . 57

2.7 Conclusion 57

3 Le parallélisme au service de l'évaluation des performances 59

3.1 Introduction 59

3.2 Méthode d'évaluation des performances et leur parallélisation 59

3.2.1 Méthode d'évaluation des performances utilisant directement le système réel 60

3.2.2 Méthode d'évaluation des performances utilisant des modèles du système réel 60

3.2.2.1 Méthodes de modélisation 60

3.2.2.2 Méthodes parallèles de résolution des modèles 61

3.3 Les méthodes numériques et leur parallélisation 62

3.3.1 Les méthodes généralistes 64

3.3.1.1 Méthode du type Jacobi 64

3.3.1.2 Méthode du type Gauss-Seidel 65

3.3.1.3 Comparaison entre les méthodes de type Jacobi et Gauss-Seidel 68

3.3.1.4 Méthodes itératives utilisant la technique de projection . 68

3.3.2 Méthodes itératives spécialisés 70

3.3.2.1 Méthode itérative d'Agrégation-Désagrégation pour les processus (NCD) 71

3.3.2.2 Méthode de Neuts pour les processus (QBD) 75

3.4 Conclusion 76

4 Parallélisation du produit matrice vecteur **77**

4.1 Introduction 77

4.2 Présentation des différentes implantation du produit matrice vecteur . . . 78

4.2.1 Méthode de décomposition par colonne (C) 79

4.2.2 Méthode de décomposition par ligne (L) 80

4.2.3 Méthode de décomposition ligne-colonne (LC) 81

4.3 Comparaison théorique entre les trois méthodes 84

4.4 Expérimentation du produit matrice vecteur sur le MEGANODE 85

4.4.1 Résultat d'accélération des deux méthodes (C) et (LC) 85

4.4.2 Temps de calcul et communication des méthodes (C) et (LC) . . . 86

4.5 Conclusion 88

5 Méthodes itératives parallèles de résolution des problèmes Markoviens **89**

5.1 Introduction 89

5.2 Schéma itératif de base 91

5.3 Schéma itératif avec effet de Gauss-Seidel local 91

5.4 Schéma itératif à retard k 92

5.5 Amélioration du coût de calcul du produit pour un schéma à retard k . . 93

5.6 Schéma asynchrone libre version 1 94

5.7 Schéma asynchrone libre version 2 95

5.8 Résumé des schémas itératifs 96

5.9 Résultats expérimentaux sur les méthodes asynchrone 97

5.9.1 Plan d'expérience et choix des problèmes à tester 97

5.9.2 Matrices creuses 97

5.9.3 Matrices du type presque décomposable (NCD) 98

5.9.4 Matrices tridiagonales par bloc 98

5.9.5 Matrices du type marche aléatoire sur une grille 2D 98

5.9.6 Conclusion 98

5.9.7 Figures 99

6 Parallélisation du produit vecteur par le descripteur d'un RAS 107

6.1 Introduction 107

6.2 Principe de la multiplication vecteur par le descripteur d'un (RAS) . . . 108

6.2.1 Ordre des composantes dans le vecteur π 109

6.2.2 Définitions des mélanges parfaits 110

6.2.3 Mélange parfait des composantes dans une base de dimension 2 . 111

6.2.4 Mélange parfait des composantes dans une base de dimension 3 . 120

6.2.5 Calcul de π [$\bigotimes_{i=1}^{N} M_{ti}$] en utilisant les mélanges parfaits dans une base de dimension 2 122

6.2.6 Calcul de π [$\otimes_{i=1}^{N} M_{ti}$] en utilisant les mélanges parfaits dans une base de dimension 3 124

6.3 Présentation des différentes approches de parallélisation 130

6.4 Première approche : parallélisation du calcul d'un terme (π [$\otimes_{i=1}^{N} M_{ti}$]) . 131

6.4.1 Introduction 131

6.4.2 Parallélisation utilisant les mélanges parfaits dans une base de dimension 2 131

6.4.2.1 Problème de placement pour résoudre les mélanges parfaits dans une base de dimension 2 132

6.4.2.2 Détails de parallélisation de π $\prod_{i=1}^{N}$ [$S_{b_i B_i}$ ($Id_{B_i} \otimes M_i$)] 138

6.4.3 Paralléllisation utilisant les mélanges parfaits dans une base de dimension 3 144

6.4.4 Parallélisation *mixte* combinant les mélanges parfaits dans une base respectivement de dimension 2 et 3 146

6.4.4.1 Impact du choix du niveau k sur la recherche du bon grain de parallélisme 146

6.4.4.2 Evaluation du coût d'une affectation 147

6.4.4.3 Critère de choix de l'ordre lexicographique 148

6.4.4.4 Algorithme de la méthode mixte: 154

6.5 Deuxième approche : parallélisation par distribution des termes de la somme π [$\Sigma_{t=1}^{C}$ ($\otimes_{i=1}^{N} M_{ti}$)] 158

6.6 Troisième approche : parallélisation de la somme et des termes de la somme160

6.6.1 Réseaux toriques pour effectuer les termes et la somme 161

6.6.2 Réseaux semi-toriques pour effectuer les termes et réseaux anneaux pour la somme 162

6.6.3 Réseaux anneaux pour effectuer les termes et réseaux anneaux pour la somme 165

6.6.4 Comparaison entre les trois stratégies et choix de la meilleure solution 165

6.6.5 Etude de l'accélération pour la solution choisie 167

6.7 Comparaison entre les 3 approches de parallélisation et choix de la plus appropriée 168

6.8 Amélioration et expérimentation de l'approche choisie 168

6.8.1 Amélioration de la méthode en utilisant les propriétés des mélanges parfaits 169

6.8.1.1 Comparaison expérimentale entre le mélange amélioré et le mélange parfait naif 171

6.8.1.2 Comparaison expérimentale entre le temps du mélange parfait et le temps du produit correspondant 172

6.8.1.3 Performances de cette approche de parallélisation . . . 172

6.8.2 Amélioration de la méthode en utilisant le recouvrement des communications par les calculs 175

6.8.2.1 Premier cas:le temps des produits locaux est du même ordre que le temps d'un mélange parfait externe 176

6.8.2.2 Deuxième cas : Le temps des produits locaux est supérieur au temps total de tous les produits externes 180

6.8.2.3 Algorithme général utilisant la technique de recouvrement 182

6.9 Conclusion 183

A Modèle de calcul asynchrone **185**

A.1 Asynchronisme total 186

A.2 Application au système d'équations linéaires 188

A.3 Modèle de calcul asynchrone partiel 189

A.4 Application au calcul du vecteur stationnaire de probabilités 191

B Terminaison des algorithmes itératifs parallèles **193**

B.1 Convergence globale 193

B.2 Convergence locale 195

C Expérimentation des méthodes de puissances classique et réactualisée en séquentiel **197**

C.1 Plan d'expérience 197

C.2 Matrices creuses 198

C.3 Matrices du type presque décomposable (NCD) 198

C.4 Matrices triagonales par bloc 198

D Génération des matrices **203**

D.1 Matrices creuses 203

D.2 Matrices du type presque décomposable (NCD) 203

D.3 Matrices du type tridiagonales blocs 203

D.4 Matrice de transition de la marche aléatoire sur une grille 2D 204

E Algorithmes de placement **205**

E.1 Algorithme de Bokhari "échange de pairs" 205

E.2 Algorithme de récuit simulé 206

F Résultats expérimentaux sur le choix de l'ordre lexicographique **209**

G Recouvrement des communications par les calculs **221**

Introduction

L'évaluation de performance est primordiale pour la conception et le développement des systèmes informatiques. Parmi les techniques utilisées pour l'évaluation quantitative des systèmes, la modélisation est la méthode la plus économique. Elle consiste à représenter le fonctionnement d'un système par un modèle. Les critères de performance du système sont obtenus par résolution d'équations mathématiques caractérisant l'état stationnaire du modèle.

Les systèmes informatiques actuels sont de plus en plus complexe et leurs modèles sont de plus en plus gros (nombre d'états croit exponentiellement avec le nombre d'entité du système). Pour résoudre ces modèles sur un calculateur, nous sommes donc confronter à deux problèmes : le premier est la capacité mémoire du calculateur. Le second, est la rapidité avec laquelle on résout ces modèles sur le calculateur, sachant que le temps de calcul est aussi fonction de la taille de l'espace d'états.

L'avènement des calculateurs parallèles constitue une bonne opportunité pour résoudre les deux problèmes cités ci-dessus. Dans cette thèse, nous allons mettre en service le parallélisme pour résoudre les problèmes de l'évaluation de performance.

Ces dernières années, les chercheurs et concepteurs de systèmes informatiques se sont rendu à l'évidence que l'approche qui consiste à augmenter la puissance individuelle des calculateurs n'est plus réaliste, car en poursuivant ce but, on est confronté aux limites technologiques des composants électroniques. Devant ce fait, l'approche calcul parallèle semble une issue possible pour augmenter d'une façon substentielle la puissance de calcul. Cette nouvelle approche, consiste à utiliser plusieurs calculateur ensemble pour résoudre un même problème.

La naissance du calcul parallèle est suivie par une diversification des machines. Ces différentes machines sont plus ou moins adaptés à différents types de parallélisme. On distingue 3 grandes classes de calculateurs parallèles, ces calculateurs sont classés en fonction du rapport (donnée/contrôle) :

La première classe est notée (SISD), elle est caractérisé par une gestion centralisée des données et un contrôle centralisé. Cette classe regroupe les calculateurs monoprocesseurs.

La seconde classe (SIMD), les données sont distribuées et le contrôle est centralisé.

Dans la troisième classe (MIMD), les données sont distribuées et le contrôle aussi.

Selon la classe , on peut avoir des machines à mémoire partagée ou distribuée :

Dans les machines à mémoire partagée, la liaison entre mémoire-processeur est très forte et se fait par accés à une mémoire commune. L'accés à la mémoire par les processeurs est arbitré par un système de gestion de la mémoire commune. La faiblesse de ce type de machine vient du fait que plus le nombre de processeurs autour de la mémoire commune est grand et plus les performances de la machine se dégradent. Ce type de machine sont dite à faible parallélisme car la philosophie de ce type de machine est : un petit nombre de processeur mais très puissant (donc coûteux).

Dans les machines à mémoire distribuée , les contraintes de communication entre processeurs sont relachées . Chaque processeur dispose de sa mémoire locale. Les processeurs communiquent entre eux via un réseau d'interconnexion. Au contraire des machines à mémoire partagé, les machines à mémoire distribuée favorisent un parallélisme massif car l'idée de ce type de machine est d'avoir un grand nombre de processeurs simples (pas coûteux).

Pour résoudre notre problème de mémoire dans le cadre des modèles à grand espace d'états, les machines à mémoire distribuée sont bien appropriées car plus en rajoute des processeurs à la machine et plus on agrandit la mémoire globale de la machine. De plus, la puissance de calcul est multipliée par le nombre de processeurs de la machine.

En réalité, dans ce type de machine (MIMD), la communication entre les processeurs s'effectue moyennant un réseau de communication. Dans les machines actuelles, la vitesse du réseau de communication est lente par rapport à la vitesse des unités de calcul et si de plus, l'application parallèle qui s'exécute sur la machine est constituée d'un mouvement important de données, il en résulte une dégradation importante dûe au coût des communications.

Organisation de la thèse

Dans **le chapitre 1**, nous allons présenter la machine cible Meganode du type (MIMD) à mémoire distribuée. Nous évaluons les performances des différentes entitées qui composent cette machine, comme : la vitesse des communications et des unités de calcul ou encore le degré de parallélisme entre les communications et les calculs sur un noeud de la machine. Ces différentes mesures sont très importantes pour une bonne utilisation de la machine. Ces informations sont à prendre en compte pour l'implantation des algorithmes parallèles sur cette machine (choix du grain de parallélisme, ...etc).

Certains algorithmes parallèles génèrent un mouvement de données important. Dans **le chapitre 2** , nous discutons les principaux facteurs qui influent sur le coût des communications et nous proposons de nouvelles procédures de communications efficaces que nous utiliserons lorsqu'on abordera le chapitre de parallélisation des méthodes de résolution des modèles.

Dans **le chapitre 3**, nous revenons à l'évaluation des performances et nous présentons les différentes méthodes de résolution des modèles et leur parallélisation, tout en insistant sur les méthodes de résolution numérique parallèle qui constitue l'objet de cette thèse. Dans les méthodes de résolution numérique, on se restreint à étudier les méthodes itératives vu qu'elles s'appliquent bien quand l'espace d'états est grand.

Dans les méthodes itératives, l'opération de base est le produit matrice vecteur. **Le chapitre 4** est entièrement consacré à l'étude de parallélisation du produit matrice vecteur. On va présenter différentes implantations de ce problème.

Une fois le produit matrice parallèle réalisé, dans **le chapitre 5**, nous présentons différents schémas itératifs pour résoudre les modèles Markoviens. Nous proposons des schémas itératifs asynchrones qui s'adaptent bien au calcul parallèle. Ces schémas asynchrones ont pour objectifs d'éliminer les synchronisations entre processeurs et réduisent donc le coût des communications.

Enfin dans **le chapitre 6**, nous étudions la parallélisation de la résolution des modèles issus de la modélisation par réseaux d'automates stochastiques (RAS). La particularité de ce type de modèle est que leur matrice de transition n'est pas sous une forme compacte (classique). Cette matrice de transition est donnée par un descripteur qui est une formule. Mis à part que la modélisation par RAS permet un grand pouvoir d'expression, de plus l'avantage de cette méthode est de pouvoir effectuer l'opération de base des méthodes itératives qui est le produit vecteur matrice, sans construire explicitememt la matrice de transition. Il en résulte donc un gain en mémoire de stockage très important. Nous allons étudier en détails la parallélisation du produit vecteur par le descripteur d'un RAS. Nous utilisons le gain en mémoire de stockage pour traiter des modèles encore plus grand, de l'ordre du **million** d'états en un temps satisfaisant.

Nous terminons cette thèse par une conclusion et des perspectives de travaux futurs.

Chapitre 1

Présentation de la machine (MIMD) cible MEGANODE

1.1 Introduction

Les performances des diverses composantes d'une machine est une étude préliminaire essentielle pour comprendre la machine et l'exploiter au mieux. Le MEGANODE est une machine multiprocesseur à mémoire distribuée qui fonctionne selon le principe MIMD (Multiple Instruction Multiple Data). Les communications entre les Transputers de la machine se font par message. La connaissance des performances des communications et de la vitesse relative des unités de calcul est indispensable pour bien choisir le grain de parallélisme d'une application parallèle.

Dans le paragraphe 1.2, nous allons donner une description générale de la machine MEGANODE, tout en soulignant l'aspect particulier du réseau d'interconnexion de cette machine.

Nous verrons dans le chapitre suivant que le coût des communications dans un algorithme parallèle est très important. Dans le paragraphe 1.3, grâce à des mesures expérimentales, nous allons valider des modèles de communication sur cette machine.

En réalité, le coût d'un algorithme dépend fortement de l'équilibre entre la vitesse de calcul et la vitesse des communications. Dans le paragraphe 1.4, nous allons donner une idée du rapport de la vitesse calcul/communication.

Comme le temps de communication dans un algorithme parallèle constitue un surplus dans le temps total, il est toujours recommandé de réduire ce temps de communication.

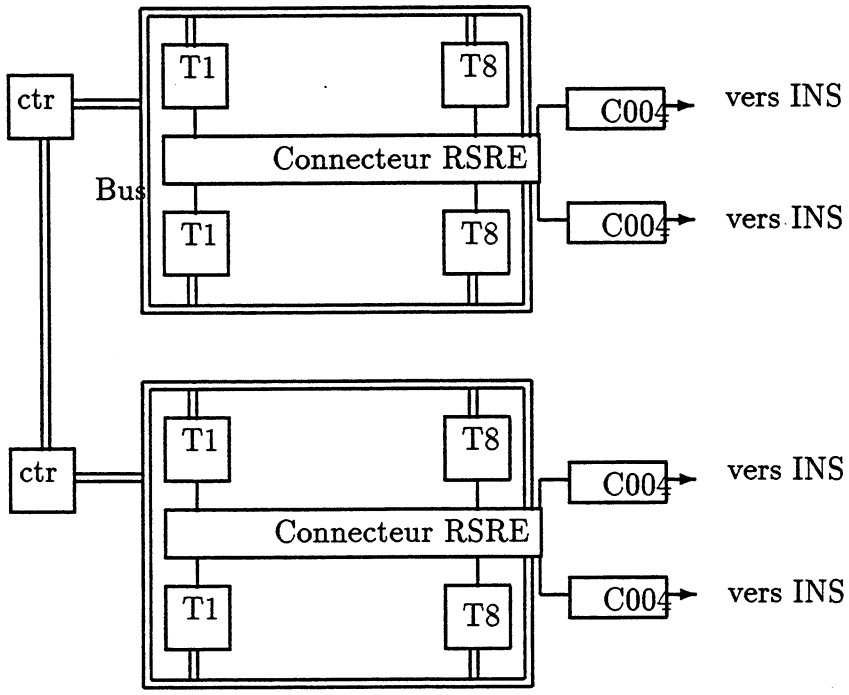


Figure 1.1: Tandem pour le MEGANODE

Une des idées pour réduire les communications est de les recouvrir par des calculs dans le cas où les calculs sont indépendants des communications et que la machine permet le recouvrement. Dans le paragraphe 1.5, nous allons mesurer le taux de recouvrement des communications par les calculs dans notre machine. Enfin en termine par une conclusion.

1.2 Description du MEGANODE à 128 Transputers

1.2.1 Vue générale

Le MEGANODE est une machine à base de Transputers (INMOS) regroupés en modules appelés tandem-nodes [21]. Chaque tandem-node peut contenir jusqu'à 32 Transputers de travail, 2 Transputers de contrôle et des Transputers serveurs (mémoire, disque) et un bus de contrôle (figure 1.1). Dans un tandem-node, les Transputers de travail sont connectés entre eux par un composant électronique appelé connecteur RSRE (crossbar 72 x 72) de Telmat.

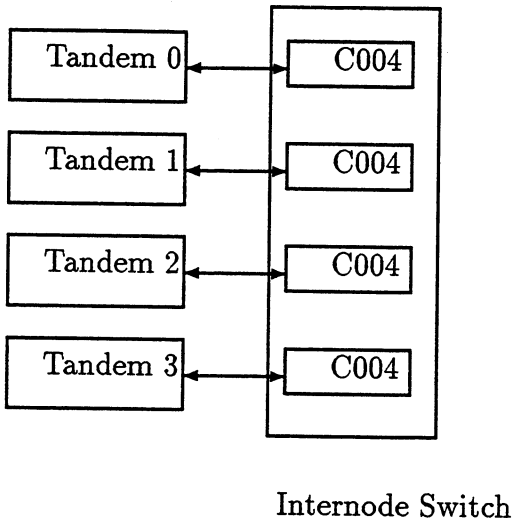


Figure 1.2: Liaisons via le connecteur internode.

Ces tandem-nodes sont reliés vers l'extérieur par des réseaux COO4 (INMOS) pour l'amplification et la resynchronisation des signaux. Le connecteur RSRE réalise n'importe quelle topologie de connexion entre les Transputers du tandem-node. Il constitue un premier niveau de connexion. Le MEGANODE à 128 Transputers est constitué de 4 tandem-nodes. Un deuxième niveau de connecteur est nécessaire pour relier des Transputers appartenant à des tandem-node différents. Ce deuxième niveau de connecteur appelé connecteur internode (INS) est réalisé par 4 connecteurs du type COO4 (INMOS) (32 x 32) (figure 1.2). Un bus de contrôle relie tous les Transputers de contrôle (ctr) du MEGANODE. Le bus de contrôle permet de véhiculer des requêtes de test (debbuging), des requêtes de configuration des connecteurs et d'autres informations comme par exemple les requêtes de synchronisation des Transputers. Le MEGANODE muni d'outils tels que les connecteurs et le bus de contrôle, est une machine à configuration variable, reconfigurable pseudo-dynamiquement ou dynamiquement. Le matériel proposé permet de changer la connectique du réseau de Transputers pendant l'exécution moyennant une synchronisation de tous les Transputers. A titre de comparaison les machines comme le FPS T40 ou l'IPSC INTEL sont des machines à topologie de connexion fixe (hypercube).

1.2.2 Réseau d'interconnexion pour Transputers

Le connecteur ou switch est un composant VLSI qui permet de connecter des Transputers entre eux. La communication via le connecteur permet des débits très importants.

Historiquement, la première utilisation de ce type de connecteur a été faite dans le réseau téléphonique. Puis grâce à ses bonnes performances il réapparait dans les systèmes informatiques tels que les architectures multiprocesseurs à mémoires partagées où il permet la connexion des processeurs aux mémoires. Actuellement et dans le projet P1085 pour la réalisation du *supernode,* le connecteur est chargé de connecter des Transputers entre eux constituant ainsi un *node.*

Définition

Un réseau d'interconnexion pour Transputers ou configurateur est constitué de deux parties :

- Une partie composant physique, constituée de 2 crossbars de 72 points de connexion et permettant de connecter n'importe quel lien d'un Transputer à n'importe quel autre lien d'un autre Transputer avec la contrainte de connexion qu'un point de connexion ne peut connecter qu'un seul lien.
- Une partie logiciel constituée d'un algorithme d'établissement des connexions et respectant les propriétés de réseau d'interconnexion reconfigurable.

Propriétés du réseau d'interconnexion

Le réseau d'interconnexion pour Transputers est prévu pour être reconfigurable c'est à dire qu'on peut changer la topologie du réseau à des instants précis appelés points de synchronisation. Le réseau d'interconnexion doit avoir les propriétés suivantes :

- L'interconnexion ne présente aucune restriction de topologie de réseau.
- La connexion de plus d'un lien entre deux Transputers est permise, elle peut être utile pour les communications rapides.
- Le réseau est accompagné d'un algorithme performant pour l'établissement des connexions dans un schéma de communication donné entre Transputers.
- Les Transputers dans le réseau peuvent ne pas être identiques. En effet, le Transputer de contrôle et le serveur mémoire ont pour fonctions la communication et le calcul simple. On peut leur dédier des Transputers T414. Par contre les Transputers de traitement demandent une grande puissance de calcul. les Transputers T800 sont les mieux adaptés puisqu'ils sont dotés en plus des Transputers T414 d'une unité de calcul flottant.

Graphe de connexions de Transputers

Soit un graphe de connexion de Transputers, les sommets du graphe représentent les Transputers, les arcs sont les liens de Transputers. On remarque qu'il est toujours possible de représenter un réseau de Transputers par un graphe complet en reliant les liens des Transputers qui ne sont pas connectés, à des Transputers fictifs. Etant donné que chaque Transputer à quatre liens, tous les sommets du graphe ont le même degré. Donc l'existence d'un cycle Eulérien est assurée (cycle qui passe par tous les arcs du graphe une seule fois).

En colorant alternativement avec deux couleurs les arcs du cycle Eulérien, on remarque l'existence de cycles de couleurs différentes, tel que l'union des sommets de chaque cycle recouvre tous les sommets du graphe. La configuration du réseau physique se base sur cette technique :

- Recherche du cycle Eulérien par des algorithmes performants en $O(n)$, avec n le nombre de Transputers dans le réseau.
- Séparation des cycles de couleurs différentes.
- Connexion des liens des Transputers de chaque cycle à un crossbar sachant que le réseau est formé de 2 crossbars.

1.3 Modèle de communication pour le MEGANODE

1.3.1 Communication à l'intérieur d'un même tandem-node

Le langage de programmation utilisé ici est le LOGICAL C. Il est muni d'une primitive de mesure du temps dont la résolution est une micro-seconde. On a mesuré les temps de communication entre 2 processus s'exécutant sur 2 Transputers différents d'un même tandem-node. Le tableau suivant montre l'évolution du temps de communication en fonction des tailles des données transférées [34].

Taille (octets)	Temps moyen (micro-seconde)	Ecart-type
8	13.87	0.37
16	22.83	0.38
24	31.86	0.38
32	40.86	0.39
1000	1129.83	0.38
2000	2254.81	0.38
3000	3379.85	0.37
4000	4504.86	0.36
8000	9004.83	0.40
9000	10129.85	0.37
10000	11254.82	0.38

Cette expérimentation a été effectué avec un échantillon de 1000 valeurs pour chaque taille de donnée considérée. L'expérience a été menée avec l'algorithme suivant :

Algorithme sur le Transputer émetteur

```
Début
   POUR chaque taille
        POUR i=1 à 1000
            1: recevoir(canal-entr,1 octet); /*synchronisation */
               t1=temps-courant();
               envoyer(canal-sort,message de taille octets);
               t2=temps-courant();
               temps-comm=t2-t1;
               tableau(i)=temps-comm;
        FINPOUR
        appel d'une procédure qui calcule la moyenne et la variance à partir du tableau;
   FINPOUR
Fin
```

Algorithme sur le Transputer récepteur

```
Début
   POUR chaque taille
      POUR i=1 à 1000
```

```
        2: envoyer(canal-sort,1 octet); /* synchronisation */
           t1=temps-courant();
           recevoir(canal-entr,message de taille octets);
           t2=temps-courant();
           temps-comm=t2-t1;
           tableau(i)=temps-comm;
      FINPOUR
      appel d'une procédure qui calcule la moyenne et la variance à partir du tableau;
   FINPOUR
Fin
```

Le rôle des instructions étiquetées 1 et 2 est de synchroniser les 2 Transputers afin d'éviter de mesurer un temps d'attente.

Etudions le tableau de résultats. On remarque un écart-type ne dépassant pas la micro-seconde qui est le plus petit temps mesurable. D'après ces mesures, on valide un modèle linéaire de communication,où les constantes β_{tandem},τ_{tandem} sont évalués par une méthode de régression linéaire (figure 1.3) :

$$tcom_{tandem} = \beta_{tandem} + \tau_{tandem}\ L$$

β_{tandem} = 4.85 micro-sec : Temps d'initialisation ou "start-up".
τ_{tandem} = 1.125 micro-sec/octet : Bande passante du lien .
L : Taille des données .

Si on compare avec la vitesse théorique du lien donné par le constructeur qui est de 10 Mbit/s, sachant qu'un octet de donnée est accompagné de 3 bits de contrôle, cela donnerait une vitesse effective de 1,1 micro-secondes par octet de données. Expérimentalement on a trouvé que la vitesse du lien est de 1.125 micro-sec/octet, ce qui est très proche de la valeur théorique.

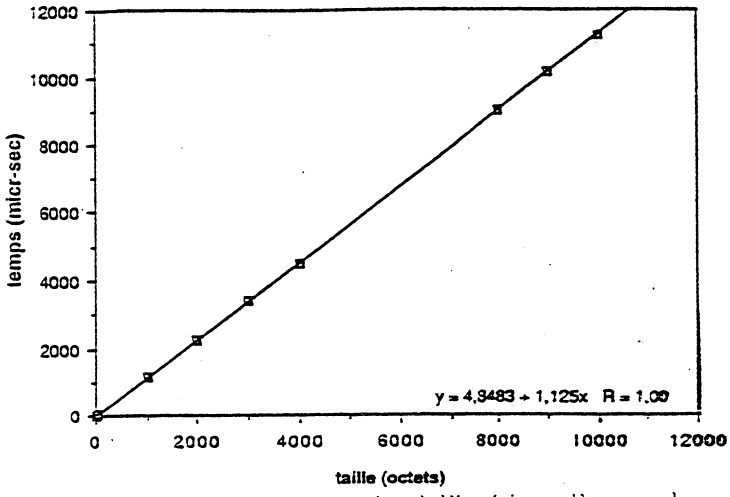


Figure 1.3: Temps de communication à l'intérieur d'un tandem-node.

1.3.2 Communication entre deux Transputers appartenant à deux tandem-nodes différents

Taille (octets)	Temps moyen (micro-seconde)	Ecart-type	Ecart-type/temps moyen
8	23.15	0.49	0.002
16	40.78	0.45	0.011
24	56.38	0.63	0.011
32	75.76	1.23	0.016
1000	2204.44	20.14	0.009
2000	4403.42	33.15	0.008
3000	6602.41	42.69	0.006
4000	8800.90	51.06	0.006
8000	17590.20	98.49	0.006
9000	19796.11	104.31	0.005
10000	21996.21	116.23	0.005

La même expérience à été effectuée mais entre 2 Transputers n'appartenant pas au même tandem-node. On remarque d'après ces résultats une variation autour de la moyenne qui est assez importante. Cela s'explique par le fait que pour transférer des données d'un Transputer d'un tandem-node vers un autre Transputer d'un autre tandem-node, les données passent par l'internode switch et elles traversent 3 connecteurs COO4 (figure 1.2). Une resynchronisation bit à bit est réalisée au niveau de chaque connecteur COO4. La variation du temps résulte de l'échantillonnage des bits à la resynchronisation (retard du bit par rapport au top d'échantillonnage). Ces retards s'accumulent dans le

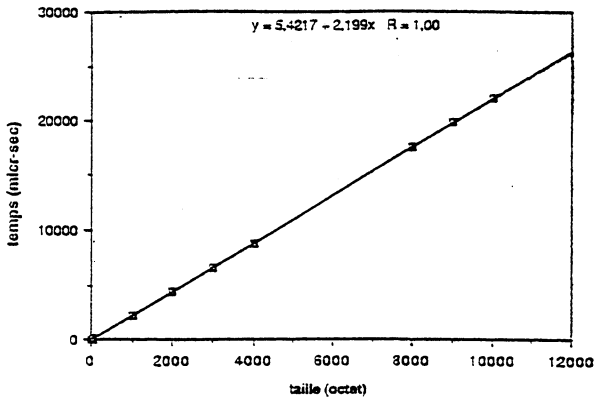


Figure 1.4: Temps de communication entre deux Transputers situés dans deux tandem-nodes différents.

pire des cas .

Au niveau de l'expérimentation, on voit bien que plus les tailles de donnée sont grandes plus l'écart-type est grand. Cependant, le rapport relatif écart-type/temps moyen diminue. Avec les résultats obtenus, on peut dériver un modèle pour le temps moyen de communication, les constantes étant toujours évaluées par régression linéaire (figure 1.4).

$$tcom_{2tandem} = \beta_{2tandem} + \tau_{2tandem}\ L$$

$\beta_{2tandem}$ =5.61 micro-sec : Temps d'initialisation ou "start-up".
$\tau_{2tandem}$ = 2.2 micro-sec/octet : Bande passante du lien .
L : Taille des données .

On remarque que la vitesse de transfert des données entre deux Transputers d'un même tandem-node est le double de celle de deux Transputers appartenant à deux tandem-nodes différents. Le temps d'initialisation est sensiblement le même.

1.3.3 Communication bi-directionnelle

Théoriquement, le temps d'une communication bi-directionnelle est le même que celui d'une communication uni-directionnelle. En pratique le temps de gestion du protocole de communication (accusés de réception des octets) n'est pas sans effet sur la performance du lien. L'expérience consiste à écrire deux processus concurrents par Transputer, un processus émetteur et un processus récepteur.

Algorithme sur le Transputer 1

```
Début
   POUR chaque taille
      POUR i=1 a 1000
         recevoir(canal-entr, 1 octet);
         t1=temps-courant();
         ProcPar(processus-envoi, processus-reception);
         t2=temps-courant();
         temps-comm=t2-t1 ;
         tableau(i)=temps-comm;
      FINPOUR
      appel d'une procédure qui calcule la moyenne et la variance à partir du tableau;
   FINPOUR
Fin
```

Algorithme sur le Transputer 2

```
Début
   POUR chaque taille
      POUR i=1 a 1000
         envoyer(canal-entr, 1 octet);
         t1=temps-courant();
         ProcPar(processus-envoi, processus-reception);
         t2=temps-courant();
         temps-comm=t2-t1;
         temps-comm=t2-t1 ;
         tableau(i)=temps-comm;
      FINPOUR
      appel d'une procédure qui calcule la moyenne et la variance à partir du tableau;
   FINPOUR
Fin
```

processus-envoi= envoyer(canal-sort, taille octets);
processus-réception=recevoir(canal-entr, taille octets);
La procédure ProcPar lance deux processus concurrents s'exécutant en temps partagé grâce à l'ordonnanceur microprogrammé du Transputer. Cette procédure est bloquante et l'instruction suivant l'appel de ProcPar ne s'exécute que si les deux processus lancés terminent. La mesure du temps d'une communication bi-directionnelle inclue le temps

de gestion de l'ordonnanceur qui est négligeable et le temps de commutation qui est de moins d'une micro-seconde [22].

D'après les mesures, on obtient un rapport de 1.47 pour le temps d'une communication bi-directionnelle sur le temps d'une communication unidirectionnelle intra-tandem. Le rapport théorique estimé par le constructeur est de 1.48 [22].

Des mesures similaires ont été effectuées dans le cas de 2 Transputers qui appartiennent à 2 tandems-nodes différents. On a la même bande passante que dans le cas mono-directionnel. Ce résultat surprenant s'explique par le recouvrement des messages est des acquittements. Le temps d'initialisation de la communication bi-directionnelle entre deux tandems différents est de l'ordre de 10 micro-secondes.

1.3.4 Communication sur les liens en parallèle

Grâce à l'ordonnanceur et au DMA sur chaque lien du Transputers T800, on peut effectuer des communications parallèles mono- et bi-directionnelles [55]. Les mesures résumés dans le tableau suivant sont des temps moyens (micro-secondes) relevé d'un échantillon de taille 10, avec une resynchronisation des Transputers après chaque mesure.

Taille (octets)	2 liens bi-directionnels	3 liens bi-directionnels	4 liens bi-directionnels	2 liens mo-directionnels	3 liens mo-directionnels	4 liens mo-directionnels
10	43	55	69	70	104	138
50	88	99	110	135	161	186
100	142	155	166	216	240	266
500	585	595	611	869	893	917
1000	1145	1149	1157	1694	1712	1735
5000	5609	5619	5625	8180	8312	8328
10000	11104	11178	11204	16545	16562	16579
20000	22253	22333	22383	32777	33063	33078
30000	33556	33643	33655	49545	49563	49579

La figure 1.5, montre que la bande passante des communications parallèles est indépendante du nombre de liens utilisés. En mode mono-directionnel cette bande passante est

$$\tau_{pm} = 1.125 \ \ (microsecondes/octets).$$

Dans le cas bi-directionnel :

$$\tau_{pb} = 1.64 \ \ (microsecondes/octets).$$

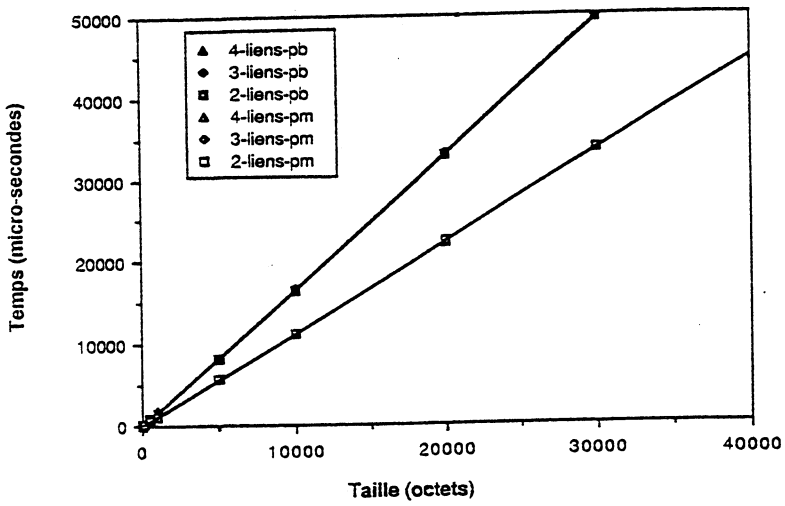


Figure 1.5: Bandes passantes des liens parallèles en mono et bi-directionnel.

Dans le tableau ci-dessous, on remarque que le temps d'initialisation des communications est croissant en fonction du nombre de liens utilisés. Le tableau suivant donne les valeurs des constantes d'initialisations des communications parallèles mono et bi-directionnels (micro-secondes).

β_{2pm}	β_{3pm}	β_{4pm}	β_{2pb}	β_{3pb}	β_{4pb}
30	32	37	35	76	100

1.4 Performance de l'unité de calcul du Transputer (T800/T414)

Le Transputer T800 est doté d'une unité de calcul flottant de plus que indépendante de l'unité de calcul habituelle (cas du Transputer T414). Le tableau expérimental suivant donne une comparaison entre les performances du Transputer T800 et T414 pour différents types d'opérations sur des variables en double précision.

Type d'opération	Temps moyen sur le T414 (μ s)	Temps moyen sur le T800 (μ s)
vr=vop_1+vop_2	52.6	1.1
$vr[\ i\ ] = vop_1[\ j\ ] + vop_2[\ k\ ]$	54.4	1.8
vr=vop_1*vop_2	69.4	1.8
$vr[\ i\ ] = vop_1[\ j] * vop_2[\ k\ ]$	70.8	2.57

Pour avoir une idée de la vitesse communication/calcul, nous allons estimer ce rapport pour une communication d'un réel double précision sur le temps de multiplication entre deux réels double précision :

$R_{com/cal}$ varie entre 3.5 et 7 selon le cas de communication dans le même tandem node ou dans deux tandems node différents.

1.5 Taux de recouvrement des communications par du calcul

Dans certains algorithmes parallèles contenant du calcul et des communications, si on peut extraire une partie du calcul qui est indépendante des communications à une étape donnée de l'algorithme, il est intéressant d'exécuter en parallèle cette partie du calcul avec les communications. On dit qu'on va recouvrir les communications par le calcul. Dans ce paragraphe nous allons étudier le taux de recouvrement des communications par du calcul pour la machine MEGANODE.

Notation :

- On note t_{calc} : le temps d'exécution de la partie calcul.
- t_{comm} : le temps d'exécution de la partie communication.
- $t_{calc+comm}$: le temps résultant de la partie calcul suivit séquentiellement par la partie communication.
- $t_{calc//comm}$: le temps résultant de l'exécution en parallèle de la partie calcul et la partie communication (primitive ProcPar).

On définit le taux de recouvrement des communications par le calcul comme :

$$Recouv = \frac{t_{calc+comm} - t_{calc//comm}}{t_{comm}} * 100$$

Dans les expériences qui sont résumés dans les tableaux suivants, nous avons mesuré le taux de recouvrement des communications par du calcul. Ces mesures représentent des moyennes sur un échantillon de taille 10 avec resynchronisation des Transputers émetteur et récepteur. Dans les expériences qu'on présente dans ce paragraphe, on considère deux types de communications :

- Communication sur 1 lien en mono-directionnel.
- Communication sur 4 liens bi-directionnel.

D'autres expériences de mesure du taux de recouvrement pour d'autre types de communication sont présentées dans l'annexe G.

Recouvrement du calcul par une communication sur 1 lien mono-directionnel

$t_{calc+comm}$ (s)	0.165184	0.251584	0.424384	0.597480	0.769984	0.942784
$t_{calc//comm}$ (s)	0.089161	0.175561	0.348288	0.521088	0.693824	0.866368
t_{comm} (s)	0.078784	0.078784	0.078784	0.078784	0.078784	0.078784
$Rapport\ \frac{t_{calc}}{t_{comm}}$	1	2	4	6	8	10
$Recouv(\%)$	96.5	96.5	96.6	96.6	96.7	96.9

D'après le tableau précédent, on remarque un bon recouvrement des communication par le calcul de l'ordre de 96%.

Recouvrement du calcul par une communication sur 4 liens bi-directionnel

$t_{calc+comm}$ (s)	0.229632	0.344832	0.572320	0.805632	1.036096	1.266432
$t_{calc//comm}$ (s)	0.135872	0.250944	0.480384	0.710784	0.941120	1.171580
t_{comm} (s)	0.114432	0.114432	0.114432	0.114432	0.114432	0.114432
$Rapport\ \frac{t_{calc}}{t_{comm}}$	1	2	4	6	8	10
$Recouv(\%)$	82	82	83	83	83	83

D'après le tableau précédent, on remarque un taux recouvrement des communication par le calcul de l'ordre de 83%. Cette dégradation de performance du recouvrement et principalement dûe à la bande passante du bus d'accés à la mémoire. L'accés au bus est partagé entre l'unité de calcul et les 4 DMA sur les liens de communications. Il est clair alors que cette dégradation et d'autant plus forte que le nombre de liens utilisés en

parallèle est grand. Dans les expériences ci-dessus, cette dégradation est de l'ordre de 4% dans le cas d'un seul lien. Par contre dans le cas où on utilise 4 liens en parallèle, la dégradation est de l'ordre de 20%.

1.6 Conclusion

Les communications entre les Transputers situés sur des tandem-nodes différents sont plus coûteuses (deux fois plus) que les communications dans un même tandem-node. La machine MEGANODE présente donc une "dissymétrie" au niveau des communications. Cela nuit à l'homogénéité de la machine et rend son utilisation efficace plus difficile. Pour ce qui du rapport de vitesse communication/calcul, on note un rapport qui varie entre 3.5 et 7. Dans la machine MEGANODE, le recouvrement des communications par les calculs présente dans le pire des cas une dégradation de l'ordre de 20%. Nous rediscutons ces différents paramètre de cette machine plus loin pour l'implantation d'algorithmes parallèles.

Chapitre 2

Communications dans les machines à mémoires distribuées

2.1 Introduction

Nous avons vu dans le chapitre précédent que les machines à mémoire distribuée (MIMD) ne possèdent pas de mémoire commune. Pour échanger des données entre les processeurs de la machine, un réseau de communication qui relie tous les processeurs entre eux est utilisé. On dit que les processeurs de la machine communiquent par envoi de messages. Dans beaucoup d'algorithmes parallèles (distribué), le temps passé en communications entre les processeurs constitue une partie non négligeable du temps total pour résoudre le problème considéré. On peut quantifier cette perte par le rapport

$$R_{perte} = \frac{T_{total} - T_{cal}}{T_{cal}}$$

où T_{total} représente le temps total nécessaire pour résoudre le problème et T_{cal} représente le temps de calcul. Le temps de calcul T_{cal} , correspond au temps de l'algorithme si les communications sont instantanées.

Dans tous ce qui suit, on suppose que le système parallèle distribué consiste en un réseau de processeurs connectés par des liens de communication. Où chaque processeur utilise sa mémoire locale pour stocker des données. L'échange de données entre processeurs se fait par messages, qui sont véhiculés sur les liens de communication du réseau.

Dans le paragraphe 2.2, nous allons évoquer les principaux facteurs qui influent sur le coût des communications qui sont :

- Le mode de communication : qui définie le protocole de transfert des messages entre processeurs.
- La topologie du réseau de communication : qui représente l'éloignement des processeurs.
- Le type de communication intensive : qui caractérise le mouvement des données

Après avoir fixé un mode de communication et une topologie de connexion, nous allons nous intéresser aux différentes procédures de communications globale ou intensive. Dans le cadre de cette thèse, nous étudions de près 2 types de communications intensives, fréquement utilisées dans l'algorithmique parallèle matricielle et qui sont à la base des méthodes de parallélisation qu'on va présenter dans les chapitres 4 et 5. Ces deux procédures de communication intensives sont l'échange total et l'échange total avec accumulation.

Le paragraphe 2.3 présente deux procédures optimales qui résolvent respectivement l'échange total et l'échange total avec accumulation sur la topologie anneau.

Dans les paragraphes 2.4 et 2.5, nous proposons deux procédures optimales qui résolvent ces problèmes de communication sur une topologie tore.

Dans le paragraphe 2.6, nous proposons une variante de la procédure d'échange total pour résoudre un type de communication particulier qu'on note échange partiel personnalisé. On retrouve ce type de communication dans les algorithmes que nous présentons dans le chapitre 6. Enfin, on termine par une conclusion.

2.2 Principaux facteurs qui influent sur le coût de communication

Parmi les principaux facteurs qui influent sur le coût des communications, on recense : Le mode de communication, la topologie et le type de communication intensive. Dans ce paragraphe, nous allons étudier ces 3 facteurs.

2.2.1 Mode de communication

Le mode de communication, consiste en un protocole d'échange des messages entre différents noeuds du réseau. Nous allons passer en revue 2 modes de communication point-à-point (processeur à processeur), et nous allons montrer la différence de coût induit par chacun de ces modes de communication.

2.2.1.1 Communication par commutation de message

Dans ce mode de communication bien connu sur le nom "store-and-forward", l'unité d'information est le message. Un message émis d'une source traverse un certain nombre de processeurs intermédiaires, où le message est stocké , puis retransmis au noeud suivant, jusqu'à sa destination finale. On retrouve ce mode de communication dans les machines à base de Transputers, les séries FPS etc.

2.2.1.2 Communication par commutation de circuit

Dans ce mode de communication, un circuit est établi entre l'émetteur et le récepteur, les noeuds intermédiaires n'ont plus à stocker le message (chemin réservé d'avance), il y a donc moins de perte de temps.

Il existe trois manières pour réaliser ce mode de communication :

1. **La commutation de circuit :**

   L'émetteur commence par envoyer la tête du message qui contient des informations pour atteindre le destinataire. Cette tête de message progresse dans le réseau et établit des connexions sur chaque noeud intermédiaire, créant ainsi un chemin réservé entre la source et la destination. Par analogie avec le réseau téléphonique, sur chaque noeud intermédiaire, on positionne un commutateur . Une fois le circuit établit, l'émetteur envoie le corps du message sur ce circuit.

2. **Le routage par "wormhole" :**

   Le principe est le même que dans la commutation de circuit, mais le corps du message suit la tête dès le début de l'établissement du chemin. Si arrivé sur un noeud intermédiaire, les liens de communication ne sont pas libres (blocage), le message est stocké le long du circuit en attendant sa progression [12].

3. **Le routage "virtual cut through" :**

Il consiste en une variation du routage par *wormhole*. Dans le cas de blocage (besoin de ressource physique), le message n'est pas stocké le long du chemin, mais sur le dernier noeud. Ce procédé libère plus rapidement les ressources physiques (liens de communication). Néanmoins, on ajoute une hypothèse supplémentaire qui est : les mémoires tampons sur chaque noeud sont de taille infinie [25] .

2.2.1.3 Comparaison du coût de communication les différents modes

Nous considérons les deux modes de communication : la commutation de message et le *wormhole*. Dans les deux cas, on suppose que le message est composé de n_p paquets (unité d'information) , que le transfert d'un paquet sur un lien coûte L_p unités de temps et le chemin entre l'émetteur et le destinataire est de k liens avec $k \geq 1$. On suppose négligeable le temps d'initialisation des communications et que le réseau n'est pas chargé.

- Dans le mode commutation de message, le message entier de taille n_p paquets doit traverser les k liens de communication, soit un coût de communication de $(n_p.L_p)$ par lien. Le temps de propagation total est

$$Comm_{ComMes} = n_p \; L_p \; k.$$

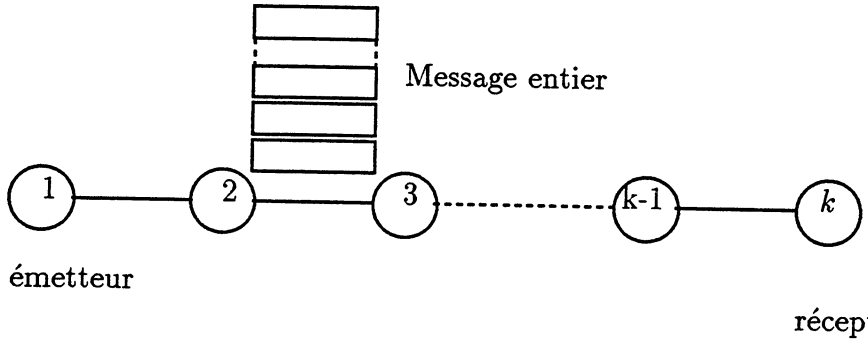
- Dans le mode *wormhole*, le message est vu en paquets que le processeur émetteur envoie l'un après l'autre. Ainsi, le premier paquet arrive à destination après $(1.L_p.k)$ unité de temps. Le second arrive L_p unité de temps après l'arrivé du premier et ainsi de suite pour les paquets restants du message. Le coût du transfert du message est

$$Comm_{WormH} = k \; L_p \; + \; (n_p - 1)L_p$$

On remarque dans l'expression $Comm_{WormH}$ que si n_p est grand (cas des grands message), le premier terme devient négligeable devant le deuxième. Donc avec le mode *wormhole* la notion de distance entre les noeuds est masquée. Ce qui n'est pas le cas dans le mode commutation de message. Le gain du mode *wormhole* est dû au pipeline des paquets qui consiste à envoyer des paquets l'un après l'autre dès que le lien est libéré (figure 2.1).

Dans les communications point à point, quand le traffic (charge du réseau) est faible, il est certain que le mode de commutation de circuit est meilleur que le mode commutation de message. Quand il s'agit de mouvement de données global, le réseau est fortement chargé. Dans ce cas, on est moins sûr des performances du mode de commutation de

Mode commutation de message :



Utilisation du pipeline dans le mode commutation de circuit :

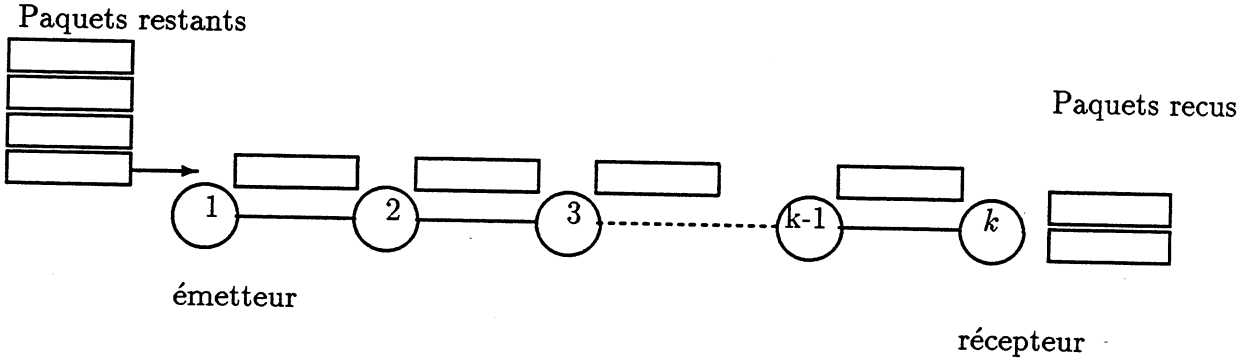


Figure 2.1: Schéma des deux modes de communication

circuit, car les délais d'obtention des circuits et la gestion des files d'attente deviennent importants.

Dans ce qui suit, le mode de communication qu'on adopte est le mode commutation de message, car on s'intéresse au types de communication dites communications intensives (mouvement de données important sur les noeuds) et que notre machine fonctionne sur ce mode de communication. Dans le mode commutation de message, on suppose que les liens de communication sont bi-directionnelles et que chaque processeur peut communiquer sur ses liens simultanément.

2.2.2 Topologie du réseau de communication

On peut citer l'influence de la topologie du réseau d'interconnexion comme un facteur important dans le coût des communications. Un réseau d'interconnexion est caractérisé par :

- Le diamètre δ du réseau : qui représente la distance maximum entre deux noeuds quelconques. Cette distance est donnée par le nombre minimum de liens qu'il faut traverser pour aller d'un noeud à autre. Pour montrer l'importance du diamètre de la topologie : le coût du transfert d'un paquet (message unitaire) entre deux noeuds quelconques est de l'ordre de $O(\delta)$.

- La connectivité m du réseau d'interconnexion : mesure le nombre m de chemins distincts connectant deux noeuds quelconques. Cette notion de connectivité est utilisée pour augmenter la bande passante. En effet, on peut imaginer qu'au lieu d'envoyer un message sur un seul chemin, on le découpe en m parties et on envoie simultanément chaque partie sur un chemin. On peut faire donc le transfert du message en un temps de l'ordre de $O(\frac{\delta}{m})$.

2.2.3 Type de communications intensive

Le coût de communication d'un algorithme parallèle, dépend de l'application. En effet le mouvement de données dans un produit scalaire, un produit matrice vecteur ou un produit matrice par matrice n'est pas le même. Dans ce qui suit nous allons essayer de répertorier et classifier les types de communications intensives les plus utilisées dans l'algorithmique parallèle.

2.2.3.1 Diffusion et multidiffusion (échange total)

Dans la diffusion, un processeur donné envoie le même message à tous les autres. L'échange total consiste en une diffusion simultané de chaque noeuds. Pour résoudre la diffusion, il suffit de trouver un arbre de recouvrement dont la racine est le noeud source du message. De même, pour l'échange total, il faut trouver des arbres de recouvrement issus de chaque noeuds. La difficulté dans ce cas est que plusieurs liens peuvent appartenir à plusieurs arbres. D'où un risque de contention de liens. Dans le paragraphe 2.4 nous résolvons ce problème dans le cas d'une topologie torique.

La diffusion apparait souvent dans les algorithmes matriciels telles que la réduction de Gauss, Jordan etc. On retrouve l'échange total dans les méthodes itératives tel que Gauss-Seidel, Jacobi ...

2.2.3.2 Accumulation et échange total avec accumulation

Dans l'accumulation, chaque noeud contient un message qu'il faut envoyer à un noeud donné avec la particularité suivante :

Aux noeuds intermédiaires, les messages sont combinés avant d'être transmis et le coût d'envoi du message combiné est le même que celui du message initial. Ce problème d'accumulation, apparait quand on veut former sur un processeur donné, la somme de chacune des données qui se trouvent sur les autres processeurs. Typiquement cela correspond à un produit scalaire distribué. En effet, la combinaison (somme) des données sur un noeud intermédiaire n'augmente pas la taille de la donnée.

L'échange total accumulé consiste à des accumulations simultanés sur chaque noeud. On retrouve ce type de communication dans une certaine version parallèle du produit matrice vecteur (voir le paragraphe 4.2.2).

2.2.3.3 Distribution, rassemblement et échange total personnalisé

- **La distribution :** correspond à une diffusion personnalisé. Un noeud donné envoie à chaque noeud un message différent.
- **Le rassemblement :** est le problème inverse de la distribution. Un noeud donné collecte toutes les données des autres noeuds.
- **L'échange total personnalisé :** consiste à ce que, chaque noeud envoie à chaque noeud un message personnalisé. Dans ce cas, chaque noeud envoie des messages

différents pour chaque noeud. Par contre dans un échange total, chaque noeud envoie le même message à tous les autres. Ce type de communication apparait dans un calcul matriciel tel que la transposition.

2.2.3.4 Dualité entre les types de communication intensives

On définie la dualité dans ce contexte comme suit :

Définition 1 *Un type de communication A est dual (ou semblable) d'un type de communication B si et seulement si on connait une méthode qui résout A, alors elle résout B aussi et avec le même coût.*

Parmi les types de communication cités précédemment, on note que :

- **La diffusion** et **l'accumulation** sont semblables :

  Le même arbre de recouvrement qui résout la diffusion (envoi de la racine en direction des feuilles) résout l'accumulation (envoi des feuilles en direction de la racine et accumulations aux noeuds intermédiaires). Ils sont donc semblables. (figure 2.2).

- **L'échange total** et **l'échange total avec accumulation** :

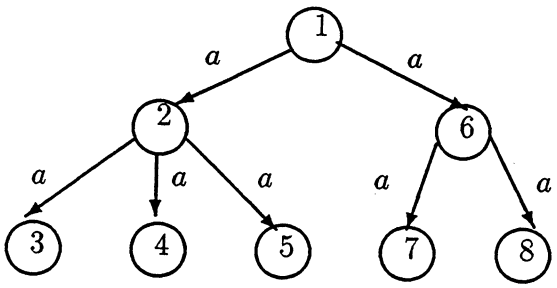
  De la même façon que dans le point précédent, ces deux problèmes sont semblables. Plus loin nous allons développer une procédure réalisant l'échange total sur un tore et on montre, comment modifier cette procédure pour effectuer l'échange total avec accumulation.

- **La distribution** et **le rassemblement** :

  De même ces deux types de communications sont semblables. Le rassemblent correspond exactememt à l'opération inverse de la distribution.

Dans ce qui suit, on ne s'intéresse qu'aux communications intensives du type échange total et son dual, car presque tous les algorithmes parallèles qu'on présente dans cette thèse, utilisent ces types de communications. Nous commencons par étudier ces deux types de communication sur la topologie anneau, puis nous proposons une nouvelle procédure pour la résolution de ces deux types de communications sur le tore.

DIFFUSION DU NOEUD 1



ACCUMULATION SUR LE NOEUD 1

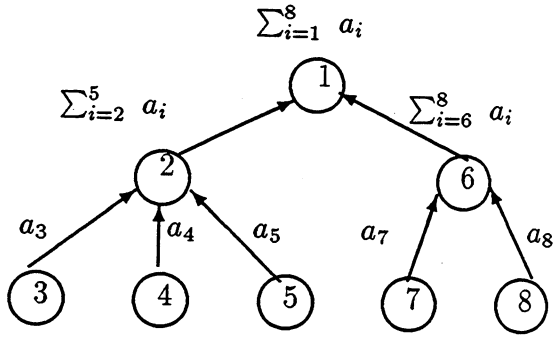


Figure 2.2: Dualité entre la diffusion et l'accumulation

Dans tous ce qui suit, nous adoptons le modèle de communication suivant :

Modèle de communication

◇ Mode de communication : commutation de message

◇ Communication paralléle sur les 4 liens en bi-directionnel

◇ Coût de communication :

$$T_{com} = \tau L$$

τ : bande passante

L : la taille des données

avec τ qui représente la bande passante et L la taille de la donnée à transférer. On suppose que L est suffisamment grand pour négliger les temps d'initialisation (voir le chapitre 1).

2.3 Echange total et échange total avec accumulation sur l'anneau

2.3.1 Echange total sur l'anneau

Soit p le nombre de processeurs de l'anneau. L'algorithme qui effectue l'échange total est constitué de $\lfloor \frac{p}{2} \rfloor$ étapes. A la première étape chaque processeur p_j envoie son message simultanément à ses voisins p_{j+1} et p_{j-1} qui sont situés respectivement à sa droite et gauche (les numéros de processeurs sont modulo p). A chaque étape i , $i \in [2..\lfloor \frac{p}{2} \rfloor]$, chaque processeur propage le message qu'il a reçu à l'étape $i-1$ soit à sa droite, s'il l'avait reçu à gauche , sinon il l'envoie à sa gauche (figure 2.3). Cet algorithme se résume comme suit :

Algorithme sur chaque processeur p_j

Début
  Envoyer son message et recevoir sur les deux liens (droite et gauche)
  Pour $i = 2$ à $\lfloor \frac{p}{2} \rfloor$
    En parallèle
      Envoyer le message reçu à l'étape $(i-1)$ par le lien droit sur le lien gauche
      Envoyer le message reçu à l'étape $(i-1)$ par le lien gauche sur le lien droit

Recevoir un message sur le lien droit
Recevoir un message sur le lien gauche
FinPour
Fin

On retrouve cet algorithme dans plusieurs travaux, on cite [43] [15]. Le temps de propagation de cet algorithme est :

$$Comm_{EchAnn} = \lfloor \frac{p-1}{2} \rfloor L\,\tau.$$

Ce qui correspond à un temps de propagation optimal sur l'anneau.

2.3.2 Echange total avec accumulation sur l'anneau

On rappelle le principe de cette procédure de communication :

Sur chaque processeur, on doit cumuler les messages personnalisés de chaque processeur. Chaque processeur contient p messages personnalisés de taille L. Cette procédure de communication est décrite dans [43] [18] et se résume comme suit :

Algorithme sur chaque processeur p_j

Début
Pour $i = \lfloor \frac{p-1}{2} \rfloor$ à 1
En parallèle
Envoyer à droite le message destiné au processeur (p_{j+i})
Envoyer à gauche le message destiné au processeur (p_{j-i})
Recevoir à droite et accumuler avec le message destiné au processeur (p_{j+1-i})
Recevoir à gauche et accumuler avec le message destiné au processeur (p_{j-1+i})
FinPour
Fin

Le coût de cet algorithme (le temps de propagation) :

$$Comm_{EchAccuAnn} = \lfloor \frac{p-1}{2} \rfloor L\,\tau.$$

C'est le même temps que l'échange total sur l'anneau (problème dual).

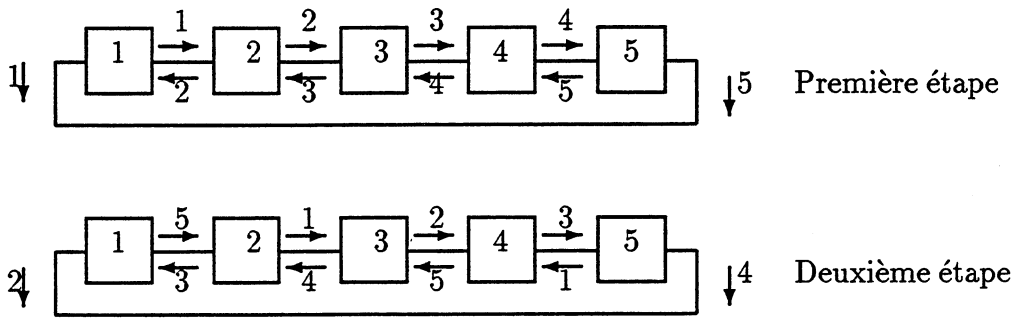


Figure 2.3: Echange total sur l'anneau

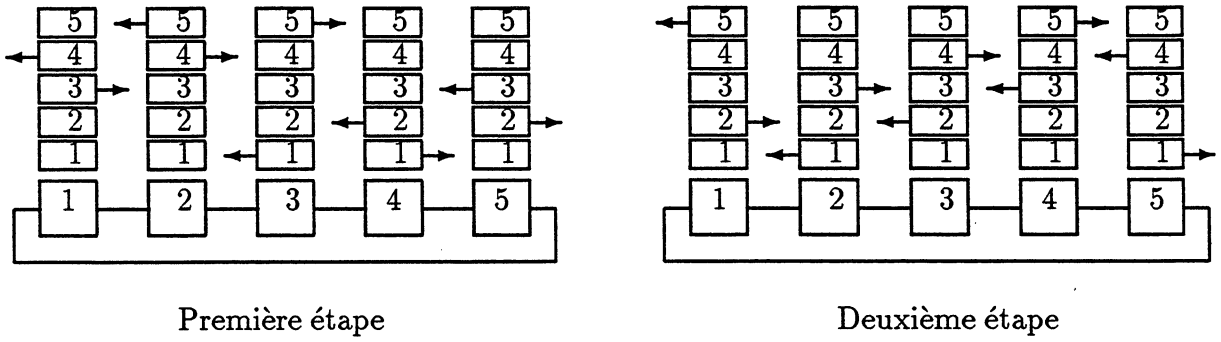


Figure 2.4: Echange total avec accumulation sur l'anneau

2.4 Echange total sur une topologie torique

Nous allons présenter une procédure *d'échange total* optimale pour la topologie tore. Cette procédure est basée sur la technique des arbres de recouvrement et utilise un espace mémoire tampon réduit.

Le tore est une topologie régulière qui s'adapte bien pour résoudre des problèmes numériques matricielles. Hormis, quelques topologies adéquates, le coût des communications d'un algorithme distribué est un facteur crucial qu'il faut minimiser.

Dans ce qui suit, nous nous intéressons à une des procédure de communication parmi les plus utilisées, connue sous le nom d'échange total ou encore multidiffusion (*all-to-all*), où chaque noeud reçoit des données de chaque noeud de la topologie. Le besoin de diffusion apparait dans les problèmes numériques où un vecteur est le résultat d'un calcul parallèle et les composantes du vecteur sont distribuées sur le réseau. Pour réutiliser le vecteur sur chaque processeur pour la prochaine étape, il faut effectuer une procédure d'échange total. Ce genre de communication intensive arrive souvent dans le calcul neuronal, le calcul itératif matricielle, etc.

Dans [20] [32], le modèle de communication est le suivant : les liens de communication sont mono-directionnels, chaque noeud ne peut communiquer que sur un lien à un instant donné, le coût de transferts est une unité de temps (indépendant de la taille du message). Dans [43], [18], on étudie l'échange total sous l'hypothèse que le coût de communication est une fonction linéaire par rapport à la taille du message ($\beta + \tau\ L$), dans [43], le mode de communication est mono-directionnel, par contre dans [18], on considère le cas des communications bi-directionnels. Dans [49], on considère l'échange total dans les réseaux d'ordinateurs (ARPANET,...).

Les solutions proposées pour résoudre l'échange total demandent un nombre de tampons mémoire qui croit avec la taille de la topologie. Comme on s'intéresse à échanger des messages de grande taille, l'espace mémoire tampon devient un facteur important, surtous quand la capacité mémoire d'un noeud est relativement petite (1 Mega-octets sur le Transputer). Dans la procédure que nous allons proposer, le nombre de tampons mémoire est réduit à zéro ou au maximum à une unité. Nous limitons notre étude aux tores carrés de dimension deux. Dans la procédure d'échange total, on suppose que la taille des données est la même sur chaque processeur et on l'appelle: unité de donnée.

On rappelle que le modèle de communication considéré dans cette thèse est le suivant:

Modèle de communication

◇ Mode de communication : commutation de message

◇ Mode de communication : commutation de message

◇ Communication paralléle sur les 4 liens en bi-directionnel

◇ Coût de communication :

$$T_{com} = \tau L$$

τ : bande passante

L : la taille des données

avec τ qui représente la bande passante et L la taille de la donnée à transférer. On suppose que L est suffisamment grand pour négliger les temps d'initialisation (voir le chapitre 1). Ce modèle de communication est bien adapté aux machines à base de Transputers et des machines comme la connection machine car le temps d'initialisation est petit devant le temps de transfert du message (voir le chapitre 1).

Dans ce qui suit, on définit une étape de communication comme l'envoi et la réception simultané d'une unité de donnée sur les 4 liens de chaque noeud. Dans la procédure d'échange total, chaque noeud doit recevoir p-1 données si p est le nombre de noeuds du tore. Comme chaque noeud à 4 liens, à chaque étape, il peut recevoir au plus 4 nouvelles données. Le temps de propagation minimum pour accomplir la procédure d'échange total est de $\frac{p-1}{4}\tau L$, correspondant à $\frac{p-1}{4}$ étape de communication.

Dans ce qui suit, nous allons présenter une procédure d'échange total qui atteint cette borne inférieure du temps de propagation. Dans le paragraphe 2.4.1, nous considérons le cas des tores carrés ($\sqrt{p} \times \sqrt{p}$) avec $\sqrt{p}$ impair.

Dans le paragraphe 2.4.2, nous allons voir comment effectuer l'échange total quand les tores sont ($\sqrt{p} \times \sqrt{p}$) avec $\sqrt{p}$ pair . Une implémentation est présenté dans le paragraphe 2.4.3 .

2.4.1 Procédure d'échange total sur un tore ($\sqrt{p} \times \sqrt{p}$) avec $\sqrt{p}$ impair

2.4.1.1 Notation

Chaque noeud du tore et repéré par un double indice (i, j) qui sont ses positions dans le tore (i=numéro de ligne, j=numéro de colonne). Dans ce qui suit les calculs d'indices sont faits implicitement modulo $\sqrt{p}$. Les quatre liens du noeud sont nommés $Nord$, Est, $West$, Sud ($NEWS$). Le noeud (i, j) communique avec le noeud $(i + 1, j)$ par son lien

Nord, avec le noeud $(i, j+1)$ par son lien *Est* , avec le noeud $(i-1, j)$ par son lien *Sud* et avec le noeud $(i, j-1)$ par son lien *West* . Il faut noter que, si (i, j) est connecté au noeud $(i+1, j)$ par son lien *Nord* , alors le noeud $(i+1, j)$ est connecté au noeud (i, j) par son lien *Sud* .

2.4.1.2 Position du problème

Un arbre de recouvrement de racine le noeud (i, j) d'un graphe connexe G, est un graphe partiel de G noté $A_{(i,j)}$ qui est connexe est acyclique.

Diffuser des données sur un arbre de recouvrement de racine donnée, consiste à propager les données de la racines vers tous les autres noeuds sur cet arbre de recouvrement. Dans notre approche, l'échange total correspond à une multidiffusion depuis chaque noeud. Pour résoudre cette multidiffusion, nous avons besoin de spécifier un arbre de recouvrement issu de chaque noeud du tore. Nous considérons l'envoi parallèle de chaque noeud du tore (i, j) sur l'arbre de recouvrement $A_{(i,j)}$ dont il est racine. La difficulté réside dans le fait que les liens peuvent appartenir à plusieurs arbres de recouvrement dans une même étape de communication. Pour résoudre ce problème, nous construisons des arbres de recouvrement $A_{(i,j)}$ qui soient arcs-disjoint dans le temps i.e, à chaque étape, un lien quelconque du tore n'appartient qu'à un et un seul arbre de recouvrement $A_{(i,j)}$. Nous allons montrer plus loin quelles contraintes topologique sur la structure des arbres de recouvrement sont suffisantes pour obtenir la propriété d'arbre arcs-disjoints dans le temps (voir la définition d'arbre F_{NB}).

Le borne inférieure du temps de propagation de la procédure échange total est $\frac{p-1}{4}\tau L$ correspondant à $\frac{p-1}{4}$ étapes de communication. L'idée de base est de construire des arbres de recouvrement de profondeur $\frac{p-1}{4}$. Les arbres de recouvrement contiennent $p-1$ noeuds plus la racine, il suffit donc de considérer ces arbres de recouvrement comme des arbres constitués de 4 branches linéaires de profondeur $\frac{p-1}{4}$. Nous notons par F cette famille d'arbres.

Pour accomplir l'échange total en $\frac{p-1}{4}$ étapes, nous avons besoin d'arbre de recouvrement $A_{(i,j)} \in F$ issué de chaque noeud du tore (i, j). La propriété d'arcs-disjoints dans le temps des arbres s'énonce comme une conséquence de la propriété suivante :

Définition 2 *On définie une famille d'arbres de recouvrement $F_{NB} \subset F$, noté arbres NEWS-Equilibré. Un arbre est NEWS-Equilibré si et seulement si :*

Pour chaque $d \in [1, \frac{p-1}{4}]$, les 4 noeuds situés à la profondeur d de l'arbre de recouvrement sont connectés aux noeuds de la profondeur $d+1$ (de la même branche) par des

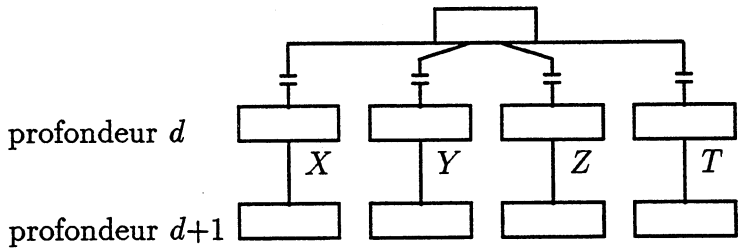


Figure 2.5: Connexions à la profondeur d d'un arbre F_{NB} , où (X, Y, Z, T) sont distincts dans L.

liens distincts de l'ensemble Lien=(Nord, Est, West, Sud) (figure 2.5).

2.4.1.3 Exemple d'arbre de recouvrement F_{NB} du tore ($\sqrt{p} \times \sqrt{p}$) avec $\sqrt{p}$ impair

Pour prouver l'existence des arbres de recouvrement F_{NB}, nous allons exhiber un exemple de ces arbres. Pour construire un arbre de recouvrement F_{NB} de racine le noeud (i, j), nous allons procéder en trois phases. La première phase consiste à partitionner les noeuds du tore en 4 régions distincts plus le noeud racine. Dans la deuxième phase, on couvre l'une des 4 régions par une chaine qui va représenter une des branches de l'arbre de recouvrement. Dans la dernière phase, on déduit les autres branches de l'arbre de recouvrement, par rotation de la première branche.

- **Première phase : partitionnement des noeuds du tore en 4 régions**

  On pose $m=\lfloor\frac{\sqrt{p}}{2}\rfloor$. On Définie les quatre régions ($NordEst$, $SudEst$, $SudWest$, $NordWest$) du tore autour du noeud (i,j) comme suit (figure 2.6) :

  Si $1 \leq x \leq m$ et $0 \leq y \leq m \Longrightarrow$ le noeud $(i+x,j+y)$ appartient à la région $NordEst$

  Si $-m \leq x \leq 0$ et $1 \leq y \leq m \Longrightarrow$ le noeud $(i+x,j+y)$ appartient à la région $SudEst$

  Si $-m \leq x \leq -1$ et $-m \leq y \leq 0 \Longrightarrow$ le noeud $(i+x,j+y)$ appartient à la région $SudWest$

  Si $0 \leq x \leq m$ et $-m \leq y \leq -1 \Longrightarrow$ le noeud $(i+x,j+y)$ appartient à la région $NordWest$

  Il est clair que ces régions couvre tous les noeuds du tore sauf le noeud (i, j).

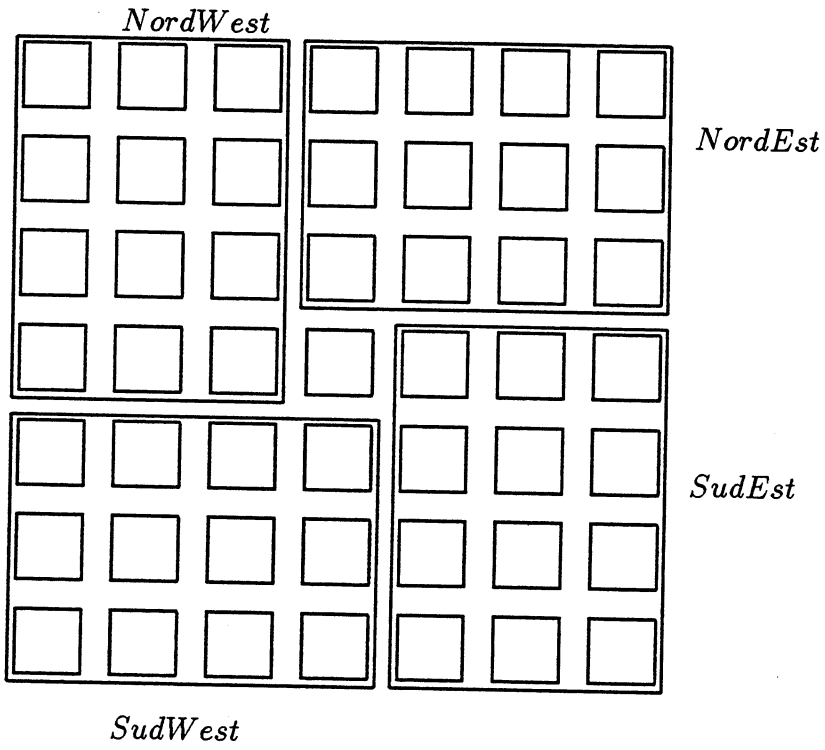


Figure 2.6: Partitionnement des noeuds du tore

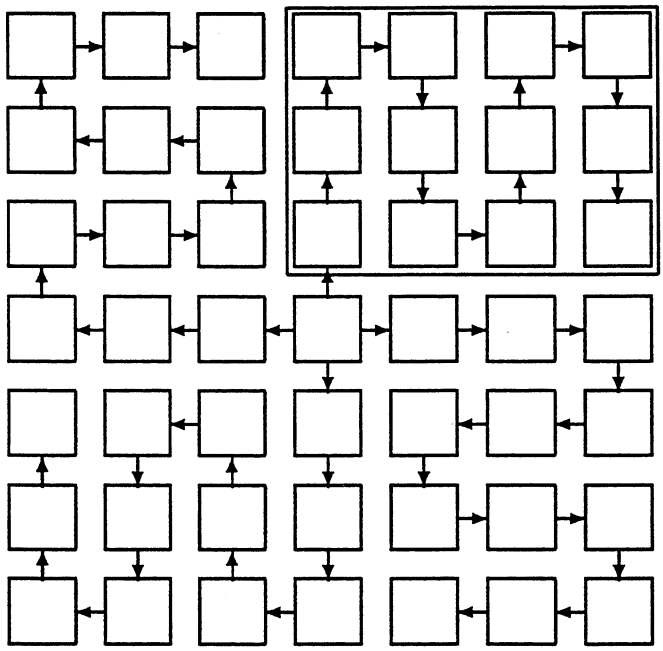


Figure 2.7: Un arbre de recouvrement F_{NB} du tore

- **deuxième phase : construction de la branche** $NordEst$ (figure 2.7)

  La branche $NordEst$ est formée par la concaténation de $m+1$ sous-branches de m noeuds, les sous-branches sont numérotées de 0 à m (figure 2.7). On considère que cette branche est orientée dans le sens : du noeud racine de l'arbre de recouvrement vers le noeud terminal.

  La sous-branche numéroté k noté SB_k contient les noeuds $(i+x,\ j+k)$, avec $1 \leq x \leq m$. Les noeuds de la sous-branche SB_k avec k impair sont connectés par une connexion *Nord-Sud*. Les noeuds de la sous-branche SB_k avec k pair sont connectés par une connexion *Sud-Nord*. Les $(m+1)$ sous-branches sont connectées comme suit :

  Dans le cas où k est pair, le noeud $(i+m, j+k)$ de la sous-branche SB_k est connecté au noeud $(i+m, j+k+1)$ de la sous-branches SB_{k+1} par une connexion *Est-West*. Pour k impair, le noeud $(i+1, j+k)$ de la sous-branche SB_k est connecté au noeud $(i+1,\ j+k+1)$ de la sous-branche SB_{k+1} par une connexion *Est-West* .

- **La troisième phase : déduction des autres branches de l'arbre par rotation** (figure 2.7)

  Soit T_{ore} l'ensemble des noeuds du tore, $Lien$=$\{Nord,\ Est,\ Sud,\ West\}$ l'ensemble des directions, et on définie trois applications :

  - Soit $R_{(i,j)}$ la rotation de $\frac{\pi}{2}$ autour du noeud $(i,\ j)$.

    $$R\ :\ T_{ore} \longrightarrow T_{ore},$$

    qui associée au noeud x_1=$(i+i_1, j+j_1)$, le noeud $R_{(i,j)}(x_1)$=$(i-j_1, j+i_1)$.
  - Soit $\Gamma_{(i,j)}$ le graphe homomorphique associé avec $R_{(i,j)}$.
  - Finalement, soit l'application S ,

    $$S\ :\ Lien \longrightarrow Lien,$$

    qui fait la rotation des points cardinaux de $\frac{\pi}{2}$: $S(Nord)$=Est, $S(Est)$=Sud, $S(Sud)$=$West$, $S(West)$=$Nord$.

  Il faut remarquer que les noeuds des régions ($NordEst$, $SudEst$, $SudWest$, $NordWest$) ont leurs images par $R_{(i,j)}$ respectivement dans les région ($SudEst$, $SudWest$, $NordWest$, $NordEst$). On construit à la main la branche $NordEst$. La branche $SudEst$ est déduite de la branche $NordEst$ par $\Gamma_{(i,j)}$. Similairement, la branche $SudWest$ est déduite de la branche $SudEst$ par $\Gamma_{(i,j)}$, et ainsi de suite pour la branche $NordWest$. Le noeud racine $(i,\ j)$ est connecté respectivement par ses liens ($Nord$, Est, Sud, $West$) aux branches ($NordEst$, $SudEst$, $SudWest$, $NordWest$).

L'arbre de recouvrement résultant est NEWS-Equilibré : à chaque profondeur d de l'arbre, si le noeud de la branche $NordEst$ est connecté à un noeud à une profondeur donnée par le lien l_1, alors à la même profondeur dans les branches $SudEst$, $SudWest$, $NordWest$, les liens $S(l_1)$, $S \circ S(l_1)$, $S \circ S \circ S(l_1)$ sont utilisés et sont différentes.

2.4.1.4 Procédure d'échange total optimal pour le tore ($\sqrt{p} \times \sqrt{p}$) avec $\sqrt{p}$ impair

Revenons au cas général, on étudié la procédure utilisant les arbres de recouvrement F_{NB} . On suppose que chaque arbre de recouvrement issu de chaque noeud du tore vérifie la propriété suivante :

Propriété 1 : *Etant donné un arbre de recouvrement $A_{(i_0,j_0)} \in F_{NB}$ de racine le noeud (i_0, j_0), pour tous les autres noeuds (i,j), l'arbre de recouvrement $A_{(i,j)}$ est déduit de l'arbre $A_{(i_0,j_0)}$ par translation d'amplitude (i-i_0, j-j_0).*

On rappelle que notre approche pour effectuer la procédure d'échange total est basée sur une diffusion simultané sur les arbres de recouvrement issus de chaque noeud du tore. La procédure d'échange total comporte une étape particulière d'initialisation, où chaque noeud du tore envoie sa donnée à ses quatre voisins. Etant donnée la topologie du tore, chaque noeud va recevoir sur chacun de ses quatre liens des données différentes. La procédure d'échange total est vu comme la succession d'étape de communication : si un noeud donné (i,j) appartient à l'arbre $A_{(i,j)}$ à la profondeur d et est connecté au noeud (i',j') (à la profondeur $d+1$), alors la $(d+1)ieme$ étape de communication de ce noeud inclue la communication avec le noeud (i',j') .

Dans la preuve qui va suivre, on note les branches de l'arbre F_{NB} par les noms des liens avec lesquelles elles sont connectées au noeud racine. On parle alors de branches $Nord$, Sud, Est, $West$ dans chacun des arbres.

Dans ce qui suit, nous allons prouver que notre procédure d'échange total est optimale, dans le sens où elle requière exactement un temps de $\frac{p-1}{4}\tau L$.

Lemme 1 *A la première étape de l'échange total (à la profondeur 1 dans l'arbre), chaque noeud du tore appartient à 4 arbres de recouvrement distincts dans des branches différentes*

Preuve :

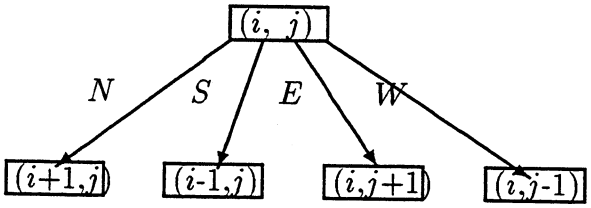


Figure 2.8: Arbre de recouvrement à la profondeur 1

Etant donné un noeud (i, j), on cherche tous les arbres qui contiennent ce noeud à la profondeur 1. Par construction, les arbres de racines respectivement $(i\text{-}1, j)$, $(i+1, j)$, $(i, j\text{-}1)$, $(i, j+1)$ contiennent le noeud (i, j) respectivement sur leur branche *Nord*, *Sud*, *Est* et *West* (figure 2.8). Ceci termine la preuve du lemme 1.

Lemme 2 *à chaque profondeur* $d \in [\ 1, \frac{p-1}{4}\ ]$ *des arbres de recouvrement* F_{NB} *, chaque noeud appartient à quatre arbres de recouvrement distincts dans des branches différentes.*

Preuve :

Nous allons procéder par récurrence. Soit (H) l'hypothèse de recurrence :

A chaque profondeur de tous les arbres de recouvrement, chaque noeud appartient à exactement 4 arbres de recouvrement distincts dans des branches différentes.

(H) est vraie à la profondeur $d = 1$ (lemme 1). On suppose que (H) est vraie à la profondeur d et on montre qu'elle est aussi vraie à la profondeur d+1, $d + 1 \leq \frac{p-1}{4}$.

(H) est vraie à la profondeur d, implique que chaque noeud X du tore appartient à 4 arbres de recouvrement distincts dans des branches différentes. Ceci est vrai en particulier pour les quatre noeuds voisins de X, qu'on note VX_1, VX_2 , VX_3 , VX_4 .

A chaque profondeur d le noeud VX_1 appartient à quatre branches différentes dans quatre arbres de recouvrement distincts. Donc il atteint le noeud X sur l'une des quatre branches à la profondeur $d+1$ (structure NEWS-Equilibré). Avec le même raisonnement appliqué au arbres qui contiennent les noeuds VX_j avec j=2, 3, 4 à la profondeur d, on déduit qu'à la profondeur $d+1$, le noeud X appartient à quatre arbres de recouvrement. Ces arbres sont disjoints sinon ils ne sont pas recouvrants (deux branches différentes d'un même arbres atteignent le même noeud X) .

Nous allons montrer que ces branches sont différentes : les arbres issus de chaque noeud sont déduits par translation, donc toutes les branches *Nord* de chaque arbre

connecte le noeud à la profondeur d au noeud à la profondeur $d+1$ par un lien de même direction ($Nord$, Est, $West$, Sud). De plus, les arbres NEWS-Equilibré, impliquent que pour chaque profondeur d, la direction utilisé pour connecter, de la profondeur d à la profondeur $d+1$ caractérise la branche. Le noeud X va donc être atteint par ses quatre voisins dans quatre arbres distincts dans quatre branches différentes.

Théorème 1 ([1]) *Etant donné la famille d'arbre de recouvrement F_{NB}, avec des arbres de recouvrement issus de chaque noeud vérifiant la propriété 1 , la multidiffusion sur ce type d'arbres accomplie la procédure d'échange total avec un temps de propagation optimale $(\frac{p-1}{4})\ \tau L$.*

Preuve :

Nous avons montré qu'à chaque profondeur $d \in [\ 1,\ \frac{p-1}{4}\ ]$ des arbres de recouvrement F_{NB}, chaque noeud appartient à quatre arbres de recouvrement distincts dans des branches différentes . Donc à chaque étape de communication , chaque noeud du tore reçoit sur ses quatre liens des données différentes, car à chaque profondeur des arbres de recouvrement , des branches distinctes utilisent des liens différents (voir la définition des arbres F_{NB}). Il n'y a donc pas de contention sur les liens à aucune étape de communication et les arbres sont arcs-disjoints dans le temps. Comme les arbres de diffusion sont de profondeur $\frac{p-1}{4}$, la procédure d'échange total atteint la borne inférieure du temps de propagation $(\frac{p-1}{4})\ \tau L$.

Remarque :

Cette procédure d'échange total n'utilise aucun tampon mémoire de stockage supplémentaire . A chaque étape de communication, chaque noeud propage simplement les données qu'il vient de recevoir.

2.4.2 Procédure d'échange total sur un tore ($\sqrt{p} \times \sqrt{p}$) avec $\sqrt{p}$ pair

De la même une façon que dans 2.4.1.2, on construit un arbre de recouvrement qui est structuré en quatre branches linéaires de racine (i, j). Comme $p-1$ n'est pas divisible par 4, le nombre de noeud par branche n'est pas le même. L'arbre de recouvrement est constitué de trois branches linéaires de longueur $\frac{p}{4}$, la quatrième branche contient $\frac{p}{4}$-1 noeuds. Pour ce type d'arbre, on n'est pas capable de trouver des exemples d'arbre arcs-disjoints dans le temps (on ne peut non plus prouver qu'il est impossible d'en trouver). L'idée pour éviter cette difficulté est de couvrir un ensemble maximal de noeuds $E(i, j)$ du tore par un arbre de recouvrement de racine le noeud (i, j) avec des branches qui contiennent le même nombre de noeud et qui vérifié la propriété d'arbre arcs-disjoints

dans le temps: on note cette famille d'arbre par F^E_{NB}. On remarque que la profondeur d'un tel arbre, ne peut dépasser $\frac{p}{4}$-1. On note T_{ore}-$E(i, j)$ l'ensemble des noeuds du tore non atteints par cet arbre, la cardinalité de ce dernier est au maximum 3. Dans le cas général, nous n'avons pas de critère de choix pour identifier les noeuds de l'ensemble T_{ore}-$E(i, j)$. Dans ce qui suit, nous proposons un exemple d'arbre de recouvrement où le choix des noeuds de T_{ore}-$E(i, j)$ est simple.

2.4.2.1 Exemple d'arbre de recouvrement F^E_{NB} pour un tore ($\sqrt{p} \times \sqrt{p}$) avec $\sqrt{p}$ pair

On présente deux types d'arbres de recouvrement correspondant aux deux cas : $m=\lfloor \frac{\sqrt{p}}{2} \rfloor$ est impair ou pair. On montre que dans les deux cas, la procédure d'échange total qui utilise ces types d'arbres termine en un temps optimal et utilise un seul tampon mémoire de stockage sur chaque noeud.

- m impair : (figure 2.9)

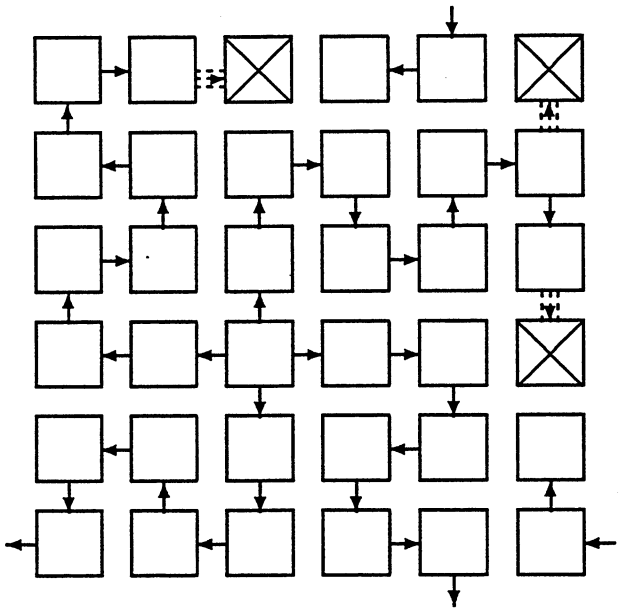
  Comme dans le paragraphe 2.4.1.3, on construit la branche $NordEst$ de l'arbre puis en déduit les autres branches par rotation. La branche $NordEst$ est la concaténation de $(m + 1)$ sous-branches de $(m - 1)$ noeuds. Les sous-branches sont numérotées de 0 à m. La sous-branche SB_k contient les noeuds de numéros $(i + x, j + k)$ avec $1 \leq x \leq m - 1$. Les noeuds de la sous-branche SB_k avec k impair, sont connectés par des connexions $Nord$-Sud. Les noeuds de la sous-branche SB_k avec k pair, sont connectés par des connexions Sud-$Nord$. Les $(m + 1)$ sous-branches sont connectées comme suit :

  Le noeud $(i + m - 1, j + k)$ de la branche SB_k avec k pair est connecté au noeud $(i+m-1, j+k+1)$ de la branche SB_{k+1} par une connexion $West$-Est . Le noeud $(i+1, j+k)$ de la branche SB_k avec k impair est connecté au noeud $(i+1, j+k+1)$ de la sous-branche SB_{k+1} par une connexion $West$-Est .

- m pair : (figure 2.10)

  En premier lieu on construit la branche $NordEst$ de l'arbre :

  La branche $NordEst$ qui est une concaténation de $(m - 1)$ sous-branches de (m) noeuds, plus une sous-branche contenant $(m - 1)$ noeuds. La sous-branche SB_k contient les noeuds de numéros $(i + k, j + x)$ avec $0 \leq x \leq m - 1$ pour k=0, m-2 et $1 \leq x \leq m - 1$ si k=m-1. Les noeuds de la sous-branches SB_k avec k impair sont connectés par des connexions Est-$West$. Les noeuds de la sous-branche SB_k avec k pair sont connectés par des connexions $West$-Est. La sous-branches $(m-1)$ sont connectées comme suit :

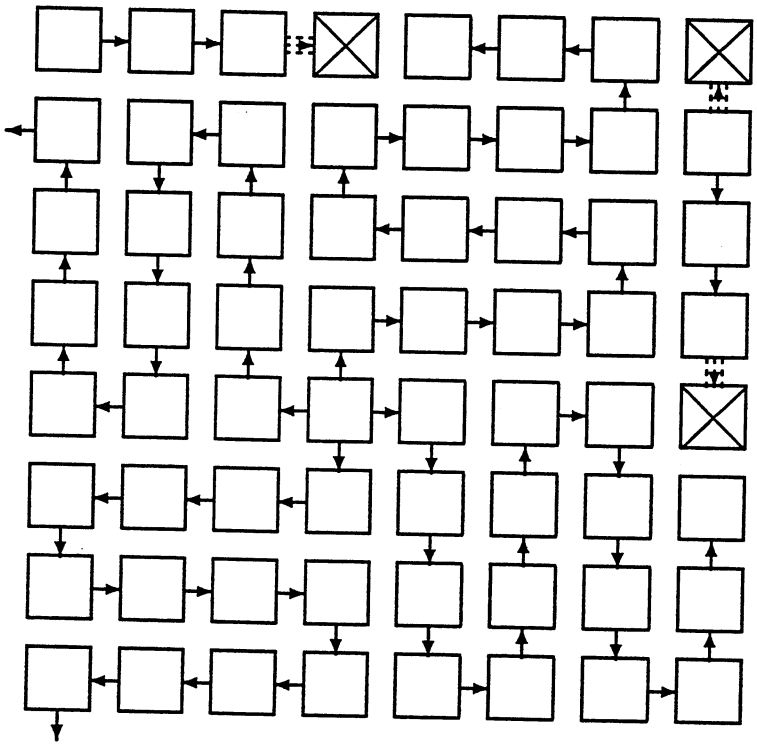


L'ensemble T_{ore}

L'ensemble $T_{ore} - E$

Communication à la dernière étape

Figure 2.9: Arbre F_{NB}^{E} pour le tore 6×6



L'ensemble $T_{ore} - E$

L'ensemble T_{ore}

Communication à la dernière étape

Figure 2.10: Arbre F_{NB}^{E} pour le tore 8×8

Le noeud $(i+k,\ j+m-1)$ de la sous-branche SB_k avec k pair est connecté au noeud $(i+k+1,\ j+m-1)$ de la sous-branche SB_{k+1} par une connexion *Sud-Nord*. Le noeud $(i+k,\ j)$ de la sous-branche SB_k avec k impair est connecté au noeud $(i+k+1,\ j)$ de la sous-branche SB_{k+1} par des connexions *Sud-Nord*. Finalement, les branches *SudEst*, *SudWest*, *NordWest* sont déduites par rotation de la branche *NordEst* .

On remarque que ces arbres sont F^E_{NB} (branches déduites par rotation). Sachant que la profondeur de ces arbres est de $\frac{p}{4}$-1, l'ensemble de noeud $E(i,\ j)$ est maximal car il est impossible de trouver des arbres de recouvrement qui soient structurés en quatre branches linéaires ayant le même nombre de noeuds dans chaque branche et qui dépassent une profondeur de $\frac{p}{4}$-1.

Dans ces deux cas, l'ensemble T_{ore}-$E(i,\ j)$ contient 3 noeuds x_m= $(i+m,\ j)$, y_m= $(i,\ j+m)$ et xy_{2m}= $(i+m,\ j+m)$. Les noeuds x_m, y_m sont situés à une distance de m de la racine $(i,\ j)$ et le noeud xy_{2m} est à distance $2m$ (diamètre) du noeud racine.

2.4.2.2 Procédure d'échange total sur un tore ($\sqrt{p} \times \sqrt{p}$) avec $\sqrt{p}$ pair

Dans ce qui suit , les arbres de diffusion F^E_{NB} vérifiant la propriété 1 sont utilisés comme dans les paragraphes précédents pour effectuer un échange total incomplet. Nous allons montrer comment compléter ce procédé, afin de réaliser la procédure d'échange total et atteindre les noeuds T_{ore}-$E(i,\ j)$, pour chaque noeud (i,j).

La procédure de diffusion sur l'arbre $A^E_{(i,j)} \in F^E_{NB}$ est la suivante :

La racine diffuse la donnée le long de l'arbre $A^E_{(i,j)}$. A l'étape $\frac{p}{4}$-1, les 3 noeuds restants doivent recevoir la donnée émise par la racine $(i,\ j)$. Du fait de la topologie tore, chacun des ces trois noeuds sont connectés à au moins un noeud de l'ensemble $E(i,\ j)$. Notons respectivement $E'(i,\ j)$=$(vxy_{2m},\ vx_m,\ vy_m)$ les noeuds voisins des noeuds $(xy_{2m},\ x_m,\ y_m)$ de $E(i,\ j)$. Il suffit de faire une étape de communication additionnelle où les 3 noeuds de $E'(i,\ j)$ envoient leurs données à leurs voisins de T_{ore}-$E(i,\ j)$.

Un bon choix de l'ensemble $E'(i,\ j)$ est tel qu'ils utilisent des liens de directions différentes dans $NEWS$ (pour éviter la contention de liens à la dernière étape). Ce choix donne une procédure d'échange optimale qui termine en $\frac{p}{4}$ étapes. De plus, l'ensemble des noeuds $E'(i,\ j)$ doivent stocker la donnée émise par la racine $(i,\ j)$ à une étape donnée, dans la perspective de la communiquer aux noeuds de T_{ore}-$E(i,\ j)$ à la dernière étape. Si les noeuds $E'(i,\ j)$ sont des noeuds terminaux dans les branches (dans $A^E_{(i,j)}$), ils doivent juste transmettre la donnée. Dans le cas général, on ne peut pas prouver

l'existence d'un tel ensemble $E'(i, j)$ avec cette bonne propriété. Mais comme dans l'exemple développé dans le paragraphe 2.4.2.1, on va proposer un choix de l'ensemble $E'(i, j)$.

On considère les deux cas : m pair et impair.

- m **impair** (figure 2.9) :
  1. $vxy_{2m} = (i + m - 1, j + m)$ de la branche $NordEst$, envoie la donnée sur son lien $Nord$.
  2. $vx_m = (i + m, j - 1)$ de la branche $NordWest$, envoie la donnée sur son lien Est .
  3. $vy_m = (i + 1, j + m)$ de la branche $NordEst$, envoie la donnée sur son lien Sud .
- m **pair** (figure 2.10) :
  1. $vxy_{2m} = (i + m - 1, j + m)$ de la branche $NordWest$, envoie la donnée sur son lien $Nord$.
  2. $vx_m = (i + m, j - 1)$ de la branche $SudWest$, envoie la donnée sur son lien Est .
  3. $vy_m = (i + 1, j + m)$ de la branche $NordWest$, envoie la donnée sur son lien Sud.

Dans les deux cas, on insiste sur le fait que les noeuds vx_m, vy_m sont situés à la profondeur $\frac{p}{4}$-1 de l'arbre $A^E_{(i,j)}$ (noeud terminaux) et le noeud vxy_{2m} est situé à la profondeur $\frac{p}{4} - m + 1$. A l'étape $\frac{p}{4}$-m+1, le noeud vxy_{2m} de chaque arbre $A^E_{(i,j)}$ stocke la donnée émise par la racine (i, j). Plus précisément, à l'étape $\frac{p}{4}$-m+1, chaque noeuds du tore stocke la donnée qu'il vient de recevoir sur son lien $West$ si m est impair, lien Est si m est pair et là renvoie sur son lien $Nord$ à dernière étape. Les noeuds vx_m, vy_m sont situés à la profondeur $\frac{p}{4}$-1 (correspondant à la dernière étape de la procédure). Ils n'ont pas besoin de stocker la donnée, il suffit qu'ils transmettent respectivement sur leur lien (Est, Sud) la donnée qu'il viennent de recevoir sur leur lien ($West$, $Nord$), pour chaque $A^E_{(i,j)}$. Dans les deux cas, la procédure d'échange total termine en $\frac{p}{4}$ étape de communication et utilisent qu'un seul tampon mémoire de stockage intermédiaire sur le noeud vxy_{2m}.

Remarque :

On vient de montrer que pour éviter le besoin de tampon mémoire de stockage, on doit choisir l'ensemble des noeuds $E'(i,j)$ situés à la profondeur $\frac{p}{4}$-1 de l'arbre $A^E_{(i,j)}$. La question est la suivante :

Est-il possible de trouver des arbres de recouvrement F_{NB}^{E} tel que : les noeuds de l'ensemble $E'(i, j)$ sont tous situés à la profondeur $\frac{p}{4}$-1. Dans le cas particulier de tore 4×4, la réponse est oui. Dans le cas général, on n'a pas de réponse.

2.4.3 Implémentation de l'échange total

Dans ce paragraphe on présent un algorithme pour réaliser un échange total sur un tore $(\sqrt{p} \times \sqrt{p})$ avec $\sqrt{p}$ impair, en utilisant les types d'arbres construits dans le paragraphe 2.4.1.3 comme exemple. On rappelle que m est la longueur des branches SB_k.

On Définit la procédure *Communication*(l_1, l_2, l_3, l_4 : dans L) comme :

En parallèle :
- Envoi de la donnée reçu à l'étape précédente par le lien *Nord* sur le lien l_1.
- Envoi de la donnée reçu à l'étape précédente par le lien *Est* sur le lien l_2.
- Envoi de la donnée reçu à l'étape précédente par le lien *West* sur le lien l_3.
- Envoi de la donnée reçu à l'étape précédente par le lien *Sud* sur le lien l_4.

Et la procédure *PremiereEtape*() comme :

En parallèle :
- Envoie de la donnée locale sur les 4 liens.
- Reçoie des données sur les 4 liens.

Algorithme à chaque noeud :

Procédure échange total
{
PremiereEtape().
Communication(*Sud*, *West*, *Est*, *Nord*);
k=0; /* *le numéro de la sous-branche* */
m= $\frac{\sqrt{p}-1}{2}$; /* *le nombre de sous-branche* */

While($k \leq m$)

{

for (i=1; $i \leq m$-2; i++) /* *Construction de la sous-branche* SB_k, k *pair* */

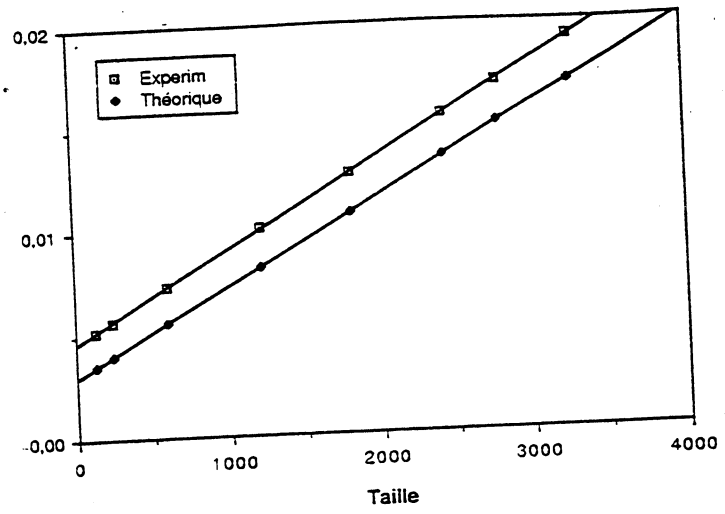
Communication(*Sud*, *West*, *Est*, *Nord*);

if ($k < m$)

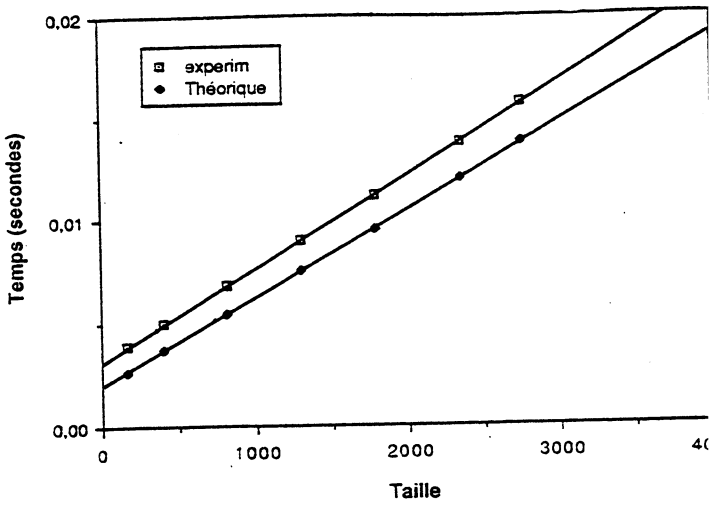
```
{
 k=k + 1;
 Communication(West, Nord, Sud, Est); /* Changement de direction */
 Communication(West, Nord, Sud, Est); /* Changement de direction */
 for(i=1; i ≤ m-2; i++) /* Construction de la sous-branche SB_k, k impair */
 Communication(Sud, West, Est, Nord);
 if (k < m)
 {
 Communication(Est, Sud, Nord, West); /* Changement de direction */
 Communication(Est, Sud, Nord, West); /* Changement de direction */
 }
}
k = k + 1;
}
}
```

L'implémentation de notre échange total sur un réseau de Transputers résulte en une procédure étape par étape qui requière des synchronisation à chaque étape. La synchronisation est naturellement fourni par le mode synchrone de communication (rendez-vous) du Transputer. Dans ce cas le réseau MIMD est utilisé comme un réseau SPMD. Si le réseau n'exécute que l'échange total, il se comporte alors comme dans un mode presque synchrone. Les calcul locaux ne sont pas coûteux, ils se réduisent en quelques incrémentation de compteur et quelques tests

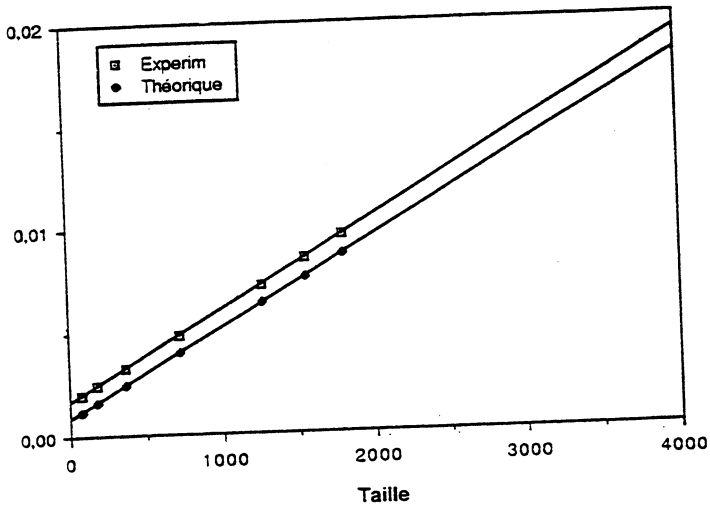
La figure 2.11 montre la performance de la procédure d'échange total sur la machine MEGANODE sur différents réseaux toriques. On remarque que la différence relative entre le coût théorique et l'expérimental (synchronisations et coût des initialisations) varie comme une fonction décroissante du nombre de processeurs du réseaux de 16 à 10 %.



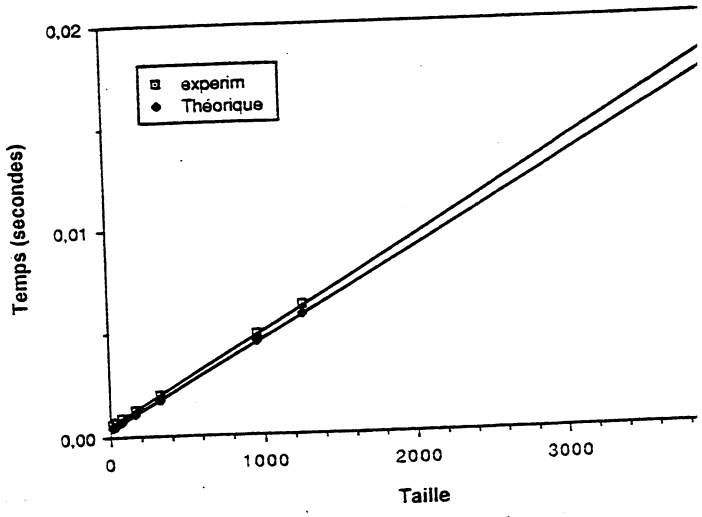
Tore (11 × 11)



Tore (9 × 9)



Tore (6 × 6)



Tore (4 × 4)

Figure 2.11: Comparaison du temps théorique et expérimental de l'échange total sur un tore

2.4.4 Conclusion sur l'échange total sur le tore

Par l'utilisation de la méthode de la superposition des arbres de recouvrement, on atteint la borne inférieure du temps de propagation de la procédure d'échange total. Nous avons montré que notre procédure utilise dans le pire des cas un seul tampon de mémoire (0 quand $\sqrt{p}$ est impair, 1 dans le cas où $\sqrt{p}$ est pair). Dans la littérature [32] [43] [49], les procédure proposées utilisent un espace mémoire tampon dont la taille est croissante avec le nombre de processeurs du réseau. La procédure que nous avons proposée s'adapte bien pour échanger des données de tailles assez grandes , toute en évitant le problème d'espace mémoire. Une extension de ce travail est présenté dans [51] qui consiste en une généralisation de la méthode dans le cas des tores à trois dimension.

2.5 Echange total avec accumulation sur le tore

Dans ce paragraphe, nous utilisons la dualité entre l'échange total et l'échange total avec accumulation pour construire une procédure de communication optimale qui résout l'échange total avec accumulation sur un tore de processeurs. L'idée est d'utiliser les arbres de recouvrement (avec un sens de parcours inverse) décrits dans les paragraphes précédents, pour effectuer l'échange total avec accumulation.

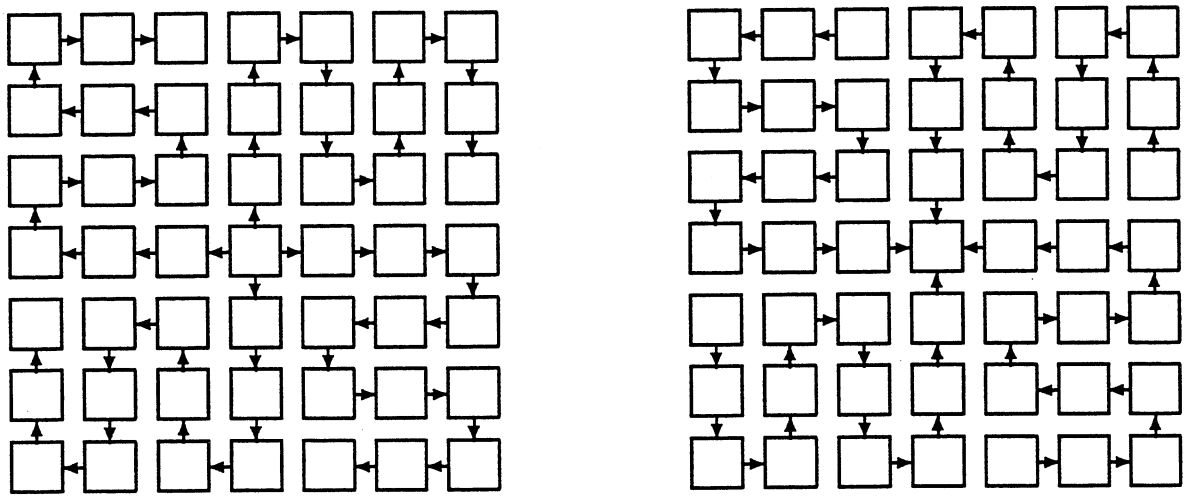
2.5.1 Principe de l'échange total avec accumulation sur le tore

On rappelle que l'échange total avec accumulation consiste à ce que chaque processeur contient p message différents. l'objectif de ce type de communication est de cumuler sur chaque processeur les messages qui lui sont destiné est qui se trouvent sur les autres processeurs. Pour réaliser cette procédure, nous allons utiliser les arbres de recouvrement du tore en inversant le chemin des données.

2.5.2 Procédure d'accumulation en utilisant les arbres de recouvrement

On rappelle que les arbres de recouvrement choisis du tore sont des arbres à 4 branches linéaires de profondeur $\frac{p-1}{4}$. Observant un arbre quelconque de racine donnée r et supposons que chaque noeud de l'arbre contient un message différent. Pour effectuer une accumulation de tous ces messages sur le noeud r, le procédé est simple :

Dans la première étape de l'accumulation, les 4 noeuds du dernier niveau ($\frac{p-1}{4}$) de



Arbre de diffusion

Arbre d'accumulation

Figure 2.12: Arbre de diffusion et d'accumulation

l'arbre r envoient leurs messages à leurs voisins du niveau supérieur. Une fois que ces derniers reçoivent les messages du niveau inférieur, ils les cumulent avec les leurs, puis ils propagent les résultats au niveau supérieur et ainsi de suite jusqu'à la racine r (figure 2.12). Nous appellons arbre d'accumulation , l'arbre de recouvrement parcouru dans le sens inverse.

2.5.3 Procédure d'échange total avec accumulation dans le tore

Nous avons vu précédemment que l'échange total avec accumulation correspond à une accumulation simultanée sur chaque noeud (multi-accumulation). Pour résoudre l'échange total avec accumulation, nous allons superposer les effets de chaque arbre d'accumulation de chaque noeud du tore.

Comme nous l'avons montrer dans le paragraphe 2.4.1, si on choisit les arbres d'accumulation $NEWS$-Equilibrés alors chaque étape de la multi-accumulation est sans contention de liens. Il en résulte que le temps pour effectuer l'échange total avec accumulation est le même que celui de l'échange total (problème dual).

Pour préciser comment se font les accumulations sur chaque noeud à chaque étape de la procédure d'échange total avec accumulation, on rappelle que si on observe les p

arbres d'accumulation chaque noeud à chaque niveau des arbres appartient à exactement 4 arbres de racines différentes dans des branches différentes. Donc à chaque étape chaque noeud envoie sur ses 4 liens des messages différents. En particulier la première étape est très importante car c'est pendant cette étape qu'on décide du choix des 4 messages à envoyer parmi les p autres messages. Pour réaliser ce choix des messages à envoyer à la première étape, chaque noeud, sachant la structure de l'arbre, sa position et sur quel lien il va émettre, calcul le numéro de la racine qu'il doit atteindre sur chacun de ses liens. Une fois ces racines r_j des arbres connues, il suffit d'envoyer le message m_j et le numéro r_j sur le lien correspondant. L'utilité de l'envoi du numéro de la racine permet une grande facilité lors du cumule des messages sur chaque noeud à chaque étape. En effet à chaque étape de la procédure chaque processeur, reçoit 4 messages différents qu'il identifie par leurs destinations finale. Il cumule chaque messages qu'il à reçu avec le message correspondant qu'il à localement. Puis transmet les messages résultants et leurs destinations finales sur les arbres d'accumulation correspondants.

2.5.4 Implémentation de l'échange total avec accumulation

Dans ce paragraphe on présent un algorithme pour réaliser un échange total avec accumulation sur un tore ($\sqrt{p} \times \sqrt{p}$) avec $\sqrt{p}$ impair, en utilisant les arbres d'accumulations qui sont des arbres parcourus dans le sens inverse que celui de l'exemple du paragraphe 2.4.1.3 . On rappelle que m est la longueur des branches SB_k.

On Définie la procédure *Communication*(l_1, l_2, l_3, l_4 : dans L) comme :

En parallèle :
- Envoi de la donnée reçu à l'étape précédente par le lien *Nord* sur le lien l_1.
- Envoi de la donnée reçu à l'étape précédente par le lien *Est* sur le lien l_2.
- Envoi de la donnée reçu à l'étape précédente par le lien *West* sur le lien l_3.
- Envoi de la donnée reçu à l'étape précédente par le lien *Sud* sur le lien l_4.

Et la procédure *PremiereEtape*() comme :

En parallèle :
- Envoie de la donnée local sur les 4 liens.
- Reçoie des données sur les 4 liens.

Algorithme sur chaque noeud :

Procédure échange total avec accumulation
{

```
CalculRacine()
PremiereEtape()
k=0; /* le numéro de la sous-branche */
m= (√p-1)/2; /* le nombre de sous-branche */
  While(k ≤ m)
    {
      for (i=1; i ≤ m-2; i++) /* Construction de la sous-branche SB_k, k pair */
       Communication(Sud, West, Est, Nord);
       Accumulation()
      if (k < m)
      {
       k=k + 1;
       Communication(Est, Sud, Nord, West); /* Changement de direction */
       Accumulation()
       Communication(West, Nord, Sud, Est); /* Changement de direction */
       Accumulation()
       for(i=1; i ≤ m-2; i++) /* Construction de la sous-branche SB_k, k impair */
        Communication(Sud, West, Est, Nord);
        if (k < m)
        {
         Communication(West, Nord, Sud, Est); /* Changement de direction */
         Accumulation()
         Communication(Est, Sud, Nord, West); /* Changement de direction */
         Accumulation()
        }
```

}

$k = k + 1$;

}

communication(Sud, $West$, Est, Est); }

La procédure $CalculRacine()$ consiste à envoyer des messages personnalisés sur chaque arbres d'accumulation. Cette procédure est très simple, elle se résume en deux phases :

- Calcul des racines des arbres : sachant la structure de l'arbre d'accumulation, chaque noeud (terminal dans l'arbre de diffusion correspondant) calcul les numéros des racines des arbres qu'il atteint sur chacun de ses quatre liens (voir le lemme 2).
- Préparation des messages personnalisés : connaissant la racine de l'arbre qu'on doit atteindre sur chacun des liens, il suffit de préparer pour l'envoi du message correspondant sur le lien considéré.

Pour faciliter l'opération d'accumulation, lors de la préparation des messages personnalisés à la première étape, on concatène aux messages, le numéro de la racine de l'arbre d'accumulation (destination finale).

2.6 Procédure de communication échange partiel personnalisé sur le tore

Dans cette procédure de communication, chaque noeud envoie un certain nombre de messages personnalisés (différents) à un groupe de noeuds. Le nombre de messages personnalisés que contient chaque processeur peut être différent d'un noeud à un autre. On retrouve ce type de procédure de communication dans le chapitre 6. Pour réaliser cette procédure, on va se baser sur les arbres $NEWS$-Equilibré.

2.6.1 Principe de l'échange partiel personnalisé sur le tore

Nous allons considérer les mêmes arbres de recouvrement $NEWS$-Equilibré construits dans les paragraphes précédents. On rappelle que ces arbres de recouvrement du tore sont structurés en quatre branches linéaires et que ces branches couvrent des régions

disjointes du tore. L'idée de base pour réaliser l'échange partiel personnalisé se résume comme suit :

La procédure comprend $\frac{p-1}{4}$ étapes de communication qui consiste à propager les messages sur des arbres de recouvrement issus de chaque noeud du tore. Observons un arbre de recouvrement issu d'un noeud quelconque du tore :

Lors de la première étape de communication, le noeud racine de l'arbre inclue tous les messages personnalisés de destinations les noeuds appartenant à la même branche dans un même tampon mémoire destiné à la branche considérée. Une fois que les tampons sont formées, on envoie chaque chaque tampon sur la branche correspondante. Pour contrôler la progression d'un tampon donné sur la branche qui lui est destinée, pendant la formation des tampons sur le noeud racine, on calcule (sachant la structure de l'arbre) à quelle profondeur de la branche correspondante, tous les messages personnalisés contenus dans le tampon vont être tous délivrés. Cette information est envoyer avec le tampon pour permettre d'arrêter l'acheminement de ce tampon à l'étape de communication correspondante. En pratique, pour des raisons de synchronisation et du contrôle de la terminaison de cette procédure, quand une fin d'acheminement d'un tampon est détectée avant l'étape $\frac{p-1}{4}$, on envoie un message vide (taille nulle) sur l'arbre de recouvrement correspondant.

2.6.2 Procédure de l'échange partiel personnalisé

Cette procédure de communication fonctionne de la même façon que la procédure d'échange total présentée précédemment. Pour la résumer, nous allons décrire les principales modification qu'il faut faire à la procédure d'échange total :

- Génération d'une table sur chaque noeud : $TableArbre[i][j]$, où i représente l'identification de la branche et j la position d'un noeud dans cette branche . $TableArbre[i][j]$ correspond à la structure de l'arbre issu de racine le noeud courant ($i \in [1..4]$, $j \in [1..\frac{p-1}{4}]$).
- Préparation des tampons à envoyer sur chaque branche, moyennant $TableArbre[i][j]$.
- Extraction d'un message du tampon, lorsque ce dernier arrive à destination et mise à jour de la taille du tampon.
- Si le numéro de l'étape de la procédure est supérieur à l'information contenu dans l'entête du tampon reçu, ie : tous les messages du tampon ont été délivrés, on renvoie un message vide.

- Quand on reçoit un message vide, en le propage sur l'arbre de recouvrement correspondant.

2.6.3 Remarques sur la procédure de l'échange partiel personnalisé

Cette procédure utilise la technique de diffusion sur les arbres $NEWS$-Equilibré , elle est donc sans conflit de lien, sans interblocage et utilise un espace mémoire de stockage réduit. Quelque soit le nombre de messages personnalisés dans chaque tampon, elle termine en $\frac{p-1}{4}$ étapes de communication. Le coût de cette procédure est très difficile à évaluer vu la nombre différents de messages personnalisés dans les temps et la dispersion géographique des noeuds destinataires dans les différentes arbres de recouvrement du tore. Le seul inconvénient de cette méthode réside dans le fait que le routage utilisés n'est pas orienter plus court chemin mais résout les interblocage et les conflits de liens. Cette procédure n'est certainement pas optimale mais vu l'effort de programmation pour là réaliser (simple modification de la procédure d'échange total) , cette solution reste très intéressante. Nous utilisons cette procédure dans le paragraphe 6.8.

2.7 Conclusion

Le temps de communication d'un algorithme parallèle constitue une partie non négligeable du temps total. Nous avons vu que pour réduire le coût de communication, il faut bien choisir le mode de communication, la topologie du réseau de communication et enfin d'utiliser une procédure de communication qui résout le problème de mouvement de données considéré en un temps minimum.

Nous avons présenté des procédures de communication efficaces qui résolvent un des problème les plus fréquent dans l'algorithmique parallèle matricielle. Nous allons retrouver ces procédures de communications lorsqu'on abordera la parallélisation de nos algorithmes de résolution des modèles.

Les différentes procédures que nous avons présentées dans ce chapitre vont être utilisées comme suit :

- L'échange total sur l'anneau : paragraphes 4.2.3 et 6.6.
- L'échange total avec accumulation sur l'anneau : paragraphe 4.2.2.
- L'échange total sur le tore : paragraphe 4.2.1 et chapitres 5 et 6.

- L'échange total avec accumulation sur le tore : paragraphe 4.2.2.
- L'échange partiel personnalisé sur le tore : paragraphe 6.8.2.

Chapitre 3

Le parallélisme au service de l'évaluation des performances

3.1 Introduction

Le développement des systèmes informatiques et l'apparition de nouvelles architectures parallèles n'est pas possible sans les outils d'évaluation des performances qui analysent le comportement de ces systèmes depuis leur conception à leur mise en fonctionnement.

Les systèmes informatiques actuels sont de plus en plus évolués et leur modélisation est de plus en plus complexe, notamment la tailles des modèles est de plus en plus grande. Le calcul parallèle apparait comme une technique possible pour évaluer ces systèmes, afin de pouvoir résoudre des modèles plus grands.

Présentation du chapitre : Dans le paragraphe 3.2, nous discutons les différentes méthodes d'évaluations des performances et leur parallélisation. Dans le paragraphe 3.3, on s'intéresse plus particulièrement aux méthodes de résolution numériques qui constituent l'objet de cette thèse et on termine par une conclusion.

3.2 Méthode d'évaluation des performances et leur parallélisation

On peut classer les méthodes d'évaluation en deux classes : celles qui évaluent les systèmes réels directement et celles qui utilisent des modèles de systèmes réels. Dans la

première classe, on retrouve la mesure, cette technique se base sur l'observation du système réel. La deuxième classe englobe les différentes méthodes de modélisation.

3.2.1 Méthode d'évaluation des performances utilisant directement le système réel

Pour évaluer un système réel, il faut l'observer, i.e : prendre des mesures. La mesure n'est possible que si le système considéré est opérationnel. Parmi les problèmes rencontrés, nous avons celui de traiter le grand nombre d'information. On peut classer la mesure en deux catégories.

- Le traitement séquentiel des mesures : cette méthode est bien connue dans la littérature, elle consiste à interpréter les résultats par des études statistiques.
- Le traitement parallèle des mesures : à notre connaissance, actuellement, il n'existe pas de méthode parallèle pour traiter les mesures. Toutefois, le développement des traceurs de programme parallèle va ouvrir un horizon dans ce domaine.

3.2.2 Méthode d'évaluation des performances utilisant des modèles du système réel

Dans ce paragraphe, nous allons décrire les méthodes de modélisation et leurs différentes résolutions.

3.2.2.1 Méthodes de modélisation

La modélisation est le passage d'un système réel à une formalisation mathématique. Plusieurs techniques de modélisation sont utilisées. Chacune d'elles est plus ou moins bien adaptée aux aspects spécifiques d'analyse des performances. Elles utilisent plusieurs types de représentations : réseaux de files d'attentes (RFA), réseaux de Petri stochastiques (RPS) et les réseaux d'automates stochastiques (RAS)...

Plus particulièrement, les réseaux de Pétri stochastique [29] [9] et les réseaux d'automates stochastiques [36] sont bien adaptés pour décrire les phénomènes de concurrence, de conflit et de synchronisation.

Une autre alternative de modélisation est donné par les réseaux de files d'attente [26]. cette méthode est certainement la plus utilisée pour estimer les performances d'un

système informatique, mais elle présente des difficultés pour exprimer la concurrence. L'extension du modèle de files attentes descriptives [30] permet l'expression de la concurrence en créant de nouveaux outils dans le modèle tel : les drapeaux, la génération, le regroupement ... etc.

3.2.2.2 Méthodes parallèles de résolution des modèles

La complexité croissante des modèles rend le problème de l'évaluation de performance difficile. En effet l'espace d'états est grand, cela induit des difficultés d'ordre mémoire. De plus le temps de l'évaluation des performances d'un modèle est une fonction croissante du nombre d'états du modèle. Dans les méthodes de résolution des modèles, on distingue 3 types :

- La simulation : on peut voir la simulation parallèle sous trois niveaux différents :
  1. Parallélisation par tâches fonctionnelles : on découpe le processus de simulation en différentes tâches. Chaque tâche correspond à une certaine fonction du simulateur : tâche de génération aléatoire, tâche de cumul des résultats, tâche de calcul statistique etc. Ces différentes tâches peuvent s'exécuter en parallèle de façon pipeliné.
  2. Parallélisation en utilisant des échantillons indépendants : la simulation est une épreuve statistique dont la précision des résultats dépend de la taille de l'échantillon. Dans ce type de parallélisation, chaque processeur traite un certain nombre d'échantillons. Ainsi, on peut traiter en parallèle des échantillons indépendants.
  3. Parallélisation géométrique : le système simulé est décomposé en sous-systèmes. Dans ce type de parallélisation, chaque processeur simule le fonctionnement d'un sous système.

  En retrouve dans la littérature la simulation parallèle sous le terme de simulation distribuée. Pour plus de détails sur ce domaine, nous référons les travaux [6] [7] [23] [28].

- Méthodes analytiques : ces méthodes s'imposent quand on veut calculer une formule mathématique (fonction paramètrée) des distributions de probabilité d'un système donné. La parallélisation de ces méthodes débouche sur des algorithmes parallèles de calcul formel [40] [54].

- Méthodes numériques : ces méthodes sont souvent utilisées dans la résolution des modèles. Si les modèles sont de type Markovien, ces méthodes sont basées sur des opérations matricielle, elles s'adaptent facilement au calcul parallèle.

Dans cette thèse, on ne s'intéresse qu'aux méthodes de résolution numériques des modèles Markoviens. Dans le paragraphe suivant, nous allons passer en revu les différentes méthodes numériques appliquées au problèmes Markoviens et leur parallélisation.

3.3 Les méthodes numériques et leur parallélisation

Soit un modèle (RFA,RPS,RAS), on se place sous des hypothèses Markoviennes. On note par T l'espace d'états du modèle et on construit un générateur Q ou une matrice de transition P du modèle. On se propose de calculer le vecteur de probabilités stationnaires π du modèle en résolvant les systèmes d'équations suivants :

- Dans le cas où l'échelle de temps est continue, il s'agit de résoudre le système d'équations
$$\pi\, Q \;=\; 0 \tag{3.1}$$
- De même, dans le cas où le temps est discret, il faut résoudre le système
$$\pi\, P \;=\; \pi \tag{3.2}$$

Notons au passage que du système d'équations 3.1, on peut le ramener sous la forme 3.2 et vice versa :

- On peut écrire 3.1 sous la forme
$$\frac{1}{\alpha}\,\pi\, Q \;=\; 0 \tag{3.3}$$
avec $\alpha \;=\; d_{max}\;(1+\epsilon)$, où d_{max} est le maximum en valeur absolue des éléments diagonaux de la matrice Q. le système 3.3 est équivalent à
$$\frac{1}{\alpha}\,\pi\, Q \;+\; \pi = \;\pi\;, \tag{3.4}$$
qu'on peut écrire
$$\pi\,(\frac{1}{\alpha}\,\pi\, Q \;+\; I) = \;\pi\;, \tag{3.5}$$
et on retrouve la forme du système 3.2.
- Pour le passage du système d'équations 3.2 à 3.1, en écrit 3.2 comme suit :
$$\pi\, P \;-\; \pi \;=\; 0 \tag{3.6}$$

et on obtient le résultat voulu

$$\pi (P - I) = 0 \tag{3.7}$$

Dans cette thèse, on s'intéresse à résoudre des modèles Markoviens à grand espace d'états. Dans ce qui suit, nous allons passer en revue les différentes méthodes numériques existantes et leur parallélisation dans le cas des modèles Markoviens.

Parmi les méthodes numérique, on distingue deux classes :

- Les méthodes numériques directes : des études de parallélisation des méthodes directes tel que la méthode de Gauss, Jordan, et les méthodes de décompositions QR, LU font l'objet d'une littérature abondante [52] [42] [50]. Cependant, quand la taille du problème à résoudre est assez grande, les méthodes directes deviennent coûteuses et mal adaptées.
- Les méthodes numériques itératives : ces méthodes se basent sur le calcul d'une suite de vecteurs qui converge vers la solution. D'une façon générale pour résoudre des problèmes de grande taille, les méthodes itératives sont préférables aux méthodes directes pour les raisons suivantes :
  - Elles s'adaptent bien au mode de stockage des matrices creuses.
  - L'arrêt des calculs est dicté par la précision des résultats voulus.
  - Les méthodes itératives sont très stables et présentent des erreurs d'arrondi très faibles.

Parmi les méthodes itératives nous distinguons deux classes :

1. Les méthodes généralistes :

   Le terme généraliste, vient du fait que ces méthodes ne tiennent pas en compte de la nature du modèle à résoudre. Parmi ces méthodes, nous allons présenter :

   - La méthode de Jacobi.
   - La méthode de Gauss-Seidel.
   - La méthode des puissances.
   - La méthode des puissances réactualisé.
   - Les méthodes utilisant la technique de projection : Arnoldi et GMRES.

2. Les méthodes spécialisées :

   Au contraire des méthodes généralistes, les méthodes spécialisées exploitent la nature du problème à résoudre. Nous allons décrire :

- La méthode d'Agrégation-Désagrégation pour les processus presque décomposable (NCD).
- La méthode de Neuts pour les processus quasi de naissances et de mort (QBD).

3.3.1 Les méthodes généralistes

Soit $X_1, X_2, X_3, ..., X_T$ des ensembles et $X=X_1 \times X_2 \times X_3 ... \times X_T$ leur produit cartésien. Les éléments de X sont des T-uples tels que pour $x \in X$ on écrit $x=(x_1, x_2, ..., x_T)$, avec $x_i \in X_i$ pour $i=1,T$.
Soit $f_i : X \rightarrow X_i$ une fonction donnée, et soit $f : X \rightarrow X$ la fonction définie par :

$$f(x) = (f_1(x),\ f_2(x), ...,\ f_T(x)), \forall\, x\ \in\ X. \tag{3.8}$$

Le problème est de trouver un point fixe de f, $\bar{x} \in X$ avec

$$\bar{x}=f(\bar{x}) \Leftrightarrow \bar{x}_i=f_i(\bar{x})\ \forall\, i\ \in\ [1..T].$$

On distingue 2 types de schémas itératifs :

- Schéma itératif de type Jacobi

$$x_i^{(t+1)} = f_i(x_1^{(t)},\ x_2^{(t)},\ ...,\ x_T^{(t)})\ \ ,\ \forall\, i\ \in\ [1..T].$$

- Schéma itératif de type Gauss-Seidel

$$x_i^{(t+1)} = f_i(x_1^{(t+1)},\ ...,\ x_{i-1}^{(t+1)},\ x_i^{(t)},\ ...,\ x_T^{(t)}),\quad \forall\, i\ \in\ [1..T].$$

3.3.1.1 Méthode du type Jacobi

Méthode de Jacobi

On veut résoudre

$$Q^*\ \pi\ =\ 0$$

ou (*) représente l'opération de transposition de matrice et π est vu comme un vecteur colonne. On peut partitionner la matrice Q^* comme suit :

$$Q^*\ =\ D\ -\ (L\ +\ U) \tag{3.9}$$

où D est une matrice diagonale, les matrices respectivement L et U sont des matrices respectivement strictement triangulaire inférieure et supérieure. L'itération de Jacobi est donnée par

$$\pi^{(t+1)} = H_J \, \pi^{(t)}$$

avec :

$$H_J = D^{-1}(L+U)$$

Comme D^{-1} existe car $d_{ii} \neq 0$, on peut écrire l'itération de Jacobi sous forme scalaire:

$$\pi_i^{(t+1)} = \frac{1}{q_{ii}} \sum_{j \neq i} q_{ji} \, \pi_j^{(t)}, i = 1, 2, \cdots, T. \tag{3.10}$$

Méthode des Puissances

On propose dans [17] les schémas itératifs suivants qui correspondent au cas discret :

$$\pi^{(t+1)} = \pi^{(t)} \, P \tag{3.11}$$

On démarre l'itération par un vecteur arbitraire $\pi^{(0)}$. La convergence de cette méthode dépend de la deuxième valeur propre λ_2 de P. Si λ_2 est proche de 1, la convergence est terriblement lente.

Parallélisation des méthodes de type Jacobi

La parallélisation des méthodes de type Jacobi consiste , pour chaque itération, en un produit matrice vecteur parallèle puis une sychronisation afin de commencer la prochaine itération. Dans le chapitre 4, nous allons étudier en détails la parallélisation du produit vecteur-matrice.

3.3.1.2 Méthode du type Gauss-Seidel

Méthode de Gauss-Seidel

On rappelle qu'on veut résoudre le système

$$Q^* \, \pi = 0$$

Dans une itération de la méthode de Gauss-Seidel le calcul des composantes utilise les valeurs les plus récentes des composantes. l'écriture scalaire de l'itération de Gauss-Seidel est donnée par

$$\pi_i^{(t+1)} = \frac{1}{d_{ii}} (\sum_{j=1}^{i-1} l_{ij}\, \pi_j^{(t+1)} + \sum_{j=i+1}^{T} u_{ij}\, \pi_j^{(t)}), i = 1, 2, \cdots, T. \quad (3.12)$$

l'écriture matricielle de l'itération de Gauss-seidel est

$$\pi^{(t+1)} = (D - L)^{-1}\, U\, \pi^{(t)}.$$

De cette dernière équation on identifie la matrice d'itération de Gauss-Seidel H_{GS} par

$$H_{GS} = (D - L)^{-1}\, U.$$

Méthode de Gauss-Seidel avec relaxation

Pour accélérer la convergence de la méthode de Gauss-Seidel, on utilise le schéma itérative suivant :

$$\pi_i^{(t+1)} = (1-w)\, \pi_i^{(t)} + w\, [\frac{1}{d_{ii}} (\sum_{j=1}^{i-1} l_{ij}\, \pi_j^{(t+1)} + \sum_{j=i+1}^{T} u_{ij}\, \pi_j^{(t)})], i = 1, 2 \cdots, T. \quad (3.13)$$

qui s'écrit sous forme matricielle comme suit

$$\pi_i^{(t+1)} = (1-w)\, \pi^{(t)} + w\, [D^{-1}\, (L\, \pi^{(t+1)} + U\, \pi^{(t)})] \quad (3.14)$$

Ce schéma itérative est appelé sur relaxation pour $w > 1$, et sous relaxation pour $w < 1$. Cette méthode converge plus rapidement que la méthode de Gauss-Seidel simple pour $0 < w < 2$.

Dans une itération de la méthode de Gauss-Seidel le calcul des composantes utilise les valeurs les plus récentes des composantes. On note cette aspect par l'effet Gauss-Seidel.

En reste dans le cadre de l'accélération de la convergence de la méthode de Gauss-Seidel, on présente dans ce qui suit, une version par blocs.

Méthode de Gauss-Seidel par blocs

Cette méthode à été présenté dans [47]. On considère le partitionnement suivant du système

$$\pi\, Q = 0$$

définit par

$$(\pi_1,\ \pi_2,\ ...,\ \pi_N) \begin{pmatrix} Q_{11} & Q_{12} & ... & Q_{1N} \\ Q_{21} & Q_{22} & ... & Q_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ Q_{N1} & Q_{N2} & ... & Q_{NN} \end{pmatrix}$$

On peut écrire :

$$Q^* \ = \ D_N \ - (L_N \ - \ U_N)$$

avec D_N est une matrice bloc diagonale, L_N et U_N sont respectivement des matrices strictement bloc triangulair inférieure et supérieure.

La méthode de Gauss-Seidel par bloc s'écrit :

$$(D_N \ - \ L_N)\ \pi^{(t+1)} \ = \ U_N\ \pi^{(t)}$$

Soit :

$$D_{ii}\ \pi_i^{(t+1)} \ = \ (-\sum_{j=1}^{i-1} L_{ij}\ \pi_j^{(t+1)} \ - \sum_{j=i+1}^{N} U_{ij}\ \pi_j^{(t)})\ i = 1..N.$$

Cela revient qu'à chaque itération, il faut résoudre N systèmes linéaires

$$D_{ii}\ \pi_i^{(t+1)} \ = \ z_i \ , \ i = 1..N$$

avec

$$z_i \ = \ (-\sum_{j=1}^{i-1} L_{ij}\ \pi_j^{(t+1)} \ - \sum_{j=i+1}^{N} U_{ij}\ \pi_j^{(t)})\ i = 1..N.$$

Pour résoudre ces systèmes, on peut utiliser une méthode directe si les matrices D_{ii} sont de taille relativement petites. Il suffit dans ce cas de former une fois pour toute les matrices $L_i\ U_i$ de chaque bloc diagonal. Si les matrices bloc diagonales sont de grandes tailles, on peut utiliser l'une des méthodes itérative présentée précédemment.

Remarques

Cette méthode de Gauss-Seidel par bloc génère plus de calcul par itération que la méthode de Gauss-Seidel simple, mais elle converge plus rapidement que cette dernière.

Méthode des puissances réactualisée

Dans le schéma d'itération des puissances réactualisée, on utilise le même principe d'itération que dans le cas classique mais de plus, le calcul des composantes utilise les valeurs les plus récentes.

Dans le schéma d'itération des puissances réactualisée, on utilise le même principe d'itération que dans le cas classique mais de plus, le calcul des composantes utilise les valeurs les plus récentes. Ce schéma d'itération rappel le schéma de Gauss-Seidel, dont le quel on utilise un découpage de la matrice P comme suit :

$$P = (D_P - L_P) - U_P,$$

on obtient puisque la matrice $(I + U_P)$ est inversible, le schéma itératif qui s'écrit comme suit :

$$\pi^{(t+1)} = \pi^{(t)} (D_P - L_P) (I + U_P)^{-1}.$$

Parallélisation des méthodes de type Gauss-Seidel

Nous avons vu que les méthodes de type Gauss-Seidel présentent des contraintes de précédence fortes de mise à jour des composantes (ou bloc de composante, dans la version par bloc). Dans la version Gauss-Seidel par blocs, la résolution du bloc $D_{i+1,i+1}$ dépend des résultats de la résolution du bloc D_{ii}. Toute fois, dans le cas particulier des matrices tridiagonales par bloc, dans une parallélisation où en distribue les résolutions des blocs sur les processeurs et du fait que chaque bloc n'est en interaction qu'avec deux blocs voisins, on peut paralléliser la méthode efficacement .

3.3.1.3 Comparaison entre les méthodes de type Jacobi et Gauss-Seidel

Des comparaisons théorique [15] [53], montrent que les méthodes de type Gauss-Seidel convergent plus rapidement que ceux du type Jacobi. A titre de validation de ce résultats des expérimentations de ces deux méthodes le confirme (voir l'annexe C) . D'autre part, les méthodes de type Jacobi se parallélisent bien, par contre les méthodes de type Gauss-Seidel , ne s'adaptent pas bien au calcul parallèle.

Dans le cadre de la parallélisation des méthodes itératives, il faut faire donc un compromis entre la vitesse de convergence (Gauss-Seidel) et le temps de calcul par itération (Jacobi).

3.3.1.4 Méthodes itératives utilisant la technique de projection

Nous allons présenter la méthode d'Arnoldi qui une méthode itérative typique qui utilise la technique de projection. On note que dans cette méthode, on veut résoudre le système

$$P^* \pi = \pi$$

qu'on écrit sous la forme

$$(P^* - I)\ \pi\ = 0 \tag{3.15}$$

avec π comme vecteur colonne. On pose $A = (P^* - I)$. La méthode, d'Arnoldi [57], est une méthode de projection sur un sous espace de Krylov [44] $(v_1,\ A\,v_1,\ A^2\,v_1, ..., A^{m-1}\,v_1)$. Le principe de cette méthode est d'approcher le vecteur de probabilité π par une combinaison linéaire de m vecteurs $v_i = A^i\,v_1,\ i = 1..m$ qui forme une base orthonormée.

L'algorithme d'Arnoldi se compose de 3 parties :

- Construction de la base de krylov.
- Formation du vecteur approximé à une itération donnée.
- Test de convergence.

Algorithme d'Arnoldi

Début

Choisir un vecteur initial π_0 et m la dimension de la base
Tantque *precision non atteinte*
Calculer $r_0\ = -A\,\pi_0$
Calculer $\beta\ = \|r_0\|_2$
Calculer $v_1\ = \frac{r_0}{\beta}$
/* Formation de la base du sous espace */
Pour j=1 à m
Calculer $w\ =\ A\,v_j$
Pour $i = 1$ à j
Calculer le produit scalaire $h_{ij} =< w,\ v_i >$
Calculer $w\ = w - h_{ij}\,v_i$
Finpour
Calculer $h_{j+1,j} = \|w\|_2$
Calculer $v_{j+1} = \frac{w}{h_{j+1,j}}$
Finpour
/* Formation du vecteur approximé */
Calculer $\pi_m\ =\ \pi_0\ +\ V_m\ y_m$ avec :
$V_m\ =\ (v_1,\ v_2, ...,\ v_m)$
$y_m\ =\ H_m^{-1}\ \beta\ e_1$
$e_1\ =\ (1,\ 0, ...,\ 0)$
/* Test de convergence */

Si $h_{m+1,m}\ |e_1\ y_m| < precision$
Alors $precision\ atteint = vrai$
Sinon $\pi_0 = \pi_m$ et on itère
FinTantque
Fin

Une autre méthode qui utilise la technique de projection est la méthode GMRES. Cette méthode est une procédure de minimisation au sens des moindres carrés dans le sous espace de krylov [44] [57]. La méthode GMRES est similaire à la méthode d'Arnordi, elle ne diffère de celle ci que par la manière de minimiser le résidu.

On peut proposer dans le cadre d'une extension de cette thèse d'étudier une parallélisation des méthodes de projection d'Arnoldi et GMRES qui est principalement un parallélisme par distribution de données. A chaque itération, il faudrait effectuer :

- Un produit matrice vecteur parallèle (voir le chapitre 4)
- Un produit scalaire parallèle pour calculer β
- La construction de la base : le calcul des m vecteurs v_i ne peut pas s'effectuer en parallèle car le vecteur v_j dépend des vecteurs de rang inférieur. Cette partie de l'algorithme se résume à m multiplications matrice vecteur parallèles.
- Formation de la matrice H sur chaque processeur : qui consiste principalement en des produits scalaires parallèles.
- Formation du vecteur π_m approximé : consiste à faire m produits scalaires parallèles plus une somme parallèle de 2 vecteurs.

Dans la parallélisation des méthodes de projections , les difficultés qu'il faudrait résoudre sont l'enchainement et la synchronisation entre les différentes phases de l'algorithme. En effet, dans les phases qui effectuent le produit scalaire parallèle, le coût de communication est du même ordre que le coût de calcul. Par contre, dans les phases de produit matrice-vecteur, le coût de communication est faible devant le temps de calcul.

3.3.2 Méthodes itératives spécialisés

Dans les problèmes de chaines de Markov, il est fréquent que l'espace d'états peut être partitionner en sous ensembles. Dans ce cas la matrice de transition est partitionnée en accord avec le partitionnement des sous ensembles d'états et des méthodes itératives par blocs sont développées. Dans ce qui suit, nous allons présenter deux types de méthodes:

- Méthode d'Agrégation-Désagrégation (pour processus (NCD)).
- Méthode de Neuts (pour processus (QBD)).

3.3.2.1 Méthode itérative d'Agrégation-Désagrégation pour les processus (NCD)

Cette approche de décomposition pour résoudre les chaines de Markov, se base sur le principe de diviser pour reigner :

Si le système à résoudre est très grand et complexe, on le découpe en sous-systèmes de tailles plus petites. Chaque sous-système est analysé séparément, puis en forme la solution global en rassemblant les solutions de chaque sous-système. En pratique, il est rare de trouver des chaines de Markov qui peuvent être décomposer en sous-chaines indépendantes. Par contre une classe de problèmes assez importante, apparait fréquemment dans la modélisation Markovienne, qui sont caractérisés comme suit :

- L'espace d'états est partitionné en sous-ensemble disjoints.
- Les interactions entre les états d'un même sous-ensemble sont fortes.
- Les interactions entre les états de sous-ensembles différents sont très faibles.

Ce genre de problème est appelé processus presque décomposable (NCD). Les premiers travaux sur les système (NCD) sont dûs à Simon et Ando [46], leurs recherches étaient axés sur l'évolution dynamique des modèles économiques. Ce concept à été étendu aux chaines de Markov et à l'étude des systèmes informatiques par Courtois [10].

Notation est Définition

La matrice de transition des systèmes (NCD) est de la forme :

$$P = \begin{pmatrix} P_{11} & P_{12} & \dots & P_{1N} \\ P_{21} & P_{22} & \dots & P_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ P_{N1} & P_{N2} & \dots & P_{NN} \end{pmatrix}$$

dans la quelle les éléments non nuls des blocs non diagonaux sont très petits comparés à ceux des blocs diagonaux. Les sous-blocs P_{ii} sont carrés et d'ordre n_i , $i = 1..N$, avec

$T = \sum_{i=1}^{N} n_i$. On suppose que :

$$\|P_{ii}\| = O(1) \quad i = 1..N$$

et

$$\|P_{ij}\| = O(\epsilon) \quad i \neq j$$

où $(\|.\|)$ est la norme spéctrale et ϵ un réel suffisamment petit.

Algorithme Itératif d'agrégation désagrégation (IAD)

L'algorithme d'agrégation-désagrégation se décompose en phase :

- Phase de décomposition : approximation du vecteur de probabilité (procédure de Courtois)
- Phase d'agrégation : calcul des probabilités des blocs.
- Phase de désagrégation : effectue un certain nombre d'itération par une méthodes itérative choisie. Dans la littérature [5] [37] [45] , la méthode itérative la plus recommander est la méthode de Gauss-Seidel par blocs .

Algorithme (IAD)

1. Soit le vecteur initial $\pi^{(0)}_{Courtois} = (\pi_1^{(0)}, \pi_2^{(0)}, ..., \pi_N^{(0)})$ donné par la procédure de Courtois (voir ci-dessous). On pose $t = 1$.
2. Calculer $\Phi^{(t-1)} = (\Phi_1^{(t-1)}, \Phi_2^{(t-1)}, ..., \Phi_N^{(t-1)})$ avec :

$$\Phi_i^{(t-1)} = \frac{\pi_i^{(t-1)}}{\|\pi_i^{(t-1)}\|_1}$$

3. Construire la matrice d'agrégation $A^{(t-1)}$:

$$a_{ij}^{(t-1)} = \Phi_i^{(t-1)} P_{ij}\, e, \quad e = (1,\ 1,\ ...,\ 1)$$

4. Chercher le vecteur de probabilités de $A^{(t-1)}$:

$$\xi^{(t-1)} A^{(t-1)} = \xi^{(t-1)}, \quad \|\xi^{(t-1)}\|_1 = 1$$

5. a) Calculer le vecteur $z^{(t)}$:

$$z^{(t)} = (\xi_1^{(t-1)} \Phi_1^{(t-1)}, \xi_2^{(t-1)} \Phi_2^{(t-1)}, ..., \xi_N^{(t-1)} \Phi_N^{(t-1)})$$

b) Résoudre les N systèmes d'équations pour trouver $\pi^{(t)}$:

$$\pi_i^{(t)} = \pi_i^{(t)} P_{ii} + \sum_{j>i} z_j^{(t)} P_{ji} + \sum_{j<i} z_j^{(t-1)} P_{ji}$$

Cette méthode correspond à un schéma de Gauss-Seidel par Bloc.

6. Test de convergence : si on atteint la précision voulu, on s'arrête. Sinon, on fait $t = t + 1$ et aller à 2.

Procédure de Courtois (algorithme de décomposition)

1. Analyser chaque bloc diagonal P_{ii} pour trouver le sous-vecteur μ_i de probabilités des états du bloc i (conditionnés par l'événement d'être au bloc i).

$$\mu_i P_{ii} = \lambda_{i_1} \mu_i \quad , \mu_i e = 1$$

où e est un vecteur de dimension appropriée, dont tous les composantes sont égales à 1.

2. Formation de la matrice d'agrégation A_{ag} dont les éléments sont donnés par :

$$a_{ij} = \frac{\mu_i P_{ij}}{\|\mu_i\|_1}$$

3. Déterminer le vecteur de probabilités des blocs (probabilité d'être dans un bloc i) :

$$\xi A_{ag} = \xi, \quad \xi e = 1$$

4. Former le vecteur de probabilités approximé :

$$\pi_{Courtois} = (\xi_1 \mu_1, \xi_2 \mu_2, ..., \xi_N \mu_N)$$

Parallélisme dans la méthode d'agrégation désagrégation

Dans ce paragraphe, nous discutons le parallélisme dans l'algorithme (IAD).

- Le premier point de l'algorithme (IAD) présente un parallélisme évident car les résolutions des blocs diagonaux sont indépendantes.
- Pour construire la matrice A_{ag} (point 3 de l'algorithme (IAD)), on peut procèder de deux manières :

- On suppose que N est relativement petit, on duplique la matrice A_{ag} sur chaque processeur. Une idée de la construction de cette matrice se base sur un découpage de la matrice P en N colonnes blocs. Si on affecte un bloc colonne à chaque processeur, on peut former une colonne de la matrice A_{ag} par processeur. Une étape de communication du type échange total vient en suite pour dupliquer la matrice A_{ag} sur tous les processeurs.

- On suppose que N est grand, on ne duplique pas la matrice A_{ag}, mais en forme des colonnes de cette matrice sur les processeurs comme dans le cas précédent.

• Le point 4 se décompose en deux cas, selon qu'on duplique ou pas la matrice A_{ag} sur les processeurs :

- Cas de la matrice A_{ag} dupliqué sur tous les processeurs : la résolution du problème du vecteur de probabilités de la matrice A_{ag} par une méthode directe semble la plus appropiée (N est relativement petit). Le point 4 n'est donc pas paralléliser.

- Cas où la matrice A_{ag} est répartie en colonne sur les processeurs de la machine. La résolution du problème du vecteur de probabilités de la matrice A_{ag} peut être soit une méthode parallèle directe soit par une méthode itérative. Le choix de l'une ou l'autre méthode (directe et itérative) découlera de l'étude de la vitesse de convergence des méthodes itératives par rapport au coût de calcul des méthodes directes. En effet, pour le type de matrice A_{ag} particulier, les valeurs propres de ces matrices sont toutes proches de l'unité. Les méthodes itératives en général convergent lentement pour ce type de matrice.

• Le point 5 de l'algorithme (IAD) : consiste à effectuer une itération avec un schéma de Gauss-Seidel par bloc. Nous avons vu précédemment que ce schéma itérative n'est pas efficacement parallélisable. On peut penser à remplacer cette itération par un schéma de la méthode des puissances classique parallèle. Dans le cas séquentiel, des comparaisons entre les différents types d'itération pour résoudre le point 5, montre expérimentalement que le remplacement de l'itération de Gauss-Seidel bloc par la méthode des puissances ralenti considérablement la convergence de l'algorithme (IAD) [37]. Nous somme une fois de plus devant le dilème vitesse de convergence et bonne parallélisation.

L'étude de la parallélisation des méthodes d'agrégation désagrégation fait partie de nos travaux futures.

3.3.2.2 Méthode de Neuts pour les processus (QBD)

Rappelons rapidement cette méthode, le générateur infinitésimal Q a une structure tridigonal par blocs :

$$Q = \begin{pmatrix} B_1 & C & & \\ B_2 & A_1 & A_0 & \\ & A_2 & A_1 & A_0 \\ & \cdots & \cdots & \cdots \end{pmatrix}$$

où B_1 est une matrice carrée de taille b et les matrices A_0, A_1 , A_2 sont également carrées de dimension a. La solution stationnaire du système est définie par le vecteur des probabilités π vérifiant : $\pi\, Q = 0$ et $\pi\, e = 1$ où e désigne le vecteur colonne dont toutes les coordonnées sont égales à 1. On peut partitionner π en $\pi = (\pi_0,\ \pi_1,\ \pi_2, ...)$ où π_0 est un vecteur de dimension b et les π_i, $i = 1, 2, ...$ sont des vecteurs de dimension a.

Le théorème suivant est établi par Neuts :

Théorème 2 *En notant $A = A_0 + A_1 + A_2$ le générateur infinitésimal d'un processus Markovien à espace d'états fini, et π le vecteur des probabilités stationnaires :*

- *Le processus défini par la matrice Q est récurrent positif si et seulement si*

$$\pi\, A_2\, e < \pi\, A_0\, e$$

(condition de stabilité)

- *Il existe une matrice R définie non négative, et de rayon spectral inférieur à 1, telle que*

$$\pi_i = \pi_1\, R^{i-1}, \qquad i > 1$$

- *Les vecteurs π_0 et π_1 sont déterminés par le système :*

$$\pi_0\, B_1 + \pi_1\, B_2 = 0$$

$$\pi_0\, C + \pi_1\, (A_1 + R\, A_2) = 0$$

$$\pi_0\, e + \pi_1\, (I - R)^{-1}\, e = 0$$

- *La matrice R est la solution unique, définie non négative et de rayon spectral inférieur à 1 de l'équation quadratique*

$$A_0 + R A_1 + R^2 A_2 = 0$$

Le calcul de la matrice R se fait par récurrence à partir d'une matrice initial R_0, selon l'équation

$$R_{n+1} = A_0 (-A_1)^{-1} + R_n^2 A_2 (-A_1)^{-1},\ n > 0$$

que l'on peut résoudre numériquement.

Vu la méthode de décomposition donnée par Neuts, la taille du problème à été réduite considérablement. De ce fait L'algorithme de Neuts ne nécessite pas de parallélisation. Nous avons implémenté cette méthode sur une machine séquentiel [24] et nous avons remarquer que la convergence de cette méthode est très lente. Plusieurs axes de recherches courant vise à améliorer sa convergence.

3.4 Conclusion

L'évaluation de performance des systèmes informatique engendre des espace d'états très grands. Devant cette complexité, l'avènement du calcul parallèle parait comme une solution possible.

Nous avons présenté différentes méthodes parallèles de résolution des modèles et nous nous sommes particulièrement intéresser aux méthodes de résolutions numériques du fait de leur facile mise en oeuvre et leur bonne adaptation au calcul parallèle.

Nous avons vu que le produit vecteur matrice parallèle est la brique de base des méthodes numériques itératives parallèles que nous avons présenté. Dans le chapitre suivant, nous allons justement étudier en détails la parallélisation du produit vecteur matrice.

Parmi les méthodes numériques présenter dans le paragraphe 3.3, dans cette thèse nous avons choisi d'étudier la plus fondamentale qui est la méthode de type Jacobi (la méthode de puissance).

Dans le cadre de la continuation de cette thèse, nous sommes intéressés par l'étude de la parallélistion des méthodes de projection et la méthode d'agrégation désagrégation.

Enfin , nous avons vu que le parallélisme et la vitesse de convergence des méthodes du type Jacobi et Gauss-Seidel sont deux aspects antagonistes, dans le chapitre 4, nous reviendrons sur l'étude de ce dilemme.

Chapitre 4

Parallélisation du produit matrice vecteur

4.1 Introduction

Nous avons vu dans le chapitre précédent que le produit matrice vecteur est la base des méthodes itératives. Le problème qu'on veut étudier est la parallélisation du produit matrice vecteur appliqué à un processus itératif

$$\pi^{(t+1)} = \pi^{(t)} P.$$

La différence avec un produit matrice vecteur classique réside dans le fait que l'opération à effectuer est l'affectation $\pi := \pi P$. L'emplacement des composantes du vecteur π doit être le même avant et après le produit.

Revenons à notre produit matrice vecteur et analysons l'expression qui le défini :

$$v_j = \sum_{i=1}^{T} \pi_i P_{ij} \ , \ j = 1, T. \tag{4.1}$$

Dans le cadre d'une parallélisation à grain fin, l'opération de base de l'expression 4.1 est le produit élémentaire : $\pi_i P_{ij}$. En effet ces produits élémentaires sont indépendants les uns des autres. En fait, dans un produit matrice vecteur, on peut exécuter en parallèle T^2 produits $\pi_i P_{ij}$. Cette opération élémentaire constitue le grain de la parallélisation. Quand le calcul des T^2 produits est terminé, il faut effectuer T sommes indépendantes, contenant T termes chacunes. On peut voir donc le produit matrice vecteur comme la succession de deux phases :

- La première phase est constituée de T^2 tâches indépendantes qu'on peut exécuter en un temps unitaire (le temps d'un seul produit $\pi_i \; P_{ij}$) en utilisant une machine à $p = T^2$ processeurs. Cette phase est essentiellement constituée de calcul, elle définie le grain du parallélisme.
- La deuxième phase consiste en un mouvement de données pour effectuer les sommes. Le mouvement des données consiste à des communications entre processeurs. Il est évident que le placement des données influent directement sur le coût des mouvement des données.

Nous avons vu au chapitre 2 que le mouvement des données induit des surcoût qu'il faut réduire. On rappelle que si le surcoût dû au communication est important par rapport au coût du calcul, le gain de performance du parallélisme peut être très réduit.

Dans le cas général, le grain fin implique l'utilisation d'une machine à un nombre élevé de processeurs (dans notre cas de l'ordre de T^2 processeurs). Dans cette gamme de machine, en retrouve la Connection Machine (TMC), la Paragone (INTEL) ... etc. Pour plus de détails sur l'implémentation du produit matrice vecteur avec un grain fin, nous référons les travaux [56] [33].

Dans cette thèse nous allons nous intéresser à un parallélisme gros grain sur une machine (MIMD). Par exemple : on décide que le produit scalaire est le grain de la parallélisation .

Quelque soit le type de machine utilisé pour résoudre un problème donné, il est important de choisir un bon grain de parallélisation et une bonne répartition des données pour réduire le mouvement de données.

Dans le pragraphe 4.2, nous allons présenter différentes implantations du produit matrice vecteur, chacune d'elles se distingue par le choix du grain et le mouvement des données. Une comparaison théorique de ces méthodes est donnée dans le paragraphe 4.3, qui est suivi par une expérimentation du produit matrice vecteur sur la machine MEGANODE. Enfin, on termine par une conclusion.

4.2 Présentation des différentes implantation du produit matrice vecteur

On rappelle que le vrai problème que nous étudions est le produit matrice vecteur appliqué à un processus itératif. Dans ce problème, il y a une contrainte de plus que dans le problème classique. Cette contrainte impose que la disposition des données avant et

après le produit est la même. Cela se traduit par un placement de donnée (phase 1) et un mouvement de données (phase 2) différent selon qu'on traite un produit classique ou un produit adapté à l'itération. Cependant, le problème de parallélisme est le même dans les deux cas, à savoir, le choix du bon grain et le placement des données pour réduire leur mouvement.

Les idées directrices de parallélisation dans les méthodes que nous allons présenter sont :

- Placement des données (matrice, vecteur) sur le processeur de la machine de façon à équilibrer la charge de calcul sur chaque processeur. ie : étant donné que le coût du produit est de l'ordre de $O(T^2)$, en parallèle, on vise à l'effectuer en $O(\frac{T^2}{p})$ avec p le nombre de processeurs de la machine.
- Mouvement des données, respectant la contrainte du produit itéré et réduction des communications.

4.2.1 Méthode de décomposition par colonne (C)

Dans cette méthode, on s'inspire de la définition du produit matrice vecteur comme une succession de produit scalaire indépendant :

$$v_j = \sum_{i=1}^{T} \pi_i \, P_{ij} \quad , \; j = 1, \; T.$$

On choisi le grain de parallélisme comme le produit scalaire élémentaire v_j. En effet, les T produits scalaires sont indépendants les uns des autres. Comme T est généralement grand devant p, on décompose les T produits scalaires en groupe de taille $\frac{T}{p}$, on suppose que T est divisible par p. On identifie une tâche par groupe de produit scalaire et on effectue chaque tâche sur un processeur. Ainsi, la charge de calcul par processeur est équilibrée [15].

- **Répartition du calcul**

  Dans cette méthode, les processeurs p_l sont identifiés par un numéro l=0, p-1.
  On affecte au processeur p_l les colonnes consécutives de numéros l $(\frac{T}{p})$+1 jusqu'à l $(\frac{T}{p})$+$\frac{T}{p}$. Chaque processeur contient un bloc de colonnes de taille $\frac{T}{p}$. Le vecteur $\pi^{(t)}$ est recopié en entier sur chaque processeur. Dans le cas où p divise T, la charge de calcul est équi-répartie sur les processeurs de la machine : le processeur p_l calcule localement les composantes de $\pi^{(t+1)}$ du rang l $(\frac{T}{p})$+1 au rang l $(\frac{T}{p})$+$\frac{T}{p}$. Appelons

sous-vecteur affecté au processeur p_l cet ensemble de composantes et notons Jl l'ensemble des indices de ces composantes.

- **Partie communication**

  Une fois que tous les processeurs ont terminé le calcul du sous-vecteur dont ils sont chargés, pour reconstituer le vecteur $\pi^{(t+1)}$ résultat sur chaque processeur, il faut que chaque processeur envoie son sous-vecteur à tous les processeurs, soit un schéma de communication du type *échange total*. Nous allons choisir une procédure *échange total* optimale [1]. On rappelle le principe de cette procédure :

  L'idée est que chaque processeur diffuse son sous-vecteur sur un arbre de recouvrement du tore. La procédure *échange total* est constituée de $\frac{p-1}{4}$ étapes, où chaque processeur reçoit et envoie simultanément sur ses 4 liens 4 sous-vecteurs. A la fin de la procédure chaque processeur a reconstitué le vecteur résultat et peut donc recommencer l'étape de calcul pour constituer une méthode itérative.

4.2.2 Méthode de décomposition par ligne (L)

Dans cette méthode, le produit matrice vecteur est vu comme suit :

$$v = \sum_{i=1}^{T} L_i$$

où L_i est un vecteur ligne de taille T dont les éléments l_j sont donnés par :

$$l_j = \pi_i P_{ij} \quad , j = 1, T.$$

On choisi le grain de parallélisme comme le calcul d'un vecteur ligne L_i. En effet le calcul des T vecteurs lignes L_i sont indépendant les uns des autres. Comme dans la méthode précédente, on définit des tâches où chacune d'elles est constituée d'un groupe de $\frac{T}{p}$ vecteurs ligne L_i à calculer (même hypothèse, T divisible par p). En affectant chaque tâche à un processeur, on exécute en parallèle les p tâches. Ainsi , la charge de calcul est équilibrée sur chaque processeur [15].

- **Répartition du calcul**

  Dans cette méthode, les processeurs sont repérés de la même manière que la méthode précédente. On affecte à chaque processeur p_l les lignes de la matrice de numéros $l\ (\frac{T}{p})+1$ jusqu'à $l\ (\frac{T}{p})+\frac{T}{p}$. Chaque processeur contient un bloc de ligne de taille $\frac{T}{p}$ et un sous-vecteur correspondant au $\frac{T}{p}$ composantes du vecteur $\pi^{(t)}$ du numéro $l\ (\frac{T}{p})+1$ jusqu'à $l\ (\frac{T}{p})+\frac{T}{p}$. Le processeur p_l calcul les $\frac{T}{p}$ termes de la somme qui définie chaque composante du vecteur $\pi^{(t+1)}$.

- **Partie communication**

  Une fois que le processeur p_l a terminé le calcul des $\frac{T}{p}$ termes de la somme qui définie chaque composante du vecteur $\pi^{(t+1)}$, il est nécessaire d'accumuler les termes correspondant aux composantes du vecteur $\pi^{(t+1)}$ du numéro l $(\frac{T}{p})+1$ jusqu'à l $(\frac{T}{p})+\frac{T}{p}$. Pour ce faire chaque processeur envoie un sous-vecteur de taille $\frac{T}{p}$ constitué des $\frac{T}{p}$ termes de la somme qui définie les composantes du vecteur $\pi^{(t+1)}$ affectées au processeur p_l. Soit un schéma de communication du type *échange total avec accumulation* . Pour réaliser ce schéma de communication, on utilise la même technique des arbres de recouvrement du tore [1] que pour la procédure *d'échange total* (voir paragraphe 2.4).

4.2.3 Méthode de décomposition ligne-colonne (LC)

Dans cette méthode, on choisit le grain de parallélisme comme le produit élémentaire $(\pi_i\ P_{ij})$ (voir l'expression 4.1). Le calcul des T^2 produits élémentaires sont indépendants. On décompose l'ensemble des T^2 produits en groupes de taille $\frac{T^2}{p}$. Chaque groupe définit une tâche de calcul [19].

- **Répartition des données et du calcul**

  On subdivise la matrice P en sous-matrices blocs de taille $(\frac{T}{p} \times \frac{T}{p})$. On repère ces sous-matrices par un double indice P_{ij} avec $i, j = 0..p\text{-}1$. Les processeurs dans le tore sont repérés par une double indexation, p_{lc} avec $l, c = 0..\sqrt{p} - 1$. La fonction d'allocation des sous-matrices P_{ij} aux processeurs est la suivante :

  A chaque processeur est alloué p sous-matrices P_{ij}. Chaque processeur contient $\frac{T^2}{p}$ éléments de la matrice P. On affecte la matrice P_{ij} au processeur p_{lc} si $l = i\ mod \sqrt{p}$ et $c = \frac{j}{\sqrt{p}}$. On subdivise le vecteur en p blocs de taille $\frac{T}{p}$, les blocs sont notés $\pi_k^{(t)}$ pour $k = 0..p\text{-}1$. On affecte le sous-vecteur $\pi_k^{(t)}$ au processeur p_{lc} avec $l = k\ mod \sqrt{p}$ et $c = \frac{k}{\sqrt{p}}$. Chaque processeur p_{lc} calcule $\sqrt{p}$ sous-vecteurs v_{lck}, les indices l, c identifie le processeur qui a calculé le sous-vecteur, l'indice k est tel que les composantes du sous-vecteur v_{lck} sont des somme partiels des composantes du sous-vecteur $\pi_k^{(t+1)}$. Sur le processeur p_{lc}, k varie de $c\ \sqrt{p}$ jusqu'à $c\ \sqrt{p} + \sqrt{p} - 1$. Les sous-vecteurs v_{lck} sont calculés par :

$$v_{lck} = \sum_{m=0}^{\sqrt{p}-1} \pi^{(t)}_{(m\ \sqrt{p}+l)}\ P_{(m\ \sqrt{p}+l\ ,\ k)}$$

P_{00}	P_{01}	P_{02}	P_{03}
P_{10}	P_{11}	P_{12}	P_{13}
P_{20}	P_{21}	P_{22}	P_{23}
P_{30}	P_{31}	P_{32}	P_{33}

Figure 4.1: Découpage en bloc de la matrice P

Pour fixer les idées nous allons voir la répartition des données et du calcul sur un exemple .

Exemple :

Soit une machine à 4 processeurs connectés en une topologie torique. les figures 4.1 et 4.2 montrent l'affectation des sous blocs de la matrice P sur les processeurs de la machine. La répartition du calcul est la suivante :

- Calcul sur le processeur p_{00} :

$$\begin{aligned} v_{000} &= \pi_0^{(t)} P_{00} + \pi_2^{(t)} P_{20} \\ v_{001} &= \pi_0^{(t)} P_{01} + \pi_2^{(t)} P_{21} \end{aligned}$$

- Calcul sur le processeur p_{01} :

$$\begin{aligned} v_{012} &= \pi_0^{(t)} P_{02} + \pi_2^{(t)} P_{22} \\ v_{013} &= \pi_0^{(t)} P_{03} + \pi_2^{(t)} P_{23} \end{aligned}$$

- Calcul sur le processeur p_{10} :

$$\begin{aligned} v_{100} &= \pi_1^{(t)} P_{12} + \pi_3^{(t)} P_{32} \\ v_{101} &= \pi_1^{(t)} P_{11} + \pi_3^{(t)} P_{33} \end{aligned}$$

- Calcul sur le processeur p_{11} :

$$\begin{aligned} v_{112} &= \pi_1^{(t)} P_{12} + \pi_3^{(t)} P_{32} \\ v_{113} &= \pi_1^{(t)} P_{13} + \pi_3^{(t)} P_{33} \end{aligned}$$

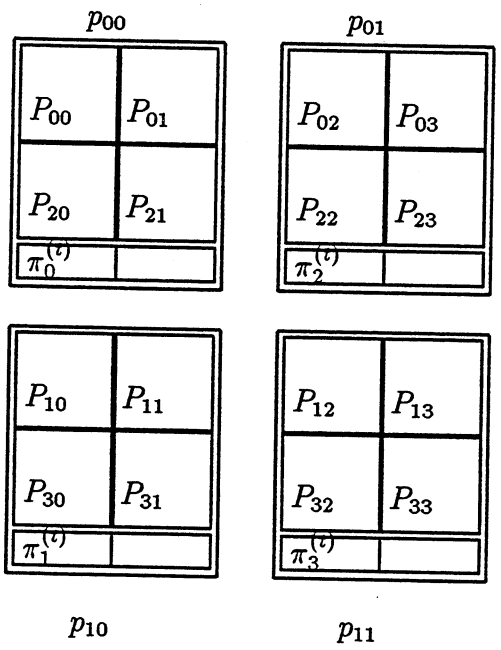


Figure 4.2: Allocation des sous-vecteurs et des sous-matrices aux processeurs

On remarque que pour pouvoir commencer le calcul, il faut que les processeurs qui se trouvent sur le même anneau horizontal du tore (le même numéro l) s'échangent leur sous-vecteur $\pi_k^{(t)}$. Après calcul, chaque processeur reçoit et cumule les sous-vecteurs calculés, comme suit :

- Sur le processeur p_{00}, on forme $\pi_0^{(t+1)}$:

$$\pi_0^{(t+1)} = v_{000} + v_{100}$$

- Sur le processeur p_{10}, on forme $\pi_1^{(t+1)}$:

$$\pi_1^{(t+1)} = v_{101} + v_{001}$$

- Sur le processeur p_{01}, on forme $\pi_2^{(t+1)}$:

$$\pi_2^{(t+1)} = v_{012} + v_{112}$$

- Sur le processeur p_{11}, on forme $\pi_3^{(t+1)}$:

$$\pi_3^{(t+1)} = v_{113} + v_{013}$$

Soit un échange de sous-vecteurs calculés entre les processeurs qui appartiennent au même anneau vertical (le même numéro c).

- **Partie communication**

  Etant donné les fonctions d'allocation des matrices P_{ij} et des sous-vecteurs $\pi_k^{(t)}$ sur les processeurs, la partie communication se décompose en deux phases :

  1. Communication avant le calcul (anneaux horizontaux) : cette phase de communication consiste à effectuer une procédure du type *échange total* sur les processeurs appartenant à la même ligne du tore (le même indice l) et cela sur toutes les lignes en parallèle (voir le chapitre 2.3.1). Cette procédure a pour effet de rapatrier les blocs de vecteur $\pi_k^{(t)}$ nécessaires sur chaque processeur pour entamer la phase de calcul (voir l'exemple).
  2. Communication après le calcul (anneaux verticaux) : cette phase de communication effectue une procédure du type *échange total avec accumulation* en parallèle sur chaque anneau vertical de processeurs (processeurs ayant le même indice c). Cette procédure a pour effet de cumuler les sous-vecteurs v_{*ck} pour former le sous-vecteur $\pi_k^{(t+1)}$ sur le même processeur qui contient le sous-vecteur $\pi_k^{(t)}$ (voir le paragraphe 2.3.2).

  Pour un calcul itératif, il suffit d'itérer le processus décrit précédemment.

Nous avons vu que ces méthodes présentent des mouvements de données différents. Le coût de ces méthodes est exposé dans le paragraphe suivant.

4.3 Comparaison théorique entre les trois méthodes

Du point de vue des temps de calcul les trois méthodes sont équivalentes (la charge de calcul globale est inchangée et est divisée par p sur chaque processeur). Les temps de communication de chaque méthode sont les suivants (on néglige les temps d'initialisations des communications) :

- Le temps de communication de la méthode décomposition par colonne (C) est donné par :

$$tcom_C = \frac{p-1}{4}\ \tau_{4pb}\ \frac{T}{p}$$

  Ce qui est le temps de *l'échange total* pour des échanges de vecteurs de taille $\frac{T}{p}$ [1]. La constante τ_{4pb} représente la vitesse du lien d'une communication bi-directionnelle sur 4 liens en parallèle (voir le paragraphe 1.3).

- La méthode de décomposition par ligne (L) à le même coût que la méthode (C) (voir le paragraphe 2.5).

$$tcom_L = \frac{p-1}{4}\ \tau_{4pb}\ \frac{T}{p}.$$

- Le coût de communication la méthode de décomposition (LC) est meilleur :

$$tcom_{LC} = \sqrt{p}\ \tau_{2pb}\ \frac{T}{p}.$$

  Ce qui est le temps d'un *échange total* pour les échanges de vecteurs de taille $\frac{T}{p}$ sur les anneaux horizontaux du tore (voir le paragraphe 2.3.1), plus le temps d'un *échange total avec accumulation* sur les anneaux verticaux du tore (voir le paragraphe 2.3.2).

La méthode décomposition (LC) est plus efficace en temps de communication. La différence des temps de communication entre la méthode (C) et (LC) est donnée par :

$$diff_{CLC} = (\frac{p-1}{4}\ \tau_{4pb}\ -\ \sqrt{p}\ \tau_{2pb}\ )\frac{T}{p}$$

Vu que les modèles de calcul des coûts des communications ne tiennent pas en compte des phénomènes de synchronisation entre les processeurs, une expérimentation du produit matrice vecteur est présenté pour valider la comparaison théorique.

4.4 Expérimentation du produit matrice vecteur sur le MEGANODE

Dans ce paragraphe, nous allons comparer expérimentalement les deux méthodes (C) et (LC) . La méthode (L) n'est pas expérimenté car elle est équivalente à la méthode (C). Nous allons essayer d'estimer, expérimentalement, la différence de coût des méthodes (C) et (LC).

4.4.1 Résultat d'accélération des deux méthodes (C) et (LC)

La figure 4.3 confirme la supériorité de la méthode (LC), on remarque que les courbes d'accélération des deux méthodes pour les grandes tailles de matrice sont proches. Tout se passe comme si le coût des communications devient faible devant le coût de calcul. Pour confirmer cela, nous allons comparer les temps de calcul et des communications des méthodes (C) et (LC).

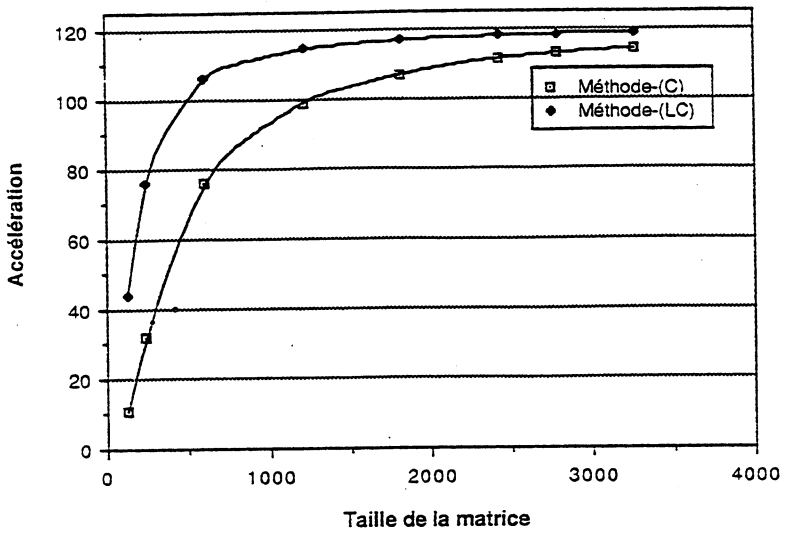


Figure 4.3: Accélération des méthodes (C) et (LC) sur 121 processeurs

4.4.2 Temps de calcul et communication des méthodes (C) et (LC)

Dans la figure 4.4, on voit la dominance du temps de calcul par rapport aux temps de communications. On remarque pour les grandes tailles (figure 4.5) , le temps de communication de la méthode (C) représente 2% du temps total. De plus l'expérimentation montre que le rapport entre les temps de communication des méthodes (C) et (LC) est de l'ordre de 2 dans le cas des grands modèles (de l'ordre de 4000 états).

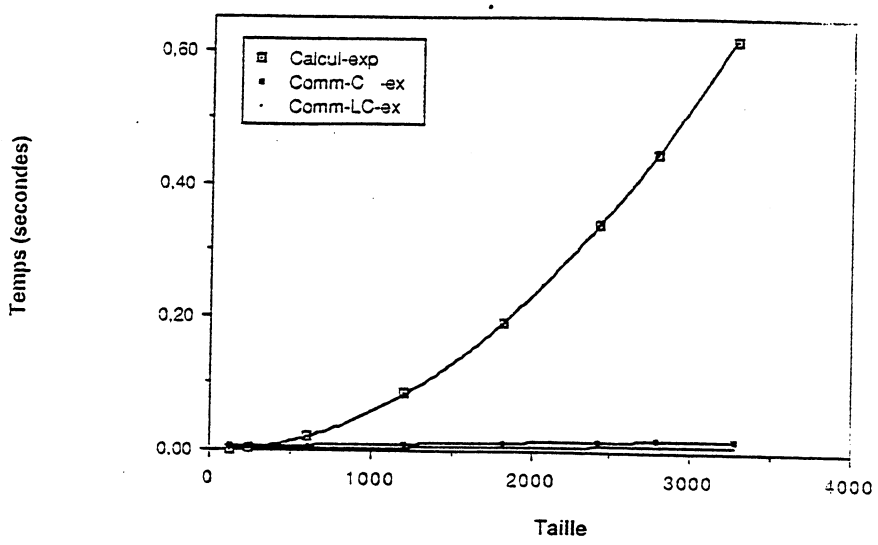


Figure 4.4: Comparaison entre le temps de communication et le temps de calcul

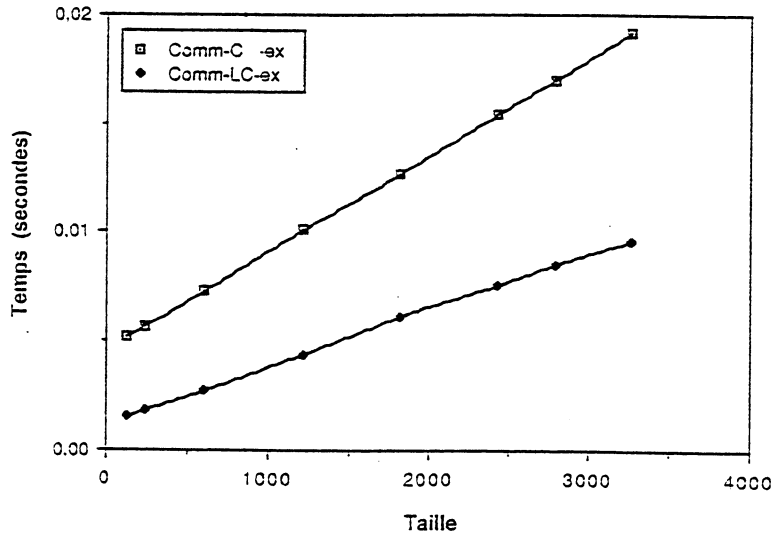


Figure 4.5: Comparaison des temps de communication des méthodes (C) et (LC)

4.5 Conclusion

Nous avons vu dans le paragraphe 4.3, que la méthode (LC) est meilleure que la méthode (C). L'expérimentation à montrer que pour les grandes tailles de la matrice, la différence entre ces deux méthodes est très faible.

Dans le chapitre suivant, on va comparer des méthodes itératives. Nous allons utiliser le produit de base selon la méthode (C) pour les raisons suivantes :

- La différence de coût entre les méthodes (C) et (LC) est faible pour les grandes tailles
- Nous allons dévelloper des méthodes itératives où la méthode (LC) n'est pas applicable (voir les paragraphe 5.3 , 5.6, 5.7, pour une explication détaillé). Or nous voulons expérimenter diverses implantations des méthodes itératives en prenant la même opération $\pi := \pi\ P$ de base afin de ne pas introduire une biaison supplémentaire.

Dans le chapitre précédent, nous avons souligné le dilemme entre la vitesse de convergence et la bonne parallélisation des méthodes de type Jacobi et Gauss-Seidel. Dans le chapitre suivant, nous allons justement étudier ce problème dans le cadre du calcul asynchrone.

Chapitre 5

Méthodes itératives parallèles de résolution des problèmes Markoviens

5.1 Introduction

Nous avons vu précédemment que l'utilisation d'un schéma itératif de type Gauss-Seidel permet une convergence rapide, mais le problème c'est que ce type de schéma à des contraintes de précédence algorithmique qui le rend completement séquentiel dans le cas général. Par contre les schémas de type Jacobi ont des contraintes de précédence moins fortes et s'adaptent bien au calcul parallèle. Le but du calcul asynchrone est de rallier les avantages des deux types de schémas.

L'itération asynchrone a été introduite par Chazan et Miranker [11](1969) sous le nom de relaxation chaotique pour résoudre des systèmes linéaires. Dans l'implémentation asynchrone de l'itération $\pi^{(t+1)}=f(\pi^{(t)})$, les processeurs n'ont plus besoin d'attendre de recevoir toutes les valeurs générées pendant l'itération précédente, chaque processeur modifie les composantes du vecteur dont il est chargé, librement, si une valeur courante d'une composante mise à jour par un autre processeur n'est pas disponible, alors une ancienne valeur de cette composante est utilisée. De plus les processeurs ne sont pas contraints à communiquer leurs résultats après chaque itération, mais quand ils le décident (voir annexe A).

L'exécution asynchrone peut offrir les avantages suivants :

1. Réduction du temps de communication et du temps d'attente dûe aux synchronisations

2. Accélération de la convergence par l'effet Gauss-Seidel tout en consevant le parallélisme de type Jacobi

Dans le paragraphe 5.2, nous présentons la méthode itérative de base qui est du type Jacobi. Cette méthode va nous permettre de comparer les différents schémas itératifs asynchrones que nous allons développer le long de ce chapitre. Un des aspects du calcul asynchrone est de pouvoir prendre en compte les dernières mises à jour de chaque composante. Cette idée existe aussi dans les schémas du type Gauss-Seidel. Appliquer un schéma de type Gauss-Seidel strictement implique un ordre total sur les tâches (mise à jour des composantes). On peut l'appliquer partiellement de façon à éliminer un certain nombre de contraintes de précédence. Cette idée est développer au paragraphe 5.3.

Un autre aspect du calcul asynchrone est de réduire les communications. Nous allons étudier l'effet du retard en isolation par des méthodes présentées dans les paragraphes 5.4 et 5.5. La notion de retard, exprime le fait que dans les mises à jour à l'itération (t), on utilise des composantes qui proviennent de l'itération $(t-k)$, avec k un entier naturel. Enfin, sachant que la machine parallèle que nous utilisons autorise, sur chaque noeud, un parallélisme entre le calcul et la communication, nous allons présenter une implantation qui est très proche de la définition du calcul asynchrone : sur chaque noeud, un processus calcule itérativement en utilisant les dernières mise à jour, alors qu'un processus de communication, participe de façon continue à une suite d'échanges totaux. Cette méthode est présentée dans le paragraphe 5.6 et une variante de cette méthode est présenter dans le paragraphe 5.7. Le paragraphe 5.8, résume les différentes schémas itératives, en déduire d'autres schémas qui sont constitués d'une combinaison des différents schémas itératives. Dans le paragraphe 5.9 nous allons expérimenter les différents schémas itératifs en les comparant à la méthode itérative de base. On termine ce chapitre par une conclusion.

Dans l'annexe A , nous allons passer en revue les conditions et théorèmes d'applications des méthodes *chaotiques* ou itérations asynchrones et quelques exemples de modèles relatifs aux chaines de Markov.

5.2 Schéma itératif de base

On s'intéresse à rechercher le vecteur de probabilités stationnaire π d'une chaine de Markov de matrice de transition P, en utilisant l'itération

$$\pi^{(t+1)} = \pi^{(t)} P.$$

La méthode que nous allons utiliser est la méthode des puissances qui est du type Jacobi (voir le paragraphe 3.3.1.1). Le produit de base (π P) qu'on effectue à chaque itération est la méthode (C) décrite dans le chapitre précédent. On rappelle que dans la méthode (C), le mouvement des données (communication) est réalisé par la procédure d'échange total présentée dans le paragraphe 2.4.

Par la suite, cette méthode de base, va nous permettre de comparer les différents schémas itératifs que nous développons dans ce chapitre.

5.3 Schéma itératif avec effet de Gauss-Seidel local

Nous avons vu précédemment que l'utilisation des mises à jour les plus récentes semble accélérer la vitesse de convergence d'un schéma itératif, comme ce qui est le cas des schémas du type Gauss-Seidel. Or les méthodes de type Gauss-Seidel présentent de fortes contraintes de précédence lors de la mise à jour des composantes. De l'autre côté, les méthodes du type Jacobi ont l'avantage d'être libres de toutes contraintes de précédence de mise à jour, donc s'adapte bien au calcul parallèle. Nous proposons une méthode itérative mixte qui utilise un parallélisme de type Jacobi et un schéma de mise à jour de Gauss-Seidel partiel ou local. En d'autre termes, l'idée est la suivante :

- Faire la même distribution des données sur les processeurs que dans les schéma du type Jacobi (voir le paragraphe 4.2.1).
- Effectuer la méthode de Gauss-Seidel pour la mise à jour des composantes locales à chaque processeur.

Le schéma de mise à jour des composantes du sous-vecteur π_{jl} affecté au processeur est le suivant :

$$\pi_i^{(t+1)}, i \in Jl \text{ se calcule par les composantes } \begin{cases} \pi_j^{(t+1)} & l\left(\frac{T}{p}\right)+2 \leq j < i \\ \pi_j^{(t)} & j \geq i \\ \pi_j^{(t)} & j = l\left(\frac{T}{p}\right)+1 \end{cases}$$

Remarques :

- Dans le paragraphe 5.9, On remarque que le gain en itérations de cette méthode réactualisée parallèle est d'au moins 10% par rapport à la méthode de puissance classique.
- Dans la méthode (LC), chaque processeur effectue un calcul partiel sur le sous-vecteur dont il dispose. La méthode (LC) ne peut donc pas supporter ce schéma itératif.

5.4 Schéma itératif à retard k

Dans ce schéma, on veut exploiter le retard pour réduire les communications. La notion de retard, exprime le fait que dans les mises à jour à l'itération (t), on utilise des composantes qui proviennent de l'itération $(t-k)$, avec k un entier naturel. On effectue donc k itérations selon la méthode réactualisée du paragraphe précédent, sans communiquer les données aux autres processeurs. L'idée est de réduire les communications à un échange total pendant chaque k itérations

Algorithme 1.

```
Début
  t = 1;
  Tantque NonConvergence
          Si t Mod k + 1 = 0
          Alors AllToAll();
                TestConvergence(); {* voir l'annexe B *}
          Sinon Produit();
  t = t + 1;
  Fintantque
Fin
```

Nous avons vu précédemment que l'accélération de la convergence semble être liée avec le fait qu'on utilise des données les plus récentes (Gauss-Seidel). Cette méthode à retard, va à l'encontre de ce principe, donc la vitesse de convergence doit être pire que le cas du schéma itératif de base (voir l'expérimentation paragraphe 5.9). Il y a donc un compromis à réaliser entre la vitesse de convergence et la réduction des communications.

5.5 Amélioration du coût de calcul du produit pour un schéma à retard k

Etant donné que chaque processeur itère sur ses composantes locales sans envoyer ni recevoir de nouvelle valeurs des composantes mises à jour par les autres processeurs, on remarque dans l'expression du calcul des composantes locales, des sommes partiels invariantes pendant les k itérations en isolation. On peut donc optimiser le produit vecteur matrice, car ces constantes peuvent être évaluer une fois pour toute pour chaque groupe de k itérations.

On note par J l'ensemble des indices des éléments du vecteur local correspondant au bloc de colonnes affectées à un processeur donné (voir le paragraphe 4.2.1)). Supposons que l'itération courante de numéro (t) est constituée de la séquence *Produit*() (méthode (C)) suivit d'une procédure *AllToAll*() (procédure d'échange totale). On se place dans le contexte suivant :

On veut effectuer k itérations locales (sans communiquer les nouvelles valeurs). Calculons le nouveau vecteur π à l'itération $(t+1)$ par l'appelle de la procédure *Produit*() :

$$\pi_{(j\in J)}^{(t+1)} = \sum_{l\notin J} \pi_l^{(t)} P_{lj} + \sum_{l\in J}^{j-1} \pi_l^{(t+1)} P_{lj} + \sum_{l\in J}^{l\geq j} \pi_l^{(t)} P_{lj}. \tag{5.1}$$

Calculons le vecteur à l'itération $t+2$ (appel de *Produit*()) :

$$\pi_{(j\in J)}^{(t+2)} = \sum_{l\notin J} \pi_l^{(t)} P_{lj} + \sum_{l\in J}^{j-1} \pi_l^{(t+2)} P_{lj} + \sum_{l\in J}^{l\geq j} \pi_l^{(t+1)} P_{lj}. \tag{5.2}$$

On remarque que le premier terme

$$A_J = \sum_{l\notin J} \pi_l^{(t)} P_{lj}$$

des expressions 5.1 et 5.2 est invariant pour tous numéro d'itération inférieur ou égal à $t+k$ (ici $k=2$). Dans ce qui suit nous allons voir comment on évalue le terme A_J :

De l'équation 5.1 on tire,

$$\sum_{l\notin J} \pi_l^{(t)} P_{lj} = \pi_{(j\in J)}^{(t+1)} - \sum_{l\in J}^{j-1} \pi_l^{(t+1)} P_{lj} - \sum_{l\in J}^{l\geq j} \pi_l^{(t)} P_{lj} \tag{5.3}$$

Injectons A_J dans 5.2,

$$\pi_{(j\in J)}^{(t+2)} = \pi_{(j\in J)}^{(t+1)} - \sum_{l\in J}^{j-1} \pi_l^{(t+1)} P_{lj} - \sum_{l\in J}^{l\geq j} \pi_l^{(t)} P_{lj} + \sum_{l\in J}^{j-1} \pi_l^{(t+2)} P_{lj} + \sum_{l\in J}^{l\geq j} \pi_l^{(t+1)} P_{lj} \qquad (5.4)$$

Le coût de calcul du produit en utilisant l'équation 5.2 est de l'ordre de $O(\frac{T^2}{p})$ opération (multiplication suivit d'addition). Par contre si on utilise l'équation 5.4, le coût du produit est de l'ordre de $O(\frac{2T^2}{p^2})$ opérations, ce qui constitue un gain considérable sur tout pour les grandes valeurs de p.

Algorithme 1 amélioré.

Début
  Produit();
  AllToAll();
  Produit();
  $t = 1$;
  Tantque *NonConvergence*
    Si t Mod k = 0
    Alors *AllToAll*();
      Produit();
      TestConvergence();
    Sinon *produitAmeliore*(); {* selon l'expression 5.4 *}
  $t = t + 1$;
  FinTanque
Fin

Dans ce qui suit nous allons nous intéresser aux schémas itératifs qui favorise l'effet Gauss-Seidel.

5.6 Schéma asynchrone libre version 1

Dans ce schéma asynchrone, les processeurs mettent à jour leur composante tout en incorporant dans leur calcul, le plus rapidement possible les valeurs récentes envoyées par les autres processeurs. Etant donné que la machine parallèle que nous utilisons permet un parallélisme entre les communications et les calculs, l'idée est de lancer en parallèle un processus de calcul (mise à jour des composantes) et un processus de communication

(pour envoyer les nouvelles valeurs des composantes), le processus de communication n'est rien d'autre qu'une boucle sur la procédure d'échange total décrite dans le paragraphe 2.4. Du fait des caractéristiques de notre machine, ce schéma va donc favoriser l'effet Gauss-Seidel.

Algorithme 2.

```
Début
   (* Lancement concurrent du processus produit *)
   ProcRun(produitrept())
   Tantque NonConvergence
            AllToAll();
            TestConvergence();
   FinTanque
Fin
```

Le processus *produitrept*() est une boucle sur *produit*(), on arrête cette boucle dès que la convergence est atteinte. Etant donné que le temps d'exécution du processus *produit*() est supérieur au temps d'exécution du processus *AllToAll*(), pendant le temps d'exécution d'un calcul, on va exécuter plusieurs fois le processus communication (*AllToAll*()). Théoriquement le coût dû aux communications est très faible car pendant le temps que le processus *AllToAll*() est bloqué sur une communication, l'ordonnanceur donne la main au processus *produitrept*(). En pratique, ce coût dépend du taux de recouvrement des communications avec le calcul pour une machine donnée (voir le paragraphe 1.5).

Nous avons vu au paragraphe 4.4.2 que pour les grandes tailles, le temps du produit matrice vecteur (itération) est grand devant le temps d'une communication (échange total). De ce fait, le schéma qu'on vient de proposer, va générer plus de communication que l'algorithme itérative de base. En plus, si on ne contrôle pas le processus répetitif de communication, il est possible qu'on envoie plusieurs fois les mêmes données. Dans le schéma asynchrone qu'on va présenter au paragraphe suivant, on va essayer de contrôler le nombre de communication (échange total) pendant la durée d'un produit (itération).

5.7 Schéma asynchrone libre version 2

Dans le schéma que nous allons proposer, on essaie d'améliorer le schéma précédent. Pour éviter d'envoyer plusieurs fois les mêmes données, on pourait imaginer de mar-

quer les composantes qui ont été modifiées et n'envoyer que ceux là. Une autre piste d'amélioration, consiste à lancer en parallèle les deux processus de calcul et de communication tout en fixant le nombre d'échange total pendant la durée du produit. Bien entendu, les caractéristiques de la machine influent pour beaucoup de la méthode à utiliser.

Pour l'expérimentation, on c'est restreint à étudier l'amélioration qui consiste à fixé le nombre de communication à 1 pendant la durée d'un produit.

Algorithme 3.

```
Début
   Tantque NonConvergence
     Début
       ProcRun(produit()) (* lancement concurrent du processus produit *)
       ProcRun(AllToAll()) (* lancement concurrent du processus alltoall *)
       synchronisation() (* synchronisation des deux processus calcul, communication *)
       TestConvergence();
     Fin
Fin
```

5.8 Résumé des schémas itératifs

Dans les schémas itératifs que nous avons présenté, certains exploitent le retard pour réduire les communications, d'autres exploitent l'effet de Gauss-Seidel. On peut très bien imaginer de nouveaux schémas itératifs hybrides par combinaison des différents schémas itératifs. A titre d'exemple en peut combiner le schéma à retard k avec le schéma asynchrone libre, constituant un nouveau schéma itératif qui utilise une succession de groupe d'itérations selon les deux schémas, etc ...

En pratique, des simulations d'algorithmes asynchrones sur machine à mémoire partagée [3] donnent de bons résultats. Dans ce qui suit nous allons voir l'expérimentation des schémas itératifs précédents sur une machine à mémoires distribuées (MEGANODE) configurée en un tore de (11×11) Transputers.

5.9 Résultats expérimentaux sur les méthodes asynchrone

5.9.1 Plan d'expérience et choix des problèmes à tester

Pour l'expérimentation des schémas itératifs, nous allons tester des types de matrices qu'on retrouve dans la modélisation Markovienne des systèmes.

Généralement, les matrices issues de modèle Markovien sont creuses. Dans le paragraphe 5.9.2, nous allons tester nos différents schémas itératifs sur ce type de matrice.

Dans le paragraphe 5.9.3, on va tester un autre type de matrice qu'on retrouve fréquemment en pratique, c'est les matrices correspondant au modèle presque décomposable (NCD). Ce type de matrice a une structure en blocs. On ne considère que des matrices qui ont une structure particulière. Le paragraphe 5.9.4 résume l'expérimentation sur des matrices tridiagonales par bloc. On retrouve ce type de matrice dans les modèles quasiment de naissance et de mort (QBD)...

Enfin, dans le paragraphe 5.9.5, on teste un modèle réel dont la matrice de transition est facile à générer. Ce modèle représente la marche aléatoire sur une grille de dimension deux.

Dans l'annexe D, nous présentons les différentes méthodes de génération des différentes types de matrices que nous allons expérimenter. Pour assurer les conditions d'applications du théorème d'asynchronisme partiel, toutes les matrices qu'on va générer possèdent une sous- et sur-diagonale non nulle pour assurer l'irréductibilité de la chaine de Markov associée. De plus tous les états de la chaine de Markov contiennent une boucle pour assurer l'apériodicité et la condition (1) sur l'asynchronisme partiel pour les matrices stochastiques à savoir qu'il existe au moins un indice $\bar{i}$ tel que $P_{\bar{i}\bar{i}} > 0$ (voir annexe A).

Dans toutes les expériences qui vont suivre, pour chaque type de matrice, on prend la moyenne du nombre d'itérations et le temps moyen sur un échantillon de taille 5 de matrices aléatoires .

5.9.2 Matrices creuses

L'expérience consiste à observer le temps et le nombre d'itérations des différents schémas asynchrones en variant le pourcentage d'éléments non nuls pour diverses tailles de la matrice.

Sur la figure 5.1, on voit bien que le nombre d'itérations du schéma à retard k est croissant en fonction de k. On voit aussi le gain en itération de la méthode itérative avec effet de Gauss-Seidel local par rapport à la méthode de base (méthode des puissances). On remarque un faible gain en itérations du schéma asynchrone libre rapport à la méthode de base. On retrouve ce phénomène pour tous les types de matrices. Dans les figures 5.2, 5.3 et 5.4 , on remarque le gain de temps considérable (la moitie du temps du schéma de base) obtenu par les schémas à retard k version optimisée.

5.9.3 Matrices du type presque décomposable (NCD)

Dans cette expérience on génère des matrices bloc diagonale, tous les éléments qui n'appartiennent pas aux blocs diagonaux sont inférieurs à 0.00001. On fait varier la taille des blocs ainsi que la taille des matrices.

D'après les figures 5.6 et 5.7, on remarque un gain de temps considérable des schémas à retard k versions optimisés (presque le tiers) par rapport au schéma de base dans le cas où la taille des blocs coincide avec la taille du sous-vecteurs qu'on affecte à chaque processeur. En effet, dans ce cas tous se passe comme si chaque processeur résout un sous-système, sachant que l'interaction entre les sous-systèmes est faible (NCD).

5.9.4 Matrices tridiagonales par bloc

Dans cette expérience on génère des matrices tridiagonales par bloc et on fait varier de la même manière que dans le paragraphe précédent la taille du bloc. D'après les figures 5.9, 5.10, on remarque le même phénomène que dans le cas (NCD).

5.9.5 Matrices du type marche aléatoire sur une grille 2D

L'expérience consiste à observer le temps et le nombre d'itérations des différents schémas en variant la taille de la matrice. D'après la figure 5.12, on remarque un gain de temps des schémas à retard (de moitie) par rapport au schéma de base.

5.9.6 Conclusion

L'expérimentation des schémas asynchrones libres , nous a montré que pour ce type de schéma le gain dû à l'effet Gauss-Seidel est faible sur une machine à mémoire distribuée

à moins qu'elle soit équipée d'un réseau de communication très rapide. D'autre part, les schémas à retard de k versions optimisées donnent un très bon résultat pour les matrices aléatoires creuse qui n'ont pas de structure particulière et dans le cas des matrices blocs quand la taille des blocs coincide avec la taille des sous-vecteurs affectés aux processeurs. Dans ces cas, le gain de la méthode à retard par rapport à la méthode de base est de l'ordre de 3. On remarque que le schéma à retard k appliqué à une matrice du type (NCD) présente des similitudes de fontionnement avec les méthodes d'agrégation-désagrégation présentées dans la litérature. En fin un choix du degré d'asynchronisme k s'impose, pour englober toutes les expériences effectuées. On choisit k=3 qui donne des résultats satisfaisant dans tous les cas.

5.9.7 Figures

Légende des figures

- synch-class : schéma itératif de base selon la méthode de puissance
- synch-react : schéma itératif à effet de Gauss-Seidel local (puissance réactualisée)
- ask k : schéma itératif à retard k version optimisée
- asl : schéma itératif asynchrone libre version 1
- asc : schéma itératif asynchrone libre version 2

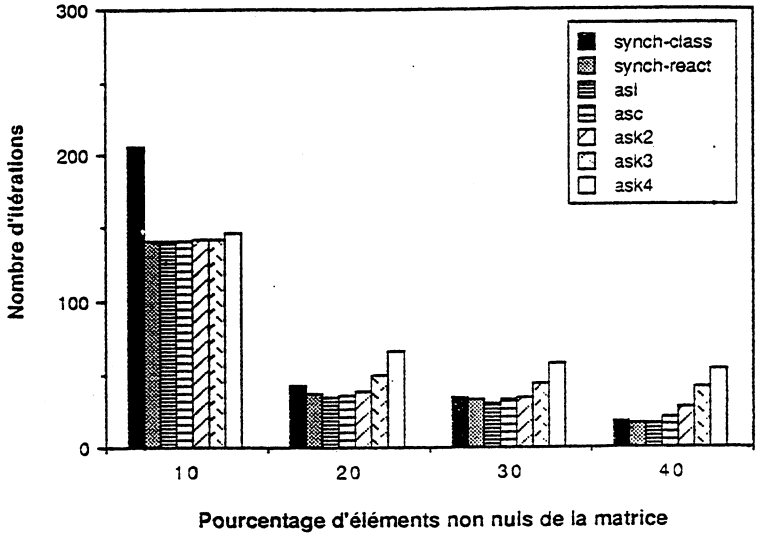


Figure 5.1: Nombre d'itération pour une matrice de taille 3267 (matrices creuses)

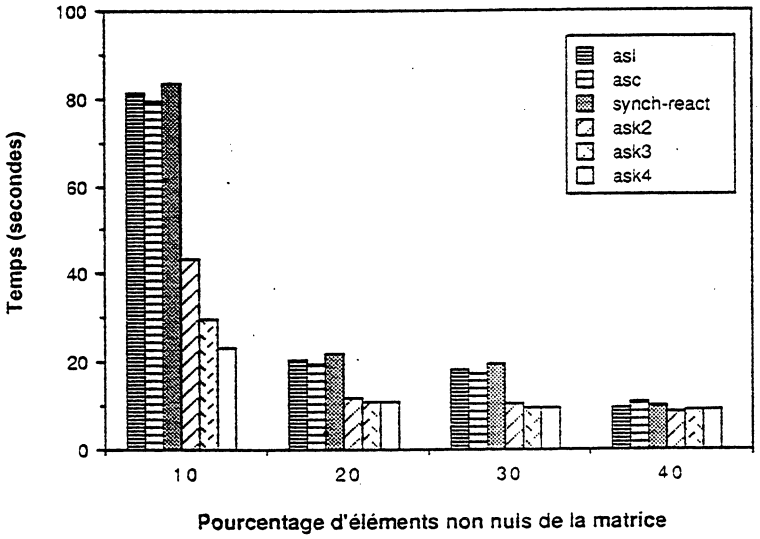


Figure 5.2: Temps de convergence pour une matrice de taille 3267 (matrices creuses)

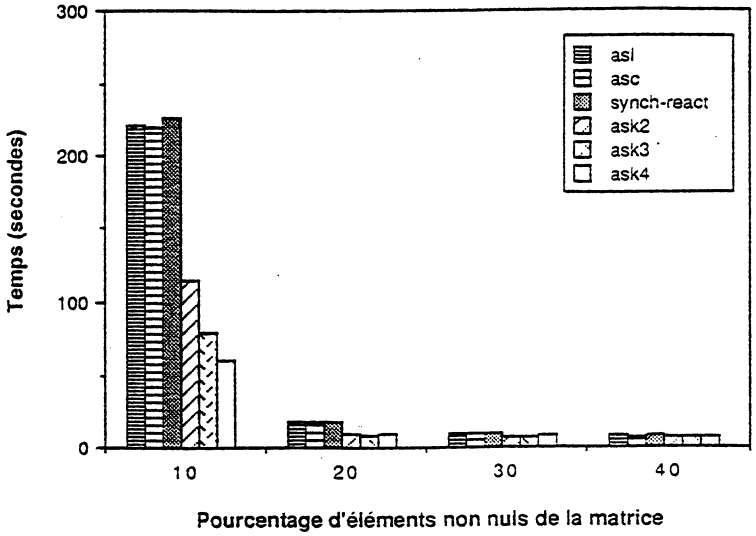


Figure 5.3: Temps de convergence pour une matrice de taille 2420 (matrices creuses)

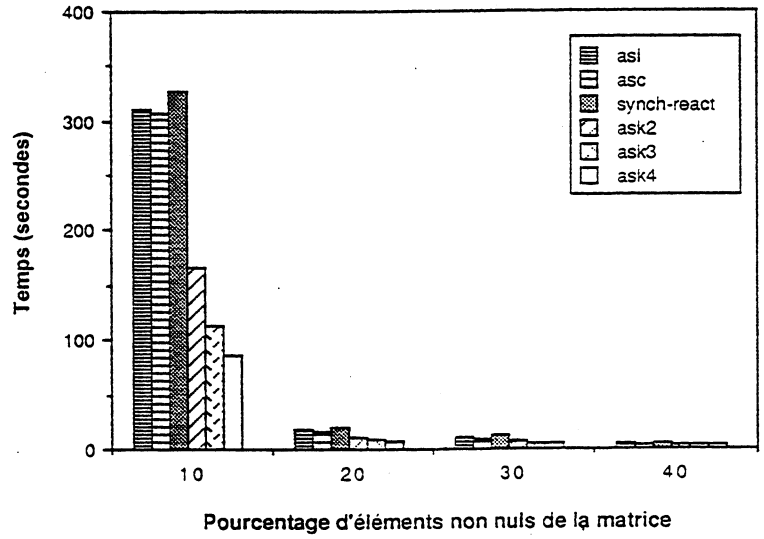


Figure 5.4: Temps de convergence pour une matrice de taille 1815 (matrices creuses)

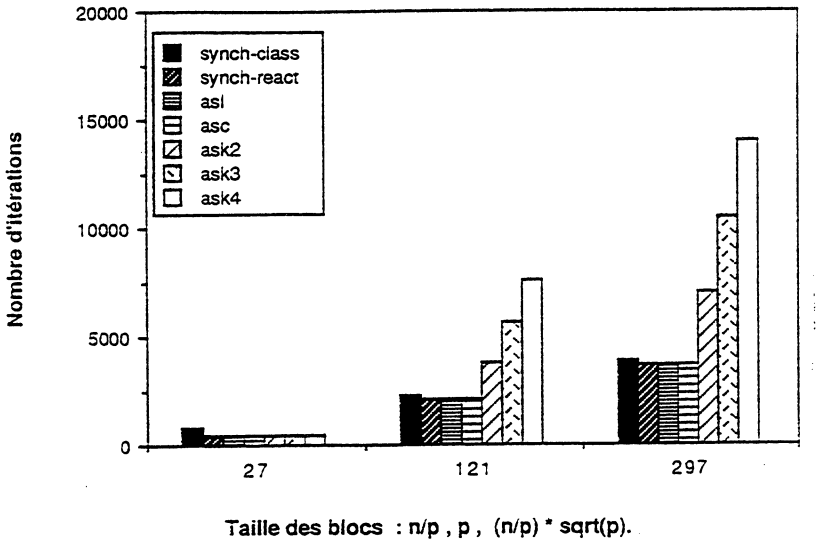


Figure 5.5: Nombre d'itération pour une matrice de taille 3267 (matrices NCD)

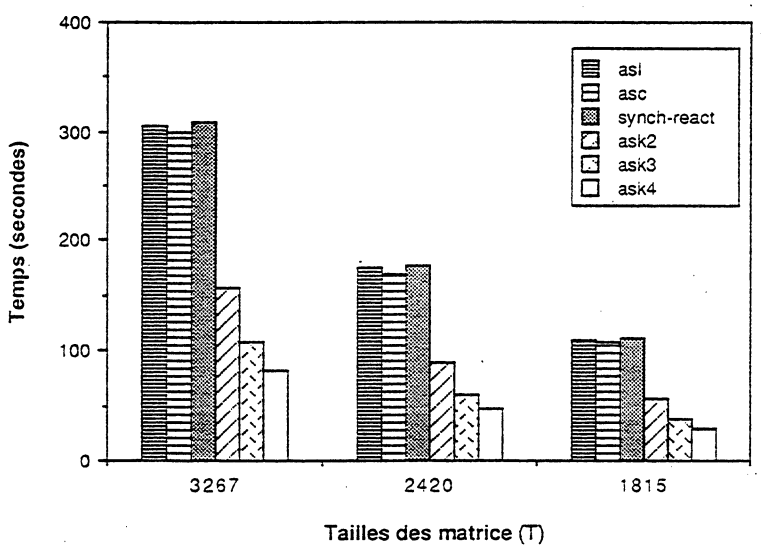


Figure 5.6: Temps de convergence pour des matrices (NCD) avec la taille des blocs $= \frac{T}{p}$

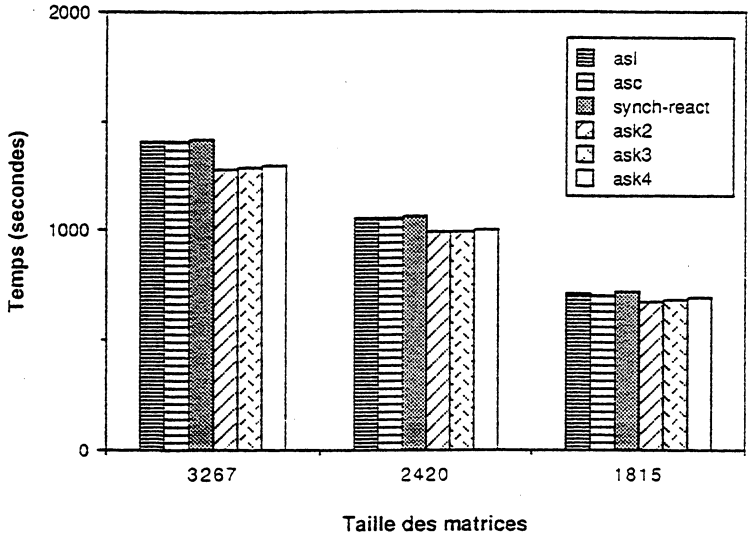


Figure 5.7: Temps de convergence pour des matrices (NCD) avec la taille des blocs = p

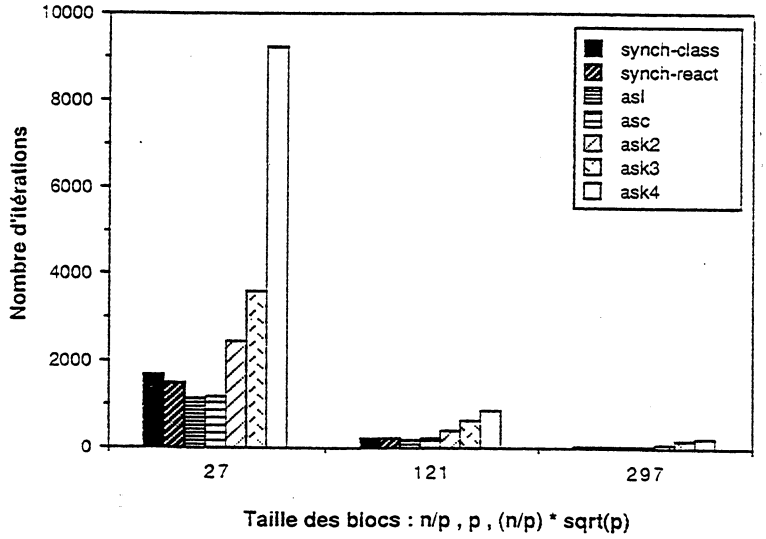


Figure 5.8: Nombre d'itération pour une matrice de taille 3267 (matrices tridiagales par blocs)

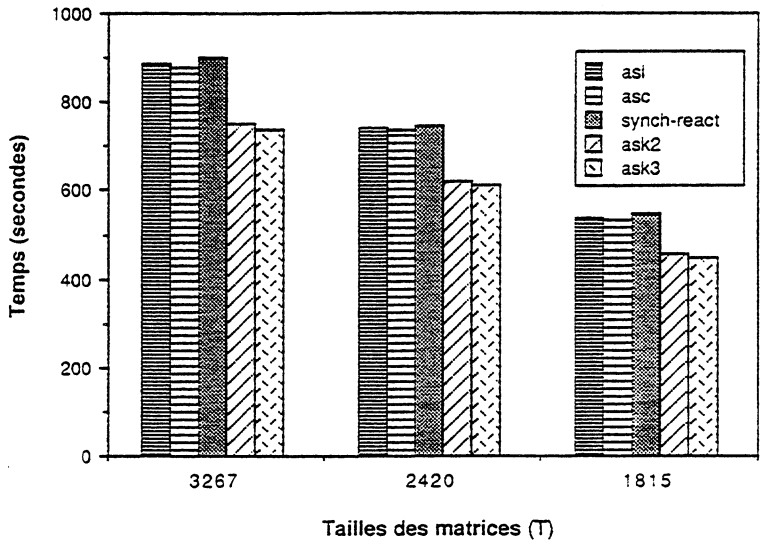


Figure 5.9: Temps de convergence pour des matrices tridiagales par blocs avec la taille des bloc = $\frac{T}{p}$

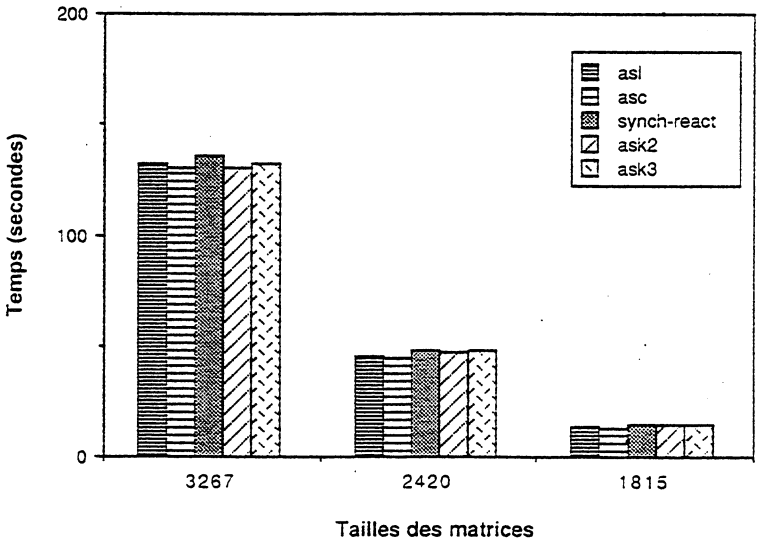


Figure 5.10: Temps de convergence pour des matrices tridiagales par blocs avec la taille des bloc = p

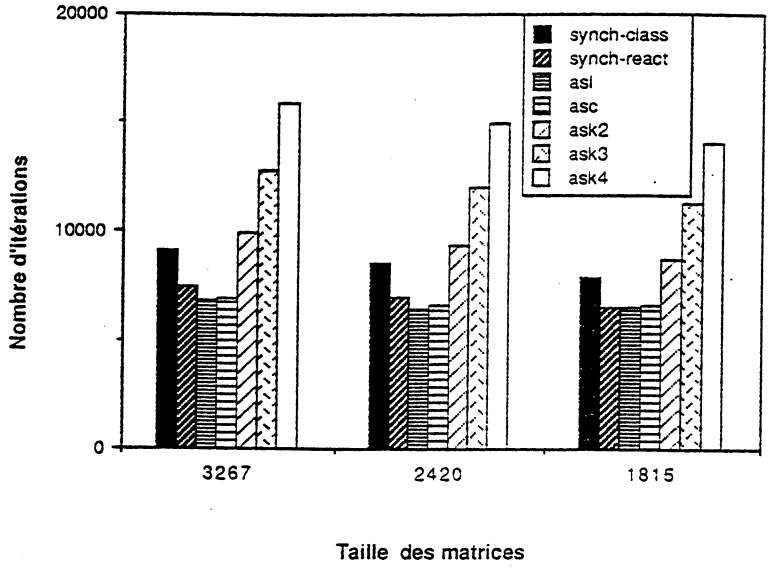


Figure 5.11: Nombre d'itération pour une matrice du type marche aléatoire sur une grille 2D

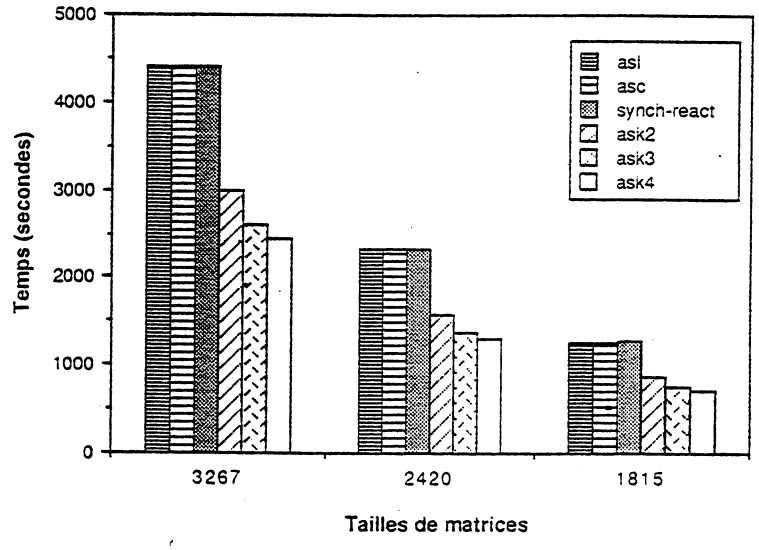


Figure 5.12: Temps de convergence pour une matrice du type marche aléatoire sur une grille 2D

Chapitre 6

Parallélisation du produit vecteur par le descripteur d'un RAS

6.1 Introduction

Dans le cadre de la parallélisation de la résolution des modèles pour l'évaluation des performances, nous allons nous intéresser aux modèles issus de la modélisation par réseaux d'automates stochastiques (RAS). La particularité de ces modèles est que leur matrice de transition ne sont pas sous une forme explicite (classique). La matrice de transition de ces modèles est donnée par un descripteur qui est une formule compacte de représentation. L'avantage de cette méthode est de pouvoir effectuer l'opération de base des méthodes itératives qui est le produit vecteur matrice, sans construire explicitement la matrice de transition. Il en résulte donc un gain en mémoire de stockage très important.

Dans ce chapitre, nous étudions la parallélisation du produit vecteur par le descripteur d'un (RAS). Pour résoudre les modèles, il suffit d'intégrer ce produit dans l'une des méthodes itératives présentées dans le chapitre 3.

Pour effectuer le produit vecteur par le descripteur d'un (RAS), l'idée est d'utiliser des résultats de l'algèbre des permutations et les produits de Kronecker. Dans le paragraphe 6.2, nous allons présenter les principes de base du produit vecteur par le descripteur, tout en insistant sur les différentes méthodes pour effectuer les permutations. Nous verrons plus loin qu'en utilisant telle ou telle méthode pour réaliser la permutation, l'implantation parallèle du produit vecteur par le descripteur est différente.

Les principes de base étant acquis, nous poursuivons notre étude et dans le paragraphe

6.3, nous allons présenter les différents niveaux de parallélisme que présente ce produit vecteur par le descripteur. Pour chaque niveau de parallélisme, nous associons une approche de parallélisme différente. Etant donnée l'expression du produit

$$\pi\ [\Sigma_{t=1}^{C}\ \ (\otimes_{i=1}^{N} M_{ti})] \tag{6.1}$$

dans le paragraphe 6.4, nous présentons l'approche qui consiste à paralléliser uniquement le calcul des termes $(\otimes_{i=1}^{N} M_{ti})$. Les C termes sont traités en séquence l'un après l'autre.

Dans le paragraphe 6.5, nous étudions une deuxième approche qui consiste à paralléliser le calcul de la somme de l'expression 6.1. Dans cette approche, le calcul des C termes $(\otimes_{i=1}^{N} M_{ti})$ s'effectue en parallèle, mais le calcul des termes n'est pas parallélisé.

Dans la dernière approche, que nous présentons au paragraphe 6.6, nous essayons de combiner les deux niveaux de parallélisme des approches précédentes.

Une comparaison entre les différentes approches de parallélisation est donnée dans le paragraphe 6.7. Cette comparaison, va nous permettre de choisir la méthode la plus appropriée à nos objectifs qui sont le traitement des gros modèle et la rapidité des calculs.

Enfin, dans le paragraphe 6.8, nous proposons des améliorations de la méthode appropriée et on termine par une conclusion.

6.2 Principe de la multiplication vecteur par le descripteur d'un (RAS)

La matrice issues des modélisations par réseaux d'automates stochastiques (RAS) est décrite par le descripteur qui à la forme suivante :

$$P\ =\ \sum_{t=1}^{C}\ [\ \otimes_{i=1}^{N} M_{ti}\ ] \tag{6.2}$$

où C représente le nombre de termes du descripteur, les matrices M_{ti} sont carrées d'ordre b_i. L'opérateur $\otimes$, représente le produit tensoriel des matrices.

Dans ce paragraphe, nous allons voir le principe de base du produit $\pi\ P$. L'idée est l'utilisation des résultats de l'algèbre des permutations et les produits de Kronecker pour éviter la construction des matrices$[\ \otimes_{i=1}^{N} M_{ti}\ ]$. Le coût mémoire de la construction de la matrice est de b^{2N} cases mémoires si on suppose que les matrice M_{ti} sont de même taille b, ce qui est prohibitif.

Pour effectuer le produit $\pi\ P$, nous allons utiliser des factorisations de l'expression 6.2. Ces factorisations se basent sur la notion de permutation des composantes du vecteur π. Le vecteur π étant initialement ordonné selon un ordre lexicographique sur les valeurs des composantes. Ces permutation correspondent à des mélanges parfaits sur les valeurs des composantes.

Dans le paragraphe 6.2.1, nous allons commencer par montrer comment sont ordonnées les valeurs des composantes dans le vecteur π.

Dans le paragraphe 6.2.2, nous donnons des définitions des mélanges parfaits en général. Puis, dans les paragraphes 6.2.3 et 6.2.4, on s'intéresse respectivement au mélange parfait dans une base de dimension 2 et 3 . Ces deux types de mélanges parfaits correspondent à deux factorisations de l'expression 6.2. Nous verrons plus loin que selon que l'on utilise telle ou telle factorisation de l'expression 6.2, l'implantation parallèle du calcul de $\pi\ P$ est fondamentalement différente.

Dans les paragraphes 6.2.5 et 6.2.6, nous présentons les méthodes de calcul de $\pi\ P$ en utilisant respectivement deux factorisations de l'expression 6.2.

6.2.1 Ordre des composantes dans le vecteur π

Soit S un système de variables aléatoires $S = (\ X_1\ ,\ X_2\ ,X_3\ \ ,...,X_N\ )$. Chaque variable aléatoire X_i représente le comportement d'un sous-système qui est décrit par une matrice M_i de taille b_i. L'ensemble des valeurs de chaque variable aléatoire X_i est $\{1,\ ...,b_i\}$. L'état global du système S est décrit par un vecteur de probabilités d'états π de taille T, avec $T = \prod_{i=1}^{N}\ (b_i)$. Une position dans le vecteur de probabilités d'états π, notée π_e, représente la probabilité que le système soit dans l'états e :

$$\pi_e\ =\ Prob(X_1 = e_1,\ X_2 = e_2,\ X_3 = e_3,...,\ X_N = e_N)$$

avec

$$e\ =\ (e_1,\ e_2,\ e_3,...,\ e_N).$$

Chaque variable aléatoire X_i prend plusieurs valeurs, les composantes du vecteur de probabilités π sont ordonnées selon un ordre lexicographique sur e.

Exemple :

Soit un système à trois variables aléatoires $(X_1,\ X_2,\ X_3)$, avec $N = 3$, les matrices $M_i\ , i = 1..3$, sont de tailles : $b_1 = 2,\ b_2 = 3,\ b_3 = 2$. Le vecteur π sera organisé comme suit :

π_1 = Prob(X_1= 1, X_2= 1, X_3= 1)
π_2 = Prob(X_1= 1, X_2= 1, X_3= 2)
π_3 = Prob(X_1= 1, X_2= 2, X_3= 1)
π_4 = Prob(X_1= 1, X_2= 2, X_3= 2)
π_5 = Prob(X_1= 1, X_2= 3, X_3= 1)
π_6 = Prob(X_1= 1, X_2= 3, X_3= 2)
π_7 = Prob(X_1= 2, X_2= 1, X_3= 1)
π_8 = Prob(X_1= 2, X_2= 1, X_3= 2)
π_9 = Prob(X_1= 2, X_2= 2, X_3= 1)
π_{10}= Prob(X_1= 2, X_2= 2, X_3= 2)
π_{11}= Prob(X_1= 2, X_2= 3, X_3= 1)
π_{12}= Prob(X_1= 2, X_2= 3, X_3= 2)

Avant de montrer l'effet du mélange parfait sur l'ordre des composantes du vecteur π, nous allons commencer par définir cette notion de mélange parfait.

6.2.2 Définitions des mélanges parfaits

- **Mélange parfait :**

  Soit Γ un ensemble de mots de N lettres, la i^{eme} lettre appartient à un alphabet

  $$\alpha = \{0,1\}.$$

  On appelle mélange parfait une application $\sigma : \Gamma \longmapsto \Gamma$ qui à un mot w de Γ, w = (w_1 , w_2 ,w_3 ,...,w_N) associe un mot w' construit à partir de w en plaçant la première lettre de w en dernière position w' = (w_2 ,w_3 ,...,w_N,w_1).

- **Mélange parfait généralisé :**

  Dans la définition des mélanges parfaits généralisés, une flexibilité est introduite au niveau de l'alphabet de chaque lettre du mot :

  Chaque lettre du mot w_i prend ses valeurs dans un ensemble d'entier

  $$\alpha_i = \{ 1,...,b_i \}.$$

  L'alphabet est défini comme l'ensemble produit cartésien $\prod_{i=1}^{N} \alpha_i$.

Dans ce qui suit nous allons voir l'effet des mélanges parfaits sur l'ordre des composantes du vecteur π.

6.2.3 Mélange parfait des composantes dans une base de dimension 2

Le problème probabiliste X=(X_1 , X_2 , X_3 ,..., X_N) à N dimensions peut être vu comme un problème à 2 dimensions en regroupant dans l'ordre lexicographique les $(N-1)$ composantes de X d'un côté et une composante X_i quelconque de l'autre, soit une permutation circulaire à droite : $((X_{i+1}, ..., X_N, X_1, ..., X_{i-1}), X_i)$.

Définition des mélanges parfaits dans une base de dimension 2

Soit la base $bb_i = (d_1, d_2)$ sur l'ensemble $E = [1..T]$ qui représente les positions des composantes dans le vecteur π. La base bb_i est définie comme suit :

$$d_1 = b_i,$$

$$d_2 = B_i = \prod_{j=1, j \neq i}^{N} b_j$$

Ainsi, on décompose l'ensemble E de deux manières :

- d_1 blocs de d_2 éléments, l'élément ki = (i_1 , i_2) est défini comme étant l'élément de rang i_2 dans le bloc de numéro i_1. Son rang absolu est :

$$ki = (i_1 - 1)\, d_2 + i_2 \quad , \; ki = 1..T, \; i_2 = 1..d_2, \; i_1 = 1..d_1.$$

- d_2 blocs de d_1 éléments, l'élément kj =(j_1 , j_2) est défini comme étant l'élément de rang j_2 dans le bloc de numéro j_1, son rang absolu est :

$$kj = (j_1 - 1)\, d_1 + j_2, \; kj = 1..T, \; j_2 = 1..d_1, \; j_1 = 1..d_2.$$

Un mélange parfait des éléments de E dans la base bb_i fait passer la composante π_{kj} du vecteur π avec kj qui s'écrit (j_1, j_2) dans la base $bb_i = (d_1, d_2)$, à la position kj^{σ_2} qui s'écrit (j_2, j_1) dans la base $bb_i^{\sigma_2} = (d_2, d_1)$.

En d'autre terme, une (d_1 , d_2)-permutation dans E est une application notée σ_2 (d_1, d_2), telle que :

σ_2 $(d_1$, d_2): ki =(i_1 -1) $d_2 + i_2 \longmapsto kj = (i_2$ -1) $d_1 + i_1$.
kj est l'image de ki par l'application: σ_2 $(d_1$,d_2)[ki] = kj.

Exemple :

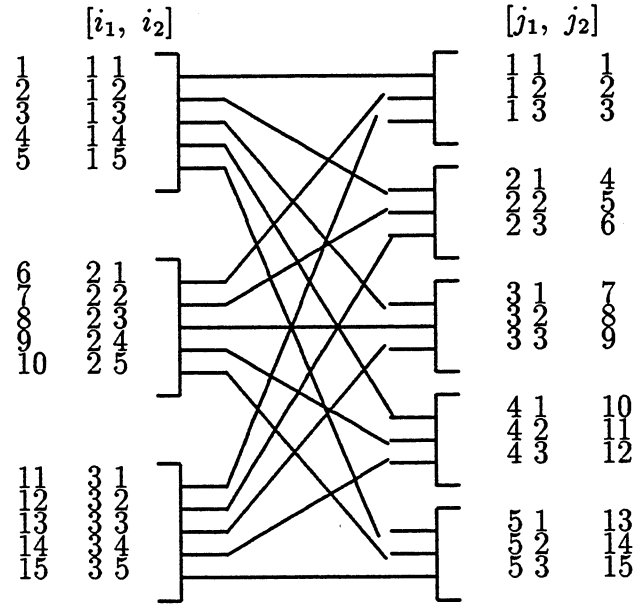


Figure 6.1: Schéma de la (3,5)-permutation

Si on reprend l'exemple précédent, on suppose qu'on effectue le mélange parfait dans base bb_1 :

$$d_1 = b_1 = 2 ,$$

$$d_2 = B_1 = \frac{T}{b_1} = 6$$

L'élément π_{k_e} avec $k_e = 11$ s'écrit $(2, 5)$ dans la base $bb = (b_1, B_1)$:

$$k_e = (i_1 - 1)\, d_2 + i_2 = (2 - 1) \times 6 + 5 = 11$$

$$k_e{}^{\sigma_2} = (i_2 - 1)\, d_1 + i_1 = (5 - 1) \times 2 + 2 = 10$$

Schématisation du mélange parfait dans une base de dimension 2

Pour effectuer une $\sigma_2(b_i, B_i)$ permutation, le vecteur π est vue comme b_i blocs de taille B_i, le premier bloc contient les composantes $\pi_1, \pi_2, ..., \pi_{B_i}$ le bloc 2 comprend les B_i éléments suivant et ainsi de suite, le dernier bloc comprend les composantes $(b_i - 1)B_i + 1, ..., T = b_i B_i$ (voir la figure 6.2) .

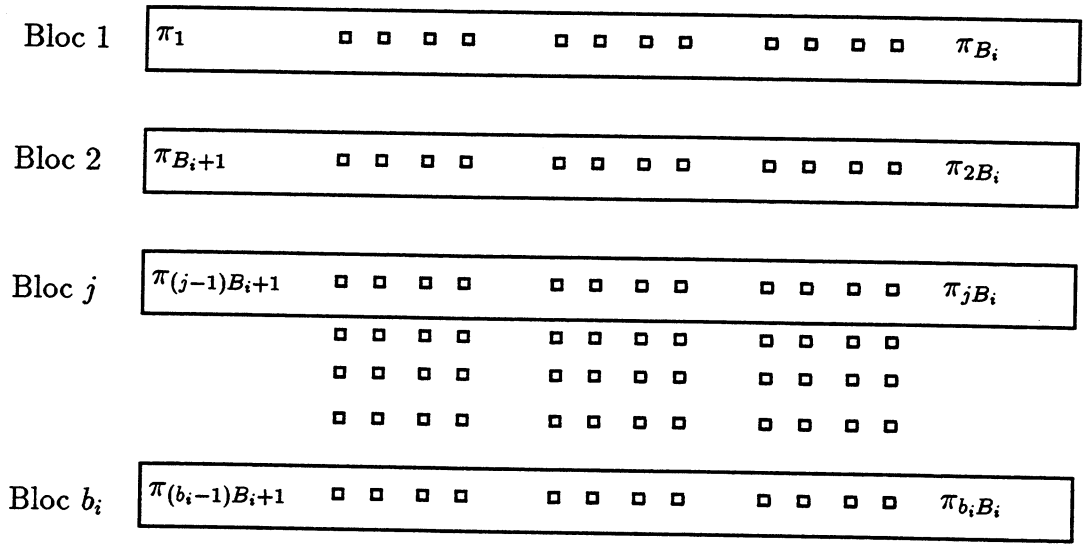


Figure 6.2: Le vecteur π vue comme b_i blocs de B_i éléments

Avant la permutation, les composantes de π sont rangées dans le vecteur π selon l'ordre ligne par ligne (bloc après bloc) (figure 6.2). Après la permutation, les éléments du vecteur π^{σ_2} sont rangées selon l'ordre colonne par colonne (figure 6.3)

Propriétés des mélanges parfaits dans une base de dimension 2

On rappelle que le mélange parfait modifie l'ordre des composantes du vecteur π. En effet, à une position k_d d'une composante du vecteur π, on associe la position k_a=$\sigma_2(k_d)$, qui sera la position de cette composante dans le vecteur π^{σ_2} après le mélange parfait.

Pour mieux comprendre le déplacement des composantes du vecteur π pour former le vecteur π^{σ_2}, nous allons formaliser le problème comme suit :

Soit p un entier qui divise la taille du vecteur T. On décompose les vecteurs π et π^{σ_2} en p sous-vecteurs de même taille :

$\pi = (s\pi_0, s\pi_1, ..., s\pi_{p-1})$, avec le sous-vecteur $s\pi_0 = (\pi_1 , \pi_2 , ... , \pi_{(\frac{T}{p})})$, le sous-vecteur $s\pi_1 = (\pi_{(\frac{T}{p}+1)} ..., \pi_{(\frac{2T}{p})})$ et ainsi de suite jusqu'au dernier sous-vecteur. On fait de même pour le vecteur π^{σ_2}, $\pi^{\sigma_2} = (s\pi_0^{\sigma_2}, s\pi_1^{\sigma_2}, ..., s\pi_{p-1}^{\sigma_2})$.

A chaque sous-vecteur $s\pi_j$, on lui associe une place p_j, $j = 0..p-1$. On dit alors que le sous-vecteur $s\pi_j$ est placé en p_j. On suppose aussi que les deux sous-vecteurs $s\pi_j$ et

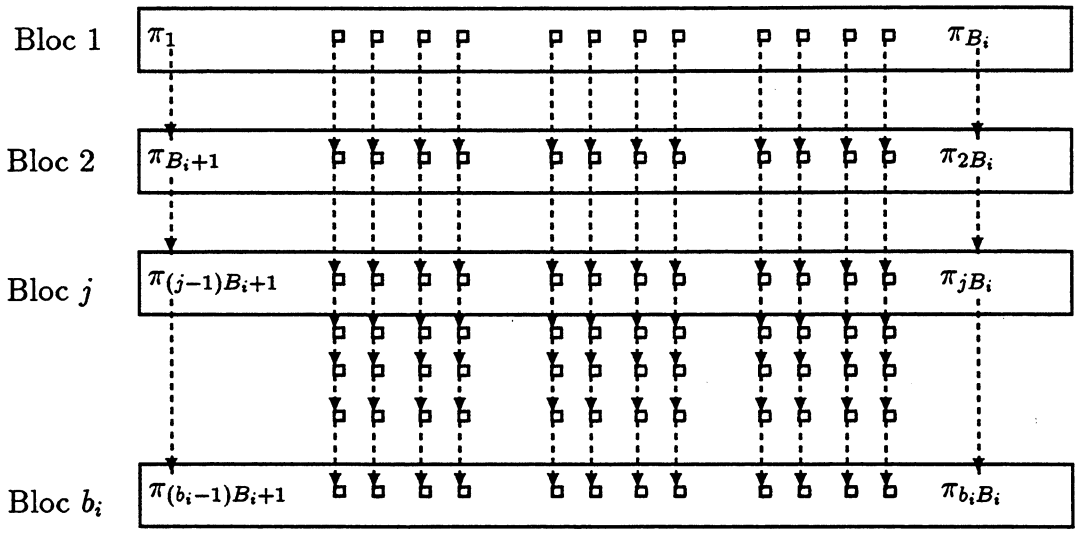


Figure 6.3: Rangement des éléments du vecteur π^{σ_2}

$s\pi_j^{\sigma 2}$ sont dans la même place p_j.

Revenons au déplacement des composantes de π par la permutation σ_2. On rappelle que dans la base $bb_i = (b_i, B_i)$, le vecteur π est vu comme b_i blocs de B_i éléments. Une position dans le vecteur est repérée par le couple d'entier (i_1, i_2), où i_1 désigne le numéro de bloc et i_2 la position dans le bloc considéré. A une position k_d donnée :

$$k_d = (i_2 - 1)\, B_i + i_1 \tag{6.3}$$

on associe la position :

$$k_a = (\, i_1 \text{ - } 1)\, b_i + i_2$$

avec :

$$1 \leq i_2 \leq b_i\,, \quad 1 \leq i_1 \leq B_i$$

D'après ce qu'on vient de voir : une position k_j d'une composante du vecteur π qui est placée en p_j, peut s'écrire :

$$k_j = p_j \frac{T}{p} + m_j, \quad 1 \leq m_j \leq \frac{T}{p}. \tag{6.4}$$

Ce qu'on veut, c'est trouver une relation entre la place p_d du sous-vecteur $s\pi_d$ qui contient la position k_d et la place p_a du sous-vecteur $s\pi_a^{\sigma 2}$ qui contient la position $k_a = \sigma_2\,(\,k_d\,)$.

Théorème 3 *Sous les hypothèses suivantes :*

- *T est divisible par p.*
- *B_i est divisible par p.*

On à la relation suivante

$$p_a = (\, p_d\, b_i + c\,)\,[p] \; Avec \; 0 \leq c < b_i \tag{6.5}$$

Preuve :

D'après 6.3 et 6.4, on a le système :

$$(S_1)\ \begin{cases} (i_2 - 1)\, B_i + i_1 = p_d \frac{T}{p} + m_d. \\ (i_1 - 1)\, b_i + i_2 = p_a \frac{T}{p} + m_a. \end{cases}$$

Avec m_d, $m_a \in [1, \frac{T}{p}]$ et $1 \leq i_2 \leq b_i$ et $1 \leq i_1 \leq B_i$.

Considérons le vecteur π^{σ_2} comme une suite de sous-vecteurs de composantes de tailles b_i . Comme chaque place contient $\frac{B_i}{p} = \frac{T}{b_i\, p}$ de ces sous-vecteurs, on a donc la relation suivante :

$$i_1 = p_a \frac{T}{b_i\, p} + l \quad , l = 1 \ldots \frac{T}{b_i\, p} \tag{6.6}$$

On remplace i_1 par sa valeur dans la deuxième équation du système (S_1) :

$$(p_a \frac{T}{b_i\, p} + l - 1\ )\, b_i + i_2 = p_a \frac{T}{p} + m_a. \tag{6.7}$$

Après développement, on obtient :

$$(p_a \frac{T}{p} + l\, b_i - b_i\ ) + i_2 = p_a \frac{T}{p} + m_a. \tag{6.8}$$

On tire i_2 de 6.8 et on obtient la relation :

$$i_2 = m_a + b_i - l\, b_i. \tag{6.9}$$

Reportons 6.6 et 6.9 dans la première équation du système (S_1), on obtient :

$$(\ m_a + b_i - l\, b_i - 1\ )\, B_i + p_a \frac{T}{b_i\, p} + l = p_d \frac{T}{p} + m_d. \tag{6.10}$$

En développant cette équation, on obtient :

$$m_a\, B_i + T - l\, T - B_i + p_a \frac{T}{b_i\, p} + l = p_d \frac{T}{p} + m_d. \tag{6.11}$$

En multipliant les deux membres de l équation 6.11 par b_i :

$$m_a\, T + T\, b_i - l\, T\, b_i - T + p_a \frac{T}{p} + l\, b_i = p_d \frac{T\, b_i}{p} + m_d\, b_i. \tag{6.12}$$

Divisons les deux membres de cette équation par T :

$$m_a + b_i - l\, b_i - 1 + \frac{p_a}{p} + l\,\frac{b_i}{T} = p_d\,\frac{b_i}{p} + \frac{m_d\, b_i}{T}. \tag{6.13}$$

Remplaçons $\frac{T}{b_i\, p} = k$ dans l'équation 6.13, on obtient :

$$m_a + b_i - l\, b_i - 1 + \frac{p_a}{p} + \frac{l}{k\, p} = p_d\,\frac{b_i}{p} + \frac{m_d}{k\, p}. \tag{6.14}$$

Arrangeons les termes de cette équation :

$$p_d\,\frac{b_i}{p} + \frac{m_d}{k\, p} - \frac{l}{k\, p} - (\,-1 - l\, b_i + m_a + b_i\,) = \frac{p_a}{p}. \tag{6.15}$$

Multiplions l'équation 6.15 par p :

$$p_d\, b_i + \frac{m_d}{k} - \frac{l}{k} - p\,(\,-1 - l\, b_i + m_a + b_i\,) = p_a. \tag{6.16}$$

On peut écrire autrement cette équation en utilisant l'opérateur modulo [ ], finalement on obtient la relation qui lie p_a avec p_d :

$$p_a = (p_d\, b_i + (\frac{m_d - l}{k})) \,[p]. \tag{6.17}$$

pour prouver que la relation 6.5 est vrai, il suffit de montrer que l'entier $\frac{m_d - l}{k}$ est compris entre 0 et b_i. En effet on a :

$$1 \leq m_d \leq \frac{T}{p} \tag{6.18}$$

$$1 \leq l \leq \frac{T}{p\, b_i}$$

On multiplie cette dernière inéquation par (-1), on obtient :

$$-\frac{T}{p\, b_i} \leq -l \leq -1 \tag{6.19}$$

On ajoute membre à membre les deux inégalités 6.18 et 6.19 :

$$1 - \frac{T}{p\, b_i} \leq m_d - l \leq \frac{T}{p} - 1 \tag{6.20}$$

Multiplions cette inégalité par $\frac{1}{k} = \frac{b_i\ p}{T}$:

$$\frac{b_i\ p}{T}(1 - \frac{T}{p\ b_i}) \leq \frac{m_d\ -\ l}{k} \leq \frac{b_i\ p}{T}(\frac{T}{p}\ -\ 1) \tag{6.21}$$

$$\frac{b_i\ p}{T}\ -\ 1 \leq \frac{m_d\ -\ l}{k} \leq b_i\ -\ \frac{b_i\ p}{T}. \tag{6.22}$$

Comme $\frac{b_i\ p}{T} < 1$ alors, on déduit que :

$0 \leq \frac{m_d\ -\ l}{k} < b_i$, ce qui achève la démonstration.

Autre vision de la réalité exprimée par le théorème

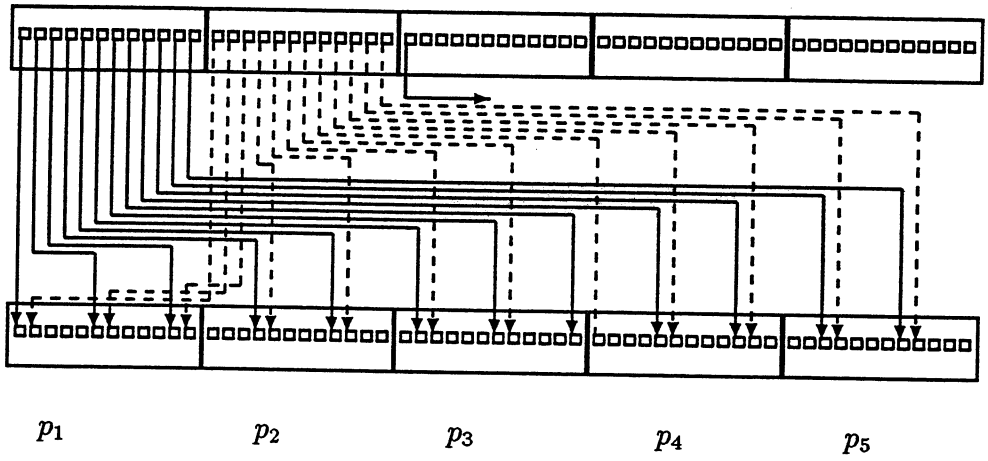
D'après le théorème 3, on déduit que les composantes du vecteur π qui sont à la place p_d vont migrer sur b_i places différentes après le mélange parfait. Par la suite, nous allons voir en réalité que le sous-vecteur placé en p_d est constitué de groupes de composantes consécutives et que chaque groupe va migrer sur une place différente. De plus, les numéros des places que vont occuper ces groupes sont consécutifs.

On rappelle que le mélange parfait $S(b_i,\ B_i)$ voit le vecteur comme b_i blocs de B_i éléments. Avant le mélange parfait, l'ordre des composantes s'obtient par un parcours ligne par ligne des b_i blocs. Après le mélange parfait, l'ordre des composantes s'obtient par un parcours colonne par colonne des b_i blocs. De ce fait en déduit les propriétés suivantes :

1. Si deux composantes sont consécutives avant le mélange parfait, elle seront consécutives à un pas b_i *modulo* T ([T]) après le mélange parfait (conséquence du parcours colonne par colonne sachant qu'il y a b_i lignes).
2. Si deux composantes sont consécutives après le mélange parfait, elles étaient consécutives à un pas B_i $[T]$ avant le mélange parfait.

Observons le déplacement des composantes avant et après le mélange parfait (vue linéaire, figure 6.4) :

On commence par la première composante qui va occuper la même position dans le vecteur après le mélange. La composante qui occupait la position 2 avant le mélange va à la position 1 plus b_i dans le vecteur après le mélange parfait. La composante qui était à la position 3 avant le mélange parfait, va passer à la position de la composante 2 après le mélanges parfait plus b_i et ainsi de suite, la composante suivante va passer à la position de la précédente après le mélange parfait plus b_i $[T]$.



On note que par exemple : π_{11} et $\pi_{12} \quad \in g_{p_1,\, p_5}$

⟶ déplacement des composantes par pas de $b_i = 5$ après la permutation $S(5, 12)$

Figure 6.4: Vue linéaire du déplacement des composantes après le mélange parfait (modèle : $b_1 = 5,\ b_2 = 4,\ b_3 = 3$)

Remarque

D'après ce qu'on vient de voir : si un certain nombre de composantes consécutives situées à une place donnée p_d vont passer à la place p_a et si la composante suivante ne va pas à la place p_a alors forcément elle va à la place $p_{(a+1)[p]}$ après le mélange parfait.

Dans le paragraphe qui suit, nous allons voir comment sont localisées les composantes (groupe) qui vont migrer après mélange parfait sur une place donnée.

Notion de groupe de composantes consécutives (gcc)

Soit x_1 une composante appartenant à la place p_d et $x_1^{\sigma_2}$ la position de cette composante après le mélange parfait. Les composantes consécutives de x_1 notés x_{1+j} vont être localisées par $x^{\sigma_2}_{(1+j.b_i)[T]} \in p_a$ après le mélange parfait. Si la composante $x_{(1+j+1)} \in p_d$ avant le mélange parfait va aller à la place p_{a+1} , on dit que les composantes consécutives x_1, ..., x_j forme un groupe qu'on note g_{da}, ce groupe est constitué de composante qui vont aller à la place p_a après le mélange parfait. La composante $x_{(1+j+1)}$ va appartenir au groupe $g_{d(a+1)}$. Toute fois, il est intéressant d'avoir une idée sur la taille moyen du groupe (gcc), nous verrons son utilité plus loin dans le paragraphe 6.8.1.

D'après le théorème 3 et ce qu'on vient de voir, on déduit qu'un sous-vecteur donné $s\pi_d$ placé en p_d est former de b_i groupes (gcc). Etant donné que la taille du sous-vecteur est de $\frac{T}{p}$, la taille d'un groupe (gcc) est donc $\frac{T}{p\ b_i}$.

6.2.4 Mélange parfait des composantes dans une base de dimension 3

Le problème probabiliste X=(X_1 , X_2 , X_3 ,..., X_N) à N dimensions peut être vu comme un problème à trois dimensions en regroupant dans l'ordre lexicographique les $(i-1)$ premières composantes de X d'une part la composante X_i et les $(N-i-1)$ composantes d'autre part, et cela pour tout $i \in [\,1..N\,]$, soit $((X_1, \ldots, X_{i-1}), X_i, (X_{i+1}, \ldots, X_N))$.

Définition du mélange parfait dans une base de dimension 3

Sur l'ensemble E, on construit une base $bbb_i = (d_1, d_2, d_3)$ avec

$$d_1 = \prod_{j=1}^{i-1} b_j,$$

$$d_2 = b_i,$$

$$d_3 = \prod_{j=i+1}^{N} b_j$$

On définit L_3 comme l'ensemble produit cartésien $\prod_{k=1}^{3}$ [$1..b_k$] ordonné lexicographiquement. Soit $f = (f_1, f_2, f_3) \in L_3$, un mélange parfait des éléments de E dans la base bbb_i, fait passer la composante π_{k_j} du vecteur π, tel que k_j s'écrit (f_1, f_2, f_3) dans la base bbb_i :

$$k_j = (f_1 - 1)\, d_3 d_2 + (f_2 - 1)\, d_2 + f_3$$

à la position $k_j^{\sigma_3}$ tel que $k_j^{\sigma_3}$ s'écrit $f = (f_3, f_1, f_2)$ dans la base $bbb_i^{\sigma_3} = (d_3, d_1, d_2)$ avec :

$$k_j^{\sigma_3} = (f_3 - 1)\, d_2 d_1 + (f_1 - 1)\, d_2 + f_2$$

Exemple

Revenant à l'exemple précédent à la phase $i = 1$ du mélange parfait : $bbb = (1, 2, 6)$

$$f = (1, 2, 5)$$
$$k_j = 11 = (1 - 1)\, 2 \times 6 + (2 - 1)\, 2 + 5$$
$$k_j^{\sigma} = 10 = (5 - 1)\, 2 \times 1 + (1 - 1)\, 2 + 2$$

Schématisation des mélanges parfaits dans une base de dimension 3

Avant le mélange parfait dans la base bbb_i, le vecteur π est vu comme

$$d_1 = \prod_{j=1}^{i-1} b_j$$

sous-blocs consécutifs. Chaque sous-blocs contient $d_2 = b_i$ colonnes consécutives de taille

$$d_3 = \prod_{j=i+1}^{N} b_j.$$

Un élément du vecteur π_{k_f} du vecteur π est repéré par le triplet (f_1, f_2, f_3) (voir la figure 6.5).

- f_1 indique le numéro du sous-bloc contenant π_{k_f},
- f_2 indique le numéro de la colonne contenant π_{k_f} dans le sous-bloc f_1,
- f_3 indique la position de π_{k_f} dans la colonne f_2 du sous-bloc f_1.

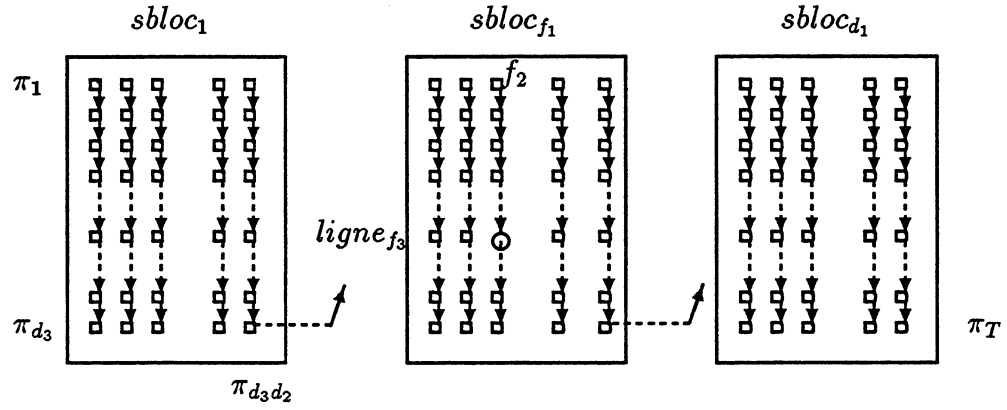


Figure 6.5: Structure en blocs du vecteur π avant le mélange parfait à la phase i

Avant la permutation, le vecteur π est ordonné comme suit :

$sbloc_1$, $sbloc_2$, ..., $sbloc_{d_1}$. L'ordre des composantes dans le $sbloc_j$ est :
Les éléments de la $colonne_1$, $colonne_2$, ..., $colonne_{d_2}$ (figure 6.5). Le vecteur π^{σ_2} est arrangé comme suit :

$ligne_1$ du sous-bloc $sbloc_1$, $ligne_1$ du sous-bloc $sbloc_2$,...,$ligne_1$ du sous-bloc $sbloc_{d_1}$. Ensuit, la $ligne_2$ du bloc $sbloc_1$, $ligne_2$ du sous-bloc $sbloc_2$,..., $ligne_2$ du sous-bloc $sbloc_{d_1}$ et ainsi de suite jusqu'à la $ligne_{d_3}$ du sous-bloc $sbloc_{d_1}$.

6.2.5 Calcul de π [$\otimes_{i=1}^{N} M_{ti}$] en utilisant les mélanges parfaits dans une base de dimension 2

Dans ce paragraphe nous allons voir une méthode de factorisation pour calculer le produit π [$\otimes_{i=1}^{N} M_{ti}$]. Cette méthode est basé sur les mélanges parfaits dans une base de dimension 2. La factorisation est donnée par le théorème suivant.

Théorème 4 $(\otimes_{i=1}^{N} M_i) = \prod_{i=1}^{N} [S_{b_i B_i} (Id_{B_i} \otimes M_i)]$

La preuve de ce théorème est donnée dans [13].

Les opérateurs $\otimes$ et $\prod$ représentent respectivement le produit tensoriel des matrices et le produit usuel des matrices.

On note b_i la taille de la matrice M_i et $B_i = \prod_{j=1 j \neq i}^{N} b_j$.

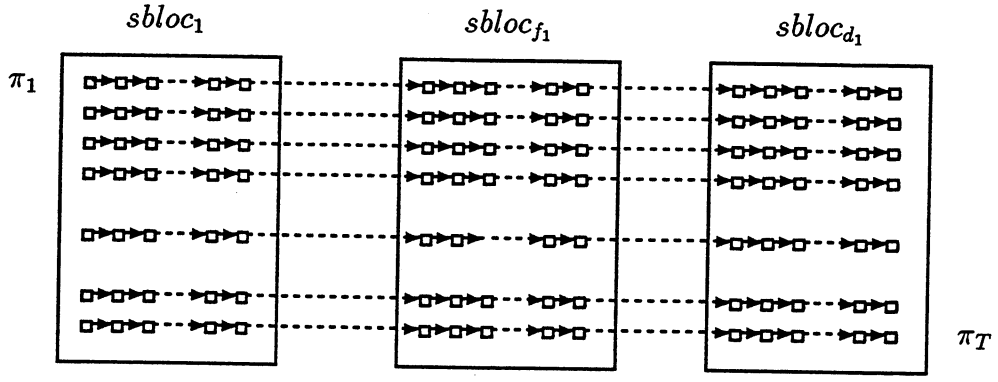


Figure 6.6: Vision du vecteur $\pi^{\sigma_3^{(i)}}$ après le mélange parfait dans la base bbb_i

$S_{b_iB_i}$ est la matrice de la $(b_i\ ,B_j\ )$-permutation et est définie par :

$S_{b_iB_i}\ (\ k_i\ ,\ k_j\ ) = 1$ ssi $\sigma_2\ (b_i\ ,\ B_i\ )[\ k_i\ ] = k_j$

Pour réaliser le produit $\pi\ [\ \otimes_{i=1}^{N} M_{ti}\ ]$, en utilisant cette factorisation, on effectue N phases. Chacune des phases i, $i = 1..N$, se décompose en deux étapes :

- Mélange parfait des composantes du vecteur π dans la base bb_i qui correspond à la permutation σ_2 (la matrice $S_{b_iB_i}$).
- Calcul des produits du vecteur π^{σ_2} par la matrice bloc diagonal $(Id_{B_i} \otimes\ M_i\ )$.

Exemple pour illustrer cette méthode

Soit un vecteur initial π donné, on veut calculer le produit :

$$\pi(\ M_1\ \otimes\ M_2)$$

M_1 , M_2 sont des matrices de tailles respectivement 2, 3 . Par le théorème précédent on a :

$$\pi\ (\ M_1\ \otimes\ M_2)\ =\ \pi\ [S_{23}\ (Id_3\ \otimes\ M_1)][S_{32}\ (Id_2\ \otimes\ M_2)].$$

Avec

$$\pi = (\pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6).$$

Après la (2, 3)-permutation sur le vecteur π on obtient :

$$\pi\ S_{23} = (\pi_1\ ,\ \pi_4\ ,\pi_2\ ,\pi_5\ ,\ \pi_3\ ,\ \pi_6\ ).$$

et

$$Id_3 \otimes M_1 = \begin{pmatrix} M_1 & 0 & 0 \\ 0 & M_1 & 0 \\ 0 & 0 & M_1 \end{pmatrix}$$

On voit que pour calculer π ($S_{23}\ Id_3 \otimes M_1$), on n'effectue que les produits $(\pi_1\ ,\ \pi_4\ )\ M_1$, (π_2 ,π_5) M_1 , $(\pi_3\ ,\ \pi_6\ )\ M_1$. On fait de même pour le deuxième facteur, on effectue d'abord la permutation S_{32}, puis le produit par ($Id_2 \otimes M_2$).

Remarque :

Dans cette méthode, le mélange parfait déplace les composantes du vecteur π. Dans la méthode que nous allons présenter par la suite qui utilise les mélanges parfaits dans une base de dimension 3, les composantes du vecteur π ne change pas de position.

6.2.6 Calcul de π [$\otimes_{i=1}^{N} M_{ti}$] en utilisant les mélanges parfaits dans une base de dimension 3

Dans ce paragraphe, nous allons utiliser une deuxième factorisation de π [$\otimes_{i=1}^{N} M_{ti}$]. Cette factorisation se base sur les mélanges parfaits dans une base de dimension 3. Elle est illustrée par le théorème suivant.

Théorème 5 π [$\otimes_{i=1}^{N}\ M_i$] = π $\prod_{i=1}^{N}$ [$S_{\sigma_3^{(i)}}$ ($Id_{B_i} \otimes M_i$) $S^*_{\sigma_3^{(i)}}$]

L'opération (*) dénote la transposition de matrice. De même, la preuve de ce théorème est donnée dans [13] .

Les matrices de permutations $S_{\sigma_3^{(i)}}$, $i \in$ [1..N] correspondent au mélanges parfaits dans la base $bbb_i = (d_1,\ d_2,\ d_3)$ définit précédemment :

$$S_{\sigma_3^{(i)}}(k_i,\ k_j)\ =\ 1\ \ ssi\ \ k_j\ =\ \sigma_3^{(i)}(k_i)$$

$\sigma_3^{(i)}$: $E \longrightarrow E$ qui transforme un élément de k_i en k_j tel que :

$$k_i\ =\ (f_1\ -\ 1)\ d_3d_2\ +\ (f_2\ -\ 1)\ d_2\ +\ f_3$$

$$k_j\ =\ (f_3\ -\ 1)\ d_2d_1\ +\ (f_1\ -\ 1)\ d_2\ +\ f_2$$

Pour réaliser le produit $\pi\ [\ \otimes_{i=1}^{N} M_{ti}\ ]$ en utilisant cette factorisation, on effectue N phases. Chacune des phases i, $i = 1..N$, se décompose en trois étapes :

- Mélange parfait des composantes du vecteur π dans la base bbb_i qui correspond à la permutation σ_3 (la matrice $S_{\sigma_3^{(i)}}$).
- Calcul des produits du vecteur π^{σ_3} par la matrice bloc diagonal $(Id_{B_i} \otimes\ M_i\ )$.
- Remise des composantes selon l'ordre initial en utilisant la permutation inverse σ_3^{-1} $(S^*_{\sigma_3^{(i)}})$.

Dans [2], on donne une vision sous forme d'une arborescence du schéma des permutations successives pour effectuer le produit $\pi\ [\ \otimes_{i=1}^{N} M_{ti}\ ]$ en se basant sur le théorème 5

Propriété des mélanges parfaits successifs

Pour effectuer le mélange parfait à la phase i, $(S_{\sigma_3^{(i)}})$, le vecteur π est vu comme un bloc BL_i constitué de $d_1^{(i)}$ sous-bloc de $d_2^{(i)}\ =\ b_i$ colonnes de taille $d_3^{(i)}\ =\ \prod_{k=i+1}^{N}\ b_k$ (voir figure 6.5). Si on effectue effectivement la permutation $S_{\sigma_3^{(i)}}$, on obtient un vecteur $\pi^{\sigma_3^{(i)}}$ dont l'ordre des composantes correspond un parcours ligne par ligne (figure 6.7). Plus précisément le vecteur $\pi^{\sigma_3^{(i)}}$ est constitué de $d_1^{(i)}\ \times\ d_3^{(i)}$ sous-vecteurs de taille b_i, ces derniers sont ordonnés comme suit :

Le premier sous-vecteur correspond à la ligne 1 du sous-bloc $sbloc_1$,..., le sous-vecteur $d_1^{(i)}$ correspond à la ligne 1 du sous-bloc $d_1^{(i)}$. Le sous-vecteur $d_1^{(i)} + 1$ correspond à la ligne 2 du sous-bloc $sbloc_1$,..., le sous-vecteur $2d_1^{(i)}$ correspond à la ligne 2 du sous-bloc $d_1^{(i)}$. Ainsi de suite jusqu'au sous-vecteur $(d_3^{(i)} - 1)\ d_1 + 1$ qui correspond à la ligne $d_3^{(i)}$ du sous-bloc $sbloc_1$,..., le dernier sous-vecteur $(d_1^{(i)}.d_3^{(i)})$ qui correspond à la ligne $d_3^{(i)}$ du sous-bloc $sbloc_{d_1^{(i)}}$.

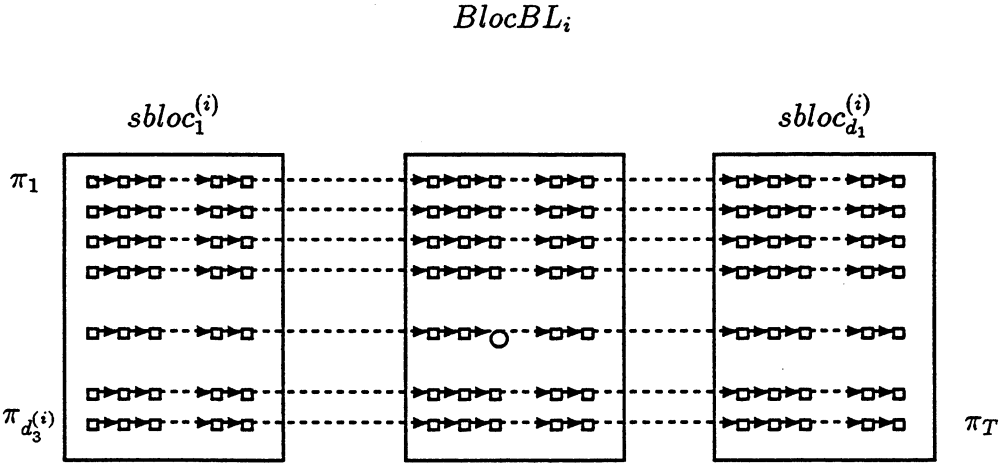


Figure 6.7: Vision du vecteur $\pi^{\sigma_3^{(i)}}$ après le mélange parfait à la phase i

Le vecteur $\pi^{\sigma_3^{(i)}}$ ainsi constituer doit être multiplié par la matrice bloc diagonale de taille T, ($Id_{B_i} \otimes M_i$). Multiplier un vecteur $\pi^{\sigma_3^{(i)}}$ par une telle matrice, revient à faire une suite de multiplication de sous-vecteur de $\pi^{\sigma_3^{(i)}}$ de taille b_i par la matrice M_i, puis replacer les composantes du vecteur résultat selon d'ordre initial par la permutation $S_{\sigma_3^{(i)}}{}^*$. Or on remarque que l'ordre comment sont faites les multiplications des sous-vecteurs par la matrice M_i n'est pas important. En faite dans la figure 6.7, il suffit de prendre chaque ligne du sous-bloc $sbloc_1$ et de la multiplier par la matrice M_i puis remettre le résultat à même ligne dans le sous-bloc $sbloc_1$ et on fait de même pour chaque ligne des sous-blocs $sbloc_j$, $j \in [\, 2..d_1 \,]$.

Remarque

Pour une phase de calcul i, si on affecte chacun des sous-blocs $sbloc_j$, $j = 1..d_1^{(i)}$ sur une place p_j, la phase de calcul du produits du vecteur π^{σ_3} par la matrice bloc diagonal ($Id_{B_i} \otimes M_i$), s'effectue indépendamment dans chaque place.

Pour effectuer le calcul de la phase $(i+1)$ et avant la permutation $S_{\sigma_3^{(i+1)}}$, le vecteur π est vu comme un bloc BL_{i+1} qui contient $d_1^{(i+1)} = b_i.d_1^{(i)}$ sous-blocs de $d_2^{(i+1)} = b_{i+1}$ colonnes contenant $d_3^{(i+1)} = \frac{d_3^{(i)}}{b_i}$ éléments. Les colonnes de $sbloc_j^{(i)}$ sont vues comme

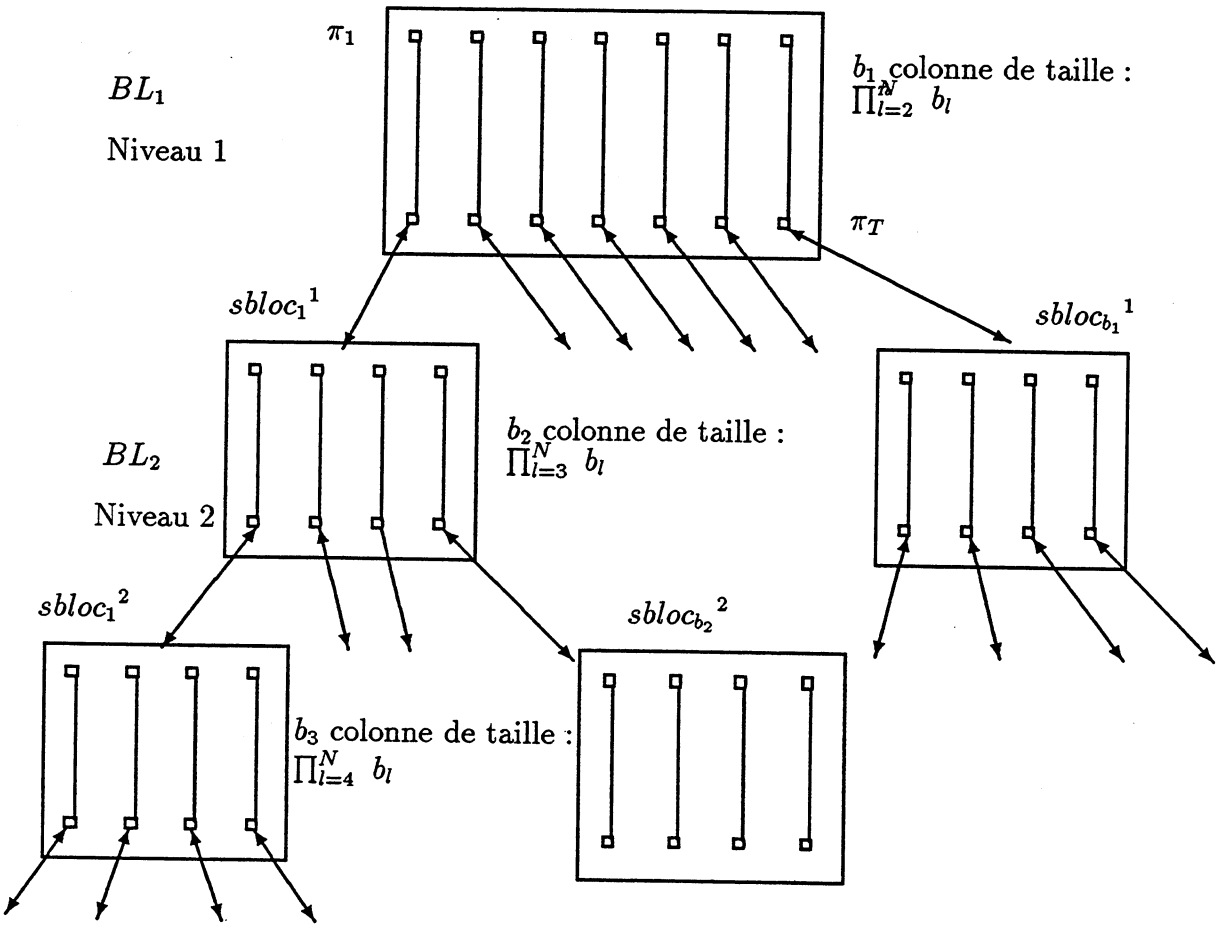


Figure 6.8: Représentation arborescente des mélanges parfaits successifs

des sous-blocs $sbloc_j^{(i+1)}$.

Pour effectuer la multiplication par la matrice M_{i+1}, on applique le même principe que dans la phase i mais en utilisant les nouveaux sous-blocs, d'où la vision d'arbre des mélanges parfaits successifs (figure 6.8).

Exemple d'utilisation des mélanges parfaits pour calculer π [$\otimes_{i=1}^{3} M_i$]

$b_1 = 2,\ b_2 = 3, b_3 = 2, T = 12$

$$BL_0 = \begin{pmatrix} \pi_1 \\ \pi_2 \\ \pi_3 \\ . \\ . \\ . \\ \pi_{11} \\ \pi_{12} \end{pmatrix}$$

$$BL_1 = \begin{pmatrix} \pi_1 & \pi_7 \\ \pi_2 & \pi_8 \\ \pi_3 & \pi_9 \\ \pi_4 & \pi_{10} \\ \pi_5 & \pi_{11} \\ \pi_6 & \pi_{12} \end{pmatrix}$$

Première multiplication :

- $(\pi_1,\ \pi_7)\ \ M_1\ =\ (\pi_1^1,\ \pi_7^1)$
- $(\pi_2,\ \pi_8)\ \ M_1\ =\ (\pi_2^1,\ \pi_8^1)$
- $(\pi_3,\ \pi_9)\ \ M_1\ =\ (\pi_3^1,\ \pi_9^1)$
- $(\pi_4,\ \pi_{10})\ \ M_1\ =\ (\pi_4^1,\ \pi_{10}^1)$
- $(\pi_5,\ \pi_{11})\ \ M_1\ =\ (\pi_5^1,\ \pi_{11}^1)$
- $(\pi_6,\ \pi_{12})\ \ M_1\ =\ (\pi_6^1,\ \pi_{12}^1)$

Décomposition du bloc BL_2 :

$$sbloc_1^2 = \begin{pmatrix} \pi_1^1 & \pi_3^1 & \pi_5^1 \\ \pi_2^1 & \pi_4^1 & \pi_6^1 \end{pmatrix}$$

$$sbloc_2^2 = \begin{pmatrix} \pi_7^1 & \pi_9^1 & \pi_{11}^1 \\ \pi_8^1 & \pi_{10}^1 & \pi_{12}^1 \end{pmatrix}$$

Deuxième multiplications :

- 1) $(\pi_1^1,\ \pi_3^1,\ \pi_5^1)\ \ M_2\ =\ (\pi_1^2,\ \pi_3^2,\ \pi_5^2)$
- 1) $(\pi_2^1,\ \pi_4^1,\ \pi_6^1)\ \ M_2\ =\ (\pi_1^2,\ \pi_4^2,\ \pi_6^2)$
- 2) $(\pi_7^1,\ \pi_9^1,\ \pi_{11}^1)\ \ M_2\ =\ (\pi_7^2,\ \pi_9^2,\ \pi_{11}^2)$
- 2) $(\pi_8^1,\ \pi_{10}^1,\ \pi_{12}^1)\ \ M_2\ =\ (\pi_8^2,\ \pi_{10}^2,\ \pi_{12}^2)$

Décomposition du bloc BL_3

$$\begin{aligned} sbloc_1^3 &= (\pi_1^2,\ \pi_2^2)\ , \\ sbloc_2^2 &= (\pi_3^2,\ \pi_4^2)\ , \\ sbloc_3^2 &= (\pi_5^2,\ \pi_6^2)\ , \\ sbloc_4^2 &= (\pi_7^2,\ \pi_8^2)\ , \\ sbloc_5^2 &= (\pi_9^2,\ \pi_{10}^2)\ , \\ sbloc_6^2 &= (\pi_{11}^2,\ \pi_{12}^2). \end{aligned}$$

Troisième multiplications :

- 1) $(\pi_1^2,\ \pi_2^2)\ M_3\ =\ (\pi_1^3,\ \pi_2^3)$
- 2) $(\pi_3^2,\ \pi_4^2)\ M_3\ =\ (\pi_3^3,\ \pi_4^3)$
- 3) $(\pi_5^2,\ \pi_6^2)\ M_3\ =\ (\pi_5^3,\ \pi_6^3)$
- 4) $(\pi_7^2,\ \pi_8^2)\ M_3\ =\ (\pi_7^3,\ \pi_8^3)$
- 5) $(\pi_9^2,\ \pi_{10}^2)\ M_3\ =\ (\pi_9^3,\ \pi_{10}^3)$
- 6) $(\pi_{11}^2,\ \pi_{12}^2)\ M_3\ =\ (\pi_{11}^3,\ \pi_{12}^3)$

Le vecteur résultat est :

$$\begin{pmatrix} \pi_1^3 \\ \pi_2^3 \\ \pi_3^3 \\ . \\ . \\ . \\ \pi_{11}^3 \\ \pi_{12}^3 \end{pmatrix}$$

6.3 Présentation des différentes approches de parallélisation

On rappelle que le problème qu'on veut paralléliser est le calcul du produit suivant :

$$\pi \ P. \tag{6.23}$$

Avec P exprimée comme suit :

$$P \ = \ \Sigma_{t=1}^{C} \ \otimes_{i=1}^{N} M_{ti} \tag{6.24}$$

Où π est le vecteur de probabilités stationnaire du système étudier, P étant la matrice caractérisant le fonctionnement du système considéré. Les matrices M_{ti} sont de tailles $b_i \ , i = 1..N$.

Dans ce chapitre nous allons étudier les différentes approches de parallélisation qui tiennent en compte des niveaux de parallélisme de l'expression 6.24 qui se résument comme suit :

- **Paralléllisation des termes : (voir le paragraphe 6.4)**

  On appelle un terme le produit $(\pi \otimes_{i=1}^{N} M_{ti})$ pour un t donné. Cette approche consiste à paralléliser terme après terme les C termes de la somme. Le calcul d'un terme t donné est répartie sur tous les processeurs de la machine.

- **Parallélisation de la somme : (voir le paragraphe 6.5)**

  Dans cette approche parallèle, on suppose que C le nombre de termes de la somme est très grand devant le nombre p de processeurs de la machine et on décide d'effectuer un terme par processeur. Le calcul des termes (le produit $(\pi \otimes_{i=1}^{N} M_{ti})$) n'est pas paralléliser.

- **Parallélisation de la somme et des termes : (voir le paragraphe 6.6)**

  Dans ce cas, nous essayons d'effectuer plusieurs termes simultanément tout en parallélisant le calcul de chaque terme. En peut imaginer que les processeurs de la machine sont groupés en un certain nombre de groupes et que chaque groupe de processeurs effectue la parallélisation d'un terme.

Nous allons voir plus loin est en détails ces 3 approches avec leurs avantages et inconvénients. Nous allons commencer par étudier la parallélisation des termes qui est la base de la première et la troisième approche de parallélisation présentée ci-dessus.

6.4 Première approche : parallélisation du calcul d'un terme ($\pi\ [\ \otimes_{i=1}^{N} M_{ti}]$)

6.4.1 Introduction

Dans les paragraphes précédent, nous avons présenté le principe du calcul du produit ($\pi\ [\ \otimes_{i=1}^{N} M_{ti}]$). Nous avons vu que selon qu'on utilise telle ou telle factorisation de cette expression, le calcul et différent. Dans le cadre de la parallélisation des termes $\pi\ [\ \otimes_{i=1}^{N} M_{ti}]$, nous allons étudier 3 méthodes de parallélisations.

Dans le paragraphe 6.4.2, nous allons présenter une méthode de parallélisation qui se base sur les mélanges parfaits dans une base de dimension 2. Nous discutons l'effet du placement des sous-vecteurs $s\pi_j$ et $s\pi_j^{\sigma_2}$ sur les processeurs de la machine. Puis on donne une implantation parallèle de cette méthode.

Dans le paragraphe 6.4.3, nous étudions la parallélisation de la méthode qui se base sur les mélanges parfaits dans une base de dimension 3. Nous verrons les problèmes dû au placement des sous-blocs $sbloc_j^{(i)}$ sur les processeurs de la machine . Enfin, dans le paragraphe 6.4.4, nous proposant une méthode parallèle mixte qui tire ces avantages des deux précédentes.

6.4.2 Parallélisation utilisant les mélanges parfaits dans une base de dimension 2

Dans le paragraphe 6.2.3, nous avons vu le principe du mélange parfait dans une base de dimension 2. On rappelle que le déplacement des composantes du vecteur π pour la formation du vecteur π^{σ_2} est donnée par le théorème 3 :

Si on place chaque sous-vecteur $s\pi_j$ et $s\pi_j^{\sigma_2}$, $j = 0..p-1$ sur un processeur p_j, lors du mélange parfait, chaque processeur doit envoyer ces b_i groupes (gcc) à b_i processeurs. Cela est vrai dans le cas des bonnes divisibilités suivantes :

- T est divisible par p.
- B_i est divisible par p.

Il faut noter qu'en pratique ces conditions sont rarement vérifiées. Dans le cas général, la notion de groupe (gcc) est toujours vraie car elle caractérise le mélange parfait. Par

contre le nombre le groupe (gcc) et la taille de ces groupes sont différents d'un processeur à un autre selon la charge des processeurs.

Dans ce qui suit, on se place dans les conditions du théorème 3 pour étudier la localité des déplacements des composantes par le mélange parfait. Etant donné le placement des sous-vecteurs $s\pi_j$ sur les processeurs p_j, nous allons essayer de trouver un placement des sous-vecteurs $s\pi_j^{\sigma_2}$ sur les processeurs p_l avec l non nécessairement égal à j, afin de réduire le mouvement des données (communication).

Nous poursuivons notre étude, en proposant une implantation parallèle générale de la méthode qui utilise les mélanges parfaits dans une base de dimension 2.

6.4.2.1 Problème de placement pour résoudre les mélanges parfaits dans une base de dimension 2

Aspect général du placement

Etant donné une topologie de communication, faire un placement de p sous-vecteurs sur p processeurs de façon à minimiser une fonction de coût dûe au communications interprocesseur est un problème Np-complet. En fait, il existe p^p façons de placer p sous-vecteurs sur p processeurs. Pour résoudre de telle problème, il n'existe aucun algorithme polynomiale qui donne une solution exacte. Cependant, on trouve dans la littérature une variété d'algorithmes qui donnent d'assez bonnes solutions (approchées).

Idée du placement

Etant donnée une topologie de communication en tore, pour minimiser le temps de communication pour une phase de permutation donnée, il faut qu'on trouve un placement des sous-vecteurs de π^{σ_2} sur les processeurs de la machine de façon que chaque processeur ne communique qu'avec ces proches voisins. En utilisant le théorème 3, on sait que chaque processeur contenant un sous-vecteur de π donné n'a besoin de communiquer qu'avec b_i processeurs à une phase de permutation i. L'idée de base est la suivante :

Partons du placement fixé vu dans le paragraphe 6.2.3, il s'agit de trouver sur quels processeurs devrait-on former les sous-vecteurs $s\pi_l^{\sigma_2}$ de telle manière que ceux ci soit les voisins les plus proches des processeurs qui contiennent les sous-vecteurs $s\pi_j$ responsables de la formation des sous-vecteurs $s\pi_l^{\sigma_2}$, avec $l,\ j\ \in\ [\ 0..p-1\ ]$.

Exemple de placement

Soit un réseau de 6 processeurs en grille (2×3). Les numéros à l'extérieur des cases (figure 6.9) représentent les numéros des processeurs, les numéros à l'intérieur sont les

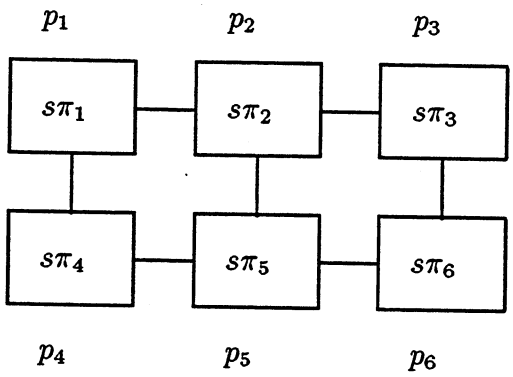


Figure 6.9: Placement initial des sous-vecteurs de π sur les processeurs

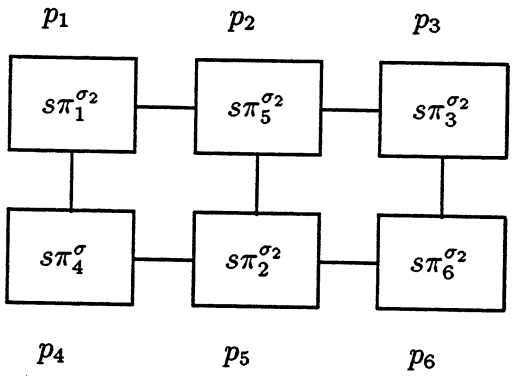


Figure 6.10: Placement des sous-vecteurs de π^{σ_2} sur les processeurs

numéros des sous-vecteurs de π. Pour l'affectation initiale, on place le sous-vecteur $s\pi_j$ sur le processeur p_j pour j allant de 1 à 6. On suppose que pour effectuer la phase de permutation, une partie des éléments du sous-vecteur $s\pi_1$ contribue à la formation des sous-vecteurs $s\pi_4^{\sigma_2}$ et $s\pi_5^{\sigma_2}$ ($s\pi_1$ contient 2 groupes : g_{14} et g_{15}). Une partie des éléments du sous-vecteur $s\pi_3$ contribue à la formation des sous-vecteurs $s\pi_5^{\sigma_2}$ et $s\pi_6^{\sigma_2}$ ($s\pi_3$ contient 2 groupes : g_{35} et g_{36}). Un bon placement se schématiserait par la figure 6.10. Avec ce placement nouveau, le sous-vecteur $s\pi_1$ va envoyer une partie de ces éléments au processeur de p_2 pour former le sous-vecteur $s\pi_5^{\sigma_2}$. De même, pour le processeur p_3 au lieu d'envoyer une partie de ses éléments au processeur p_5 qui à une distance 2 (dans le cas d'un placement $s\pi_j^{\sigma_2}$ sur p_j), il l'envoie au processeur p_2 pour former le sous-vecteur $s\pi_5^{\sigma_2}$. Ce placement minimise les communications (en terme de distance) pour effectuer la permutation des éléments du vecteur π.

Dans le paragraphe suivant, nous allons construire la fonction de coût d'une affectation dans le cas général.

Fonction de coût

Pour construire la fonction de coût, nous allons définir deux graphes :

- On définit le graphe de communication G_i pour une phase de permutation donnée par une matrice g_i :

  $g_i[\ k,l]= 1$, si le sous-vecteur $s\pi_k$ envoie une partie de ses éléments pour former le sous-vecteur numéro $s\pi_l^{\sigma_2}$ ($s\pi_k$ contient le groupe g_{kl}) , sinon $g_i[\ k,l]=0$.

- De la même façon on définit un graphe des distances D pour toutes les phases de permutations par une matrice d :

  $d[\ p_k,p_l]=$ distance minimum en nombre d'arêtes entre le processeur numéro p_k et le processeur numéro p_l dans la topologie de connexion considérer (tore).

Pour chaque phase de permutation i, on représente un placement par un vecteur map de taille p. l'élément $map[\ k\ ]$ représentant le numéro du processeur qui va contenir le sous-vecteur $s\pi_k^{\sigma_2}$. Le placement initial est mémoriser dans le vecteur $mapi$. La fonctionnelle de coût qu'il faut minimiser à chaque phase de permutation est la suivante:

$$Fcout \ = \ \sum_{k,l} c^{d[\ mapi\ [k],map[l]]}\ g_i\ [k,l] \tag{6.25}$$

La constante c est un entier positif qui pénalise l'éloignement entre deux processeurs qui contiennent respectivement, le premier un sous-vecteur de π, le second contient le sous-vecteur de π^{σ_2} à former par une partie des éléments du premier sous-vecteur.

Dans ce qui suit nous allons minimiser notre fonctionnelle de coût, en utilisant des heuristiques connue comme l'algorithme "échange de paires" et l'algorithme de récuit simulé.

Algorithme de Bokhari "échange de paires"

L'algorithme par échange de paires "pairwise exchange" est décrit dans [4], le principe de cet algorithme est le suivant :

Etant donné un placement initial, l'algorithme consiste à trouver la meilleure paire de processeurs, telle que l'échange de leur sous-vecteur fait décroitre la fonction de coût. On adopte alors ce nouveau placement pour itérer ce processus tant qu'on peut améliorer la fonction de coût. Pour éviter les minimums locaux, l'algorithme génère un placement

aléatoire en échangant aléatoirement $\sqrt{p}$ paires. L'algorithme est donné dans l'annexe E, cet algorithme a une compléxité de $O(\ p^3)$ avec p le nombre de processeurs du réseau.

Algorithme utilisant le récuit simulé

Le principe de cet algorithme repose sur une analogie entre certains effets thermodynamiques et la résolution de certains problèmes d'optimisation [41].

Le recuit est un abaissement contrôlé de température d'un ensemble de particules en interaction. Il permet d'obtenir une configuration des particules à énergie minimale lorsque la température est devenue suffisamment basse. La température imposée contrôle les états du système (placements donnés) accessibles à partir de l'état courant (placement courant).

L'énergie d'une configuration du système représente la fonction de coût d'un placement. La température est simulée par une variable de contrôle que l'on fait décroitre lentement à chaque itération. Pour cela, à chaque itération la température est multipliée par un coefficient constant $a \in [\ 0..1\ [$.

L'algorithme s'arrête lorsque la température est proche de 0. Le changement d'état du système s'exprime par une perturbation de l'état courant en permuttant aléatoirement les sous-vecteurs de deux processeurs. Si l'énergie de l'état résultant est inférieure à celle de l'état courant alors on itére ce processus avec l'état obtenu. Si l'énergie de l'état résultant est supérieure à celle de l'état courant, on accepte sous probabilité l'état obtenu. La probabilité d'acceptation est directement liée à la température courante du système. Plus la température est élevée et plus on a des chances d'autoriser des états de plus grande énergie. Cette technique de remonté permet de sortir des extremums locaux. L'algorithme à une complexité de $(K\ p^2)$, la constante K pouvant être assez grande, elle dépend des températures initiale, finale et du coefficient de variation de la température a.

Comparaison entre les deux Algorithmes (échange de paires et le récuit)

Dans la plupart des cas l'algorithme d'échange de paires donne un placement légèrement meilleur que ce lui donné par l'algorithme de récuit (figure 6.11). Néanmoins, l'algorithme de Bokhari devient très coûteux en temps dès que la taille du tore est supérieure à (7×7). Par contre l'algorithme de récuit simulé supporte bien les tores de taille relativement grande . Les figures 6.12 et 6.13 (représentent le nombre de communications cumulées qui s'effectuent entre des processeurs situés à une distance d), montrent la faible amélioration du placement initial pour une taille de grille et une phase de permutation donnée en utilisant seulement l'algorithme récuit simulé. Il faut noter, que le temps d'exécution de l'algorithme de placement pour une seule phase de

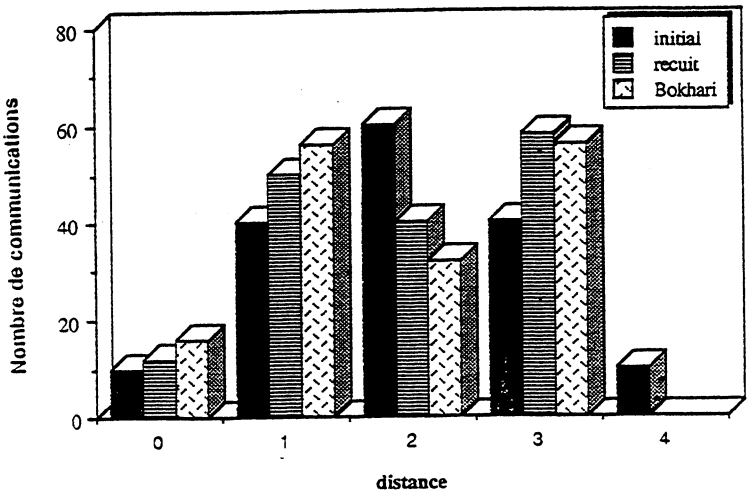


Figure 6.11: Réseau à 16 processeurs, placement à la phase i avec $b_i = 10$

permutation i est de l'ordre de la quart-heure sur une tore (11×11).

Conclusion sur le placement

Pour conclure cette étude sur le placement :

- En première remarque, le temps d'exécution de l'algorithme de placement n'est pas acceptable, car ce temps peut être supérieur au temps du calcul effectif.

- En deuxième remarque, la faible amélioration de la solution donnée par le placement, justifié la non investigation dans la parallélisation de l'algorithme du placement.

Dans ce qui suit, nous allons revenir à notre problème qui est d'effectuer en parallèle le produit π ($\otimes_{i=1}^{N} M_i$) en utilisant le théorème 4 qui se base sur les mélanges parfaits dans une base de dimension 2. En pratique, les conditions du théorème 3 sont très restrictives. Dans le paragraphe suivant, nous allons voir le principe de la parallélisation du produit ci-dessus sans tenir compte du théorème 3. Avec le placement des sous-vecteurs $s\pi_j$ et $s\pi_j^{\sigma_2}$ sur le même processeur p_j.

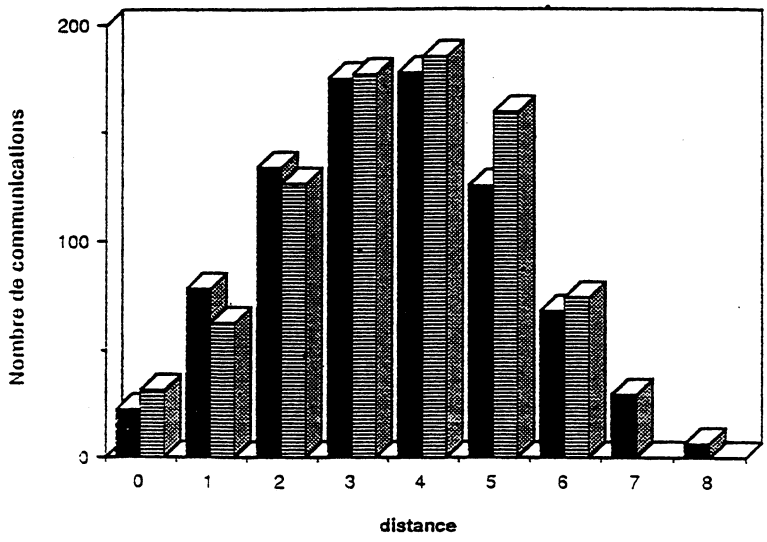


Figure 6.12: Réseau à 64 processeurs, placement à la phase i avec $b_i = 6$

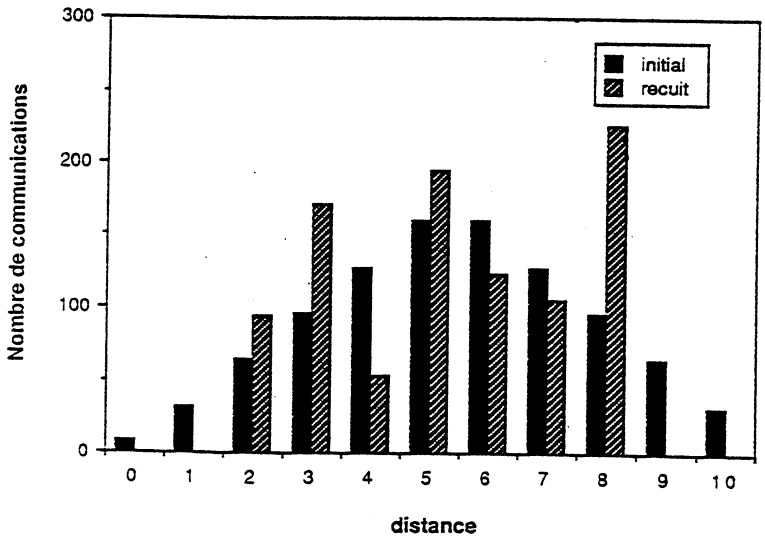


Figure 6.13: Réseau à 121 processeurs, placement à la phase i avec $b_i = 8$

6.4.2.2 Détails de parallélisation de $\pi \prod_{i=1}^{N} [S_{b_i B_i} (Id_{B_i} \otimes M_i)]$

L'idée consiste à distribuer le vecteur π sur les processeurs de la machine. Le calcul comporte la répétition (N fois) de deux phases principales :

- Calcul des permutations (nouveau placement des éléments du vecteurs π):

  Dans cette première phase, le calcul des permutations se réduit à ce que chaque processeur calcule les indices des éléments du vecteur π dont il a besoin pour effectuer les multiplications par les matrices M_i.

- Après la permutation, sur chaque processeur, le sous-vecteur $s\pi^{\sigma_2}$ est composé d'un certain nombre de blocs de taille b_i, il reste donc à effectuer les multiplications de chacun des blocs par la matrice M_i.

On appelera "étape" la réunion des deux phases.

Stratégie d'allocation des données

A chaque étape i, on s'arrange pour avoir sur chaque processeur des données (sous-vecteurs de π) de taille multiple de b_i (taille de la matrice M_i) .

Soit $T = \prod_{i=1}^{N} (b_i)$ la taille du vecteur π complet et N le nombre de matrices du produit tensoriel. On pose $B_i = \frac{T}{b_i}$ le nombre de blocs dans π de taille b_i. Si notre machine avait B_i processeurs, on aurait affecté à chaque processeur un bloc de taille b_i de π. Comme en général ce n'est pas le cas, on affecte à chaque processeur de la machine:

$$qt = B_i \; div \; p$$

blocs de taille b_i, ce qui constitue le sous-vecteur $s\pi$. Si le reste $r = \text{mod} (B_i , p)$ est non nul alors les r premiers processeurs auront $(qt + 1)$ blocs de taille b_i . Les $(p - r)$ derniers n'auront que qt sous-vecteurs de taille b_i . De la même manière vue précédemment, le vecteur π est distribué sur les processeurs de la machine telle que le sous-vecteur $s\pi_j$ est alloué au processeur $p_j \quad j \in [1..p]$:

Plus précisémment, le sous-vecteur $s\pi_1$ a pour éléments les $(qt.b_i)$ ou $(qt+1)b_i$ premières composantes du vecteur π, le sous-vecteur $s\pi_2$ a pour éléments les $(qt.b_i)$ ou $(qt + 1).b_i$ suivantes et ainsi de suite jusqu'au sous-vecteur $s\pi_p$ qui aura pour éléments les $qt.b_i$ ou $(qt + 1).b_i$ dernières composantes du vecteur π. Après chaque permutation du vecteur π, on retrouve la même allocation des sous-vecteurs $s\pi_j^{\sigma_2}$ sur les processeurs

p_j . Il faut noter que chaque processeur doit avoir les N matrices M_i dans sa mémoire pour pouvoir effectuer les phases de multiplication de leurs blocs (de taille b_i) par la matrices M_i .

Stratégie de communication pour effectuer les mélanges parfaits

Pour réaliser le mélange parfait ou permutation des éléments du vecteur π , au début de chaque étape i, chaque processeur calcule les indices des éléments dont il a besoin. Comme ces éléments peuvent être sur les autres processeurs (vu la distribution du vecteur π sur les processeurs à l'étape i-1). Il faut établir une phase de communication. On se propose dans un premier temps d'utiliser un schéma de l'algorithme de communication du type *échange total* nous verrons plus loin l'amélioration des communication, en exploitant la vision du théorème 3.

Dans un *échange total*, chaque processeur p_j doit envoyer le sous-vecteur qui lui est affecté $s\pi_j$ à tous les autres processeurs afin qu'ils puissent retirer les éléments dans ils ont besoin pour former leur sous-vecteur $s\pi_j^{\sigma_2}$. Pour ce faire, nous allons utiliser la procédure [1] proposée dans le chapitre 2.

Dans ce qui suit, on note π_i le vecteur résultat obtenu après la multiplication par la matrice M_i.

Algorithme principal sur chaque processeur :

{* le processeur local a comme numéro "*numproc*" *}

Début
{* N étant le nombre de matrices du produit tensoriel *}
(0) Calcul de l'allocation décrite ci-dessus et la distribution du vecteur π_{init}.
POUR i =1 à N
(1) Chaque processeur calcule l'ensemble des indices des éléments de π_{i-1} dont il a besoin à l'étape i. C'est le calcul de la permutation de la phase i. Ces positions sont stockées dans le sous-vecteur local *newindipi*
(2) Parcours du vecteur *newindipi* et de $s\pi_{numproc}$ pour recupérer les éléments qui lui sont nécessaires pour former $s\pi^{\sigma_2}$ *newpi* .
(3) Echange total
(4) Produit des blocs de taille b_i de $s\pi^{\sigma_2}$ par la matrice M_i .
Obtention du vecteur π_i (un nouveau $s\pi$ sur chaque processeur)
FINPOUR
FIN

Détail de chaque pas de l'algorithme principal :

- (0) Algorithme qui calcule les indices des éléments de π^{σ_2} alloué au processeur après le mélange parfait de la phase i :

  $numproc$: le numéro du processeur courant $\in [1\ ,\ p\ ]$.
  $indideb$: l'indice de début du sous-vecteur de $s\pi_{numproc}$.
  $indifin$: l'indice de fin du sous-vecteur de $s\pi_{numproc}$.

  Début
  {* calcul de $indideb$ et de $indif$ *}
  $qt = \frac{B_i}{p}$ {* division entière *}
  r = mod (B_i , p) {* mod = reste de la division entière *}
  Si $r \neq 0$
  Alors Si $numproc > r$
  Alors
  $indideb = (\ qt + 1\ )\ r\ b_i + (\ numproc\text{-}r\text{-}1\ )\ qt\ b_i + 1.$
  $indifin = indideb + qt\ b_i$ -1 .
  Sinon
  $indideb = (\ qt + 1\ )\ (numproc$ -1 $)\ b_i + 1$.
  $indifin = indideb + (\ qt + 1)\ b_i$ -1 .
  Fsi
  Sinon
  $indideb = qt\ (numproc$ -1 $)\ b_i + 1$.
  $indifin = indideb + qt\ b_i$ -1 .
  Fsi
  Fin

  Dans le paragraphe 6.2.3, nous avons défini le mélange parfait par la permutation

  $$\sigma_2\ (d_1,\ d_2)\ :\quad k_i\ =\ (\ i_1\ -1)\ d_2\ +\ i_2\ \longmapsto\ k_j\ =\ (i_2\ -1)\ d_1\ +\ i_1.$$

  Pratiquement cette application signifie que l'élément d'indice k_i dans le vecteur π va aller à la position k_j du vecteur π^{σ_2} . Pour notre algorithme nous avons besoin de l'application inverse : $\sigma_2^{-1}\ (b_i\ ,\ B_i)$ qui à chaque indice k_l associe l'indice k_{ll} de l'élément du vecteur π qui sera à la position k_l du vecteur π^{σ_2}.

- (1) Calcul des indices des éléments du vecteur π^{σ_2} alloué au processeur après le mélange parfait de la phase i .

Début
Pour $k_l = indideb$ à $indif$
{* décomposition de k_l en (i_2 , i_1) resp. numéro du bloc et position dans le bloc puis réalisation de la permutation σ_2^{-1} (b_i ,B_i). *}
$i_1 = mod(\ k_l\ ,\ b_i\ )$
$i_2 = k_l\ div\ b_i$ {*division entière *}
Si $i_1 = 0$
Alors {* Cas où k_l est le dernier élément d'un bloc de b_i éléments *}
$i_1 = b_i$
Sinon{* Le numéro de bloc est le numéro du bloc suivant *}
$i_2 = i_2 + 1$
Fsi
{* $\sigma_2^{-1}\ (\ k_l\ )$ *}
k_{ll}=(i_1 -1) $B_i + i_2$
{* stockage dans *newindipi* des indices des éléments à mémoriser *}
newindipi [k_l - $indideb$ +1] = k_{ll}
Finpour
Fin

- (3) En utilisant la procédure d'échange total qui se décompose en $\frac{p-1}{4}$ étape. Chaque étape consiste à envoyer et recevoir simultanément sur les 4 liens du processeur des données ($s\pi_j$). A la première étape, le processeur courant envoie à ses 4 voisins :
  - L'indice du début du vecteur (*indideb*)
  - L'indice de la fin du vecteur (*indif*)
  - Son sous-vecteur $s\pi_{numproc}$.

- (2),(4) Prélèvement des éléments reçus par le processeur dans le sous-vecteur $s\pi^{\sigma_2}_{numproc}$ "*newpi*". Nous allons donner deux versions, la première version ne tient pas en compte des propriétés des mélanges parfaits (groupe gcc) décrites dans le paragraphe 6.2.3. Par contre la deuxième version exploite justement ces propriétés .

Algorithme version 1

{* *indidebrecu* et *indifrecu* sont les indices respectivement du début, fin du vecteur courant reçu par un processeur donné. *}

Début
$taillevect = indif\text{-}indideb + 1$
Pour $l = 1$ à $taillevect$
Si $indidebrecu \leq newindipi\ [\ l\ ] \leq indifrecu$

```
        Alors
              newpi [ l ] = pi [ newindipi [ l ] - indidebrecu +1 ]
        Fsi
  Finpour
Fin
```

Avant de présenter la deuxième version de l'algorithme, on rappelle que chaque sous-vecteur $s\pi$ de chaque processeur est composé de groupes de composantes consécutives (gcc) qui sont notés par : g_{da}, où d indique le numéro du processeur qui contient les composantes du groupe avant la permutation et a indique le numéro du processeur qui va contenir les composantes du groupe après la permutation. Dans un sous-vecteur reçu (pendant l'échange total) par un processeur, les groupes $g_{numrecu,?}$ sont rangés comme suit :

Le premier groupe $g_{numrecu,a}$ (si a dénote le numéro du processeur qui va contenir la première composante ($indidebrecu$) après la permutation) est suivit par les groupes $g_{numrecu,a+1}$, $g_{numrecu,a+2}$, $\cdots$, $g_{numrecu,(a+j)[p]}$ (voir le paragraphe 6.2.3). Les numéros des groupes sont consécutifs modulo p. Pour connaitre cette liste de numéro, il suffit de calculer le numéro du premier groupe $g_{numrecu,a}$ et du dernier.

L'algorithme de recopie version 2 ci-dessous, se compose en deux parties :

- Vérification si le groupe $g_{numrecu,numproc}$ appartient à la liste des groupes du sous-vecteur reçu (un booleen est à vrai si la réponse est positive)
- Recherche du groupe est sa recopie dans le sous-vecteur $s\pi^{\sigma_2}$ du processeur courant $numproc$.

Algorithme version 2

```
Début
  ok=Faux
  {* Numéro des processeurs qui vont contenir le premier et le dernier groupe *}
  numd=NumProcArriv(indidebrecu)
  numf=NumProcArriv(indifrecu)
  {* Première partie : vérification si g_{numrecu,numproc} ∈ sπ_{numrecu} *}
  diff=numf − numd
  Si (diff > 0)
  Alors Si (numd ≤ numproc) Et (numf ≥ numproc)
        Alors ok= Vrai
        Fsi
  Fsi
  Si (diff < 0)
```

```
Alors Si ((numd ≤ numproc) Et (p ≥ numproc))
        Ou ((1 ≤ numproc) Et (numf ≥ numproc))
      Alors ok= Vrai
      Fsi
Fsi
{* Deuxième partie : Recherche du groupe est sa recopie dans le cas ok =vrai *}
debgroupe=Vrai
fingroupe=Faux
l=1
Tant que l ≤ taillevect Et Non fingroupe
Si indidebrecu ≤ newindipi [ l ] ≤ indifrecu
          Alors
                {* selon le cas, "pi" désigne soit sπ_numproc soit sπ_numrecu *}
                newpi [ l ] = pi [ newindipi [ l ] - indidebrecu +1 ]
                Si debgroupe=Faux Alors debgroupe=Vrai Fsi
                l=l+b_i{* prochaine composante du groupe *}
          Sinon Si debgroupe=Vrai
                Alors fingroupe=Vrai
                Sinon l=l + 1{* recherche du début du groupe *}
                Fsi
          Fsi
 FinTantque
Fin
```

- (6) Algorithme de calcul des produits des blocs de taille b_i de $s\pi^{\sigma_2}$ ("*newpi*") par la matrice M_i, le principe de cet algorithme est le suivant :

  Sur chaque processeur le sous-vecteur de données est composé d'un nombre entier de blocs de taille b_i, il suffit d'effectuer successivement le produit de chaque bloc par la matrice M_i à l'étape i (voir le paragraphe 6.2.5).

  Algorithme du produit

```
Début
{* la variable it permet de se positionner successivement au début de chaque bloc *
 it =1
 Tant que it ≤ taillevect
          Pour ic= it à b_i +it -1
               pi [ ic ] =0
               Pour il= it à b_i +it -1
                    pi [ ic ] = pi [ ic ] + newpi [ il ] * M_i [ il ,ic ]
```

FinPour
FinPour
$it = it + b_i$
{* position du début du prochain bloc à multiplier par la matrice M_i *}
FinTantque
FIN

Remarque :

Dans cette méthode de parallélisation, vu la compléxité du calcul (multiplication des blocs du sous-vecteur $s\pi_j^{\sigma_2}$ par la matrice M_i) qui est de l'ordre de $O(\frac{T\ \Sigma_{i=1}^{N}\ b_i}{p})$, cette méthode génère des communication (*échange total*) qui sont de l'ordre de $O(\frac{N\ T}{4})$. On génère un surcoût de communication qui est de l'ordre du calcul. Cette méthode s'avère donc inefficace.

Dans le pragraphe suivant nous allons étudier la parallélisation de l'expression $\pi\ [\ \otimes_{i=1}^{N} M_i$ en utilisant la vision des mélanges parfaits dans une base de dimension 3.

6.4.3 Paralléllisation utilisant les mélanges parfaits dans une base de dimension 3

Nous rappelons que cette méthode se base sur la factorisation donnée par le théorème 5. Chaque niveau i de l'arbre des mélanges parfaits successifs (voir le paragraphe 6.2.4) correspond une affectation des sous-blocs $sbloc_j^{(i)}$ sur des places p_j. Nous avons vu que le calcul de la phase i : produit du vecteur π^{σ_3} par la matrice bloc diagonal $(Id_{B_i} \otimes\ M_i\ )$, se fait indépendamment dans les places p_j. Dans le cadre de la parallélisation de ces calculs, on affecte chaque place à un processeur de la machine.

Dans le cas où le nombre de sous-blocs $sbloc_j^{(i)}$ est supérieur au nombre de processeur de la machine, certain processeurs se voit affecter plus d'un sous-bloc. Il en résulte un déséquilibrage de la charge du calcul. Nous revenons sur cet aspect dans le paragraphe suivant.

Etant donnée une affectation des sous-blocs du niveau k de l'arbre des mélanges parfaits successifs sur les processeurs, tous les mélanges parfaits pour $i\ >\ k$ se font localement. Pour effectuer les mélanges parfaits pour $i\ \leq\ k$, on regroupe les sous-blocs du niveau $i+1$ pour retrouver l'affectation du niveau i. Par ce procédé, plus on remonte dans l'arborescence et plus on regroupe les sous-blocs et moins en moins on

utilise de processeur. A la étape $i = 1$ le vecteur entier est sur un processeur et les autres processeurs sont inactifs.

L'algorithme suivant [35], illustre la méthode de réalisation des produits des sous-vecteurs de taille b_i par la matrice M_i .

Algorithme de la phase i

Début
{* *taillevect* est la taille du vecteur $s\pi$ *}
$d_1[i] = \prod_{j=1}^{i-1} b_j$;
$d_2[i] = b_i$;
$d_3[i] = \prod_{j=i+1}^{N} b_j$;
$base1 = 0$;
$ml = d_3[i] * d_2[i]$;
$nl = 1$;
Tant que ($nl \leq d_1[i]$ et $base1 < taillevect$) faire
Pour $sl = base1 + 1$ à $base1 + d_3[i]$ faire
$base2 = 0$;
Pour $l = 1$ à b_i faire
$add[l] = sl + base2$;
$elemt[l] = s\pi[add[l]]$;
$base2 = base2 + d3[i]$;
FinPour ;
Produit du sous-vecteur *elemt* par la matrice M_i
Remettre les composantes du sous-vecteur résultats a leurs places dans $s\pi$
FinPour
$base1 = base1 + ml$;
$nl = nl + 1$;
FinTanque
Fin

Remarque :

Cette solution est inecceptable car en remontant l'arbre, il y aurait de moins en moins de processeur actifs. De plus, on ne pourra pas traiter de gros problème vu la contrainte mémoire sur un processeur (à la phase 1 , le vecteur π entier est sur un processeur). Dans ce qui suit, nous allons proposer une méthode de parallélisation mixte qui combine les deux visions des mélanges parfaits.

6.4.4 Parallélisation *mixte* combinant les mélanges parfaits dans une base respectivement de dimension 2 et 3

Dans cette solution, nous allons tirer avantage des deux méthodes que nous venons de décrire dans les paragraphes précédents. Le principe de cette méthode est d'effectuer $(N-k)$ étapes (mélanges parfaits locals et produit par les matrices M_i , $i \in [k..N]$) en utilisant le placement du vecteur π telle qu'il a été décrit dans le paragraphe 6.2.4, coupe au niveau k de l'arbre (figure 6.8). Puis les k dernières étape en utilisant la méthode décrite dans le paragraphe 6.4.2.2 . Dans le paragraphe précédent, nous avons commencé à discuter un problème très important qui est l'équilibrage de la charge de calcul. En effet la charge de calcul sur chaque processeur n'est pas la même selon qu'on affecte aux processeurs des sous-blocs correspondant aux différent niveaux k de l'arbre. Dans ce qui suit, nous allons étudier l'impact du choix du niveau k (après lequel les mélanges parfaits se font localement) sur l'équilibrage de la charge et le coût des communications. Puis nous donnons l'algorithme de la parallélisation mixte.

6.4.4.1 Impact du choix du niveau k sur la recherche du bon grain de parallélisme

Soit le vecteur de probabilité de taille T ordonné selon l'ordre lexicographique sur b_1, b_2, ... b_N (voir le paragraphe 6.2.1). Supposons qu'on décide de prendre le niveau $k = 2$ pour l'affectation des sous-blocs aux processeurs. Le vecteur π est alors vu comme b_1 sous-blocs de $(\prod_{l=2}^{N} b_l)$ éléments. Si $b_1 \geq p$, on affecte un certain nombre de sous-blocs à chaque processeur de la machine de façon à équilibrer le mieux possible la charge sur les p processeurs. Toutes les multiplications du vecteur par les matrices M_2, M_3 ... M_N se feront localement sur chaque processeur sans communication entre processeurs. Pour effectuer la multiplication par la matrice M_1, il faut d'abord faire un mélange parfait des composantes du vecteur par une communication globale ou *échange total.* Dans cette affectation, on charge b_1 *mod* p processeurs à b_1 *div* p+1 sous-blocs les autres processeurs sont chargés à b_1 *div* p blocs. Dans le cas où p divise b_1, l'équilibrage de charge est parfait, sinon le déséquilibrage se répercute au niveau de chaque multiplication par les matrices M_2, M_3,...M_N. Généralement, il est rare que p divise b_1 d'où la nécessité de diminuer le déséquibre de charge.

Une façon simple pour améliorer l'équilibrage de la charge est d'augmenter le nombre de sous-blocs. Pour ce faire, il suffit de voir le vecteur π décomposé en sous-blocs selon le niveau $k = 3$. Ainsi, le nombre de sous-blocs croit $(b_1 b_2)$ et la taille des sous-blocs décroit $(\prod_{l=3}^{N} b_l)$ et un meilleur équilibrage de charge est obtenu. Le désavantage de cette affectation est qu'il faut faire un mélange parfait (*échange total*) pour effectuer la

multiplication par la matrice de taille M_2 soit un *échange total* de plus que l'affectation précédente. De façon générale, plus en descent dans l'arbre (k croissant) plus le nombre des blocs augmente meilleur est l'équilibrage de charge et plus le nombre *d'échange total* augmente.

L'idée est de trouver le meilleur équilibrage de charge en restreingnant le coût des communications. Dans ce qui suit, nous allons calculer les coûts des différentes affectations, en déduire la meilleure affectation des sous-blocs aux processeurs.

6.4.4.2 Evaluation du coût d'une affectation

On note par C_i le coût d'une affectation où le vecteur est décomposé en $\prod_{j=1}^{i} b_j$ sous-blocs de taille $\prod_{j=i+1}^{N} b_j$ et en distribuant ces sous-blocs sur les p processeurs de la machine. Il faut remarquer que C_i (coupe au niveau i de l'arbre) n'a de sens que si $\prod_{j=1}^{i} b_j \geq p$. On note par r_{cc} le rapport du temps d'envoi d'un réel sur le temps d'une opération de l'unité de calcul.

Calcul de C_k

Calculons C_1 :

$$C_1 = b_N\, x_1 + b_{N-1}\, x_1 + \cdots + b_2\, x_1 + r_{cc}\, \frac{p-1}{4}\, x_1 + (\frac{T}{b_1}\ div\ p + 1)\, b_1^2$$

avec

$$x_1 = (b_1\ div\ p + 1) \prod_{j=2}^{N} b_j$$

Les termes $b_j\ x_1$ sont les coûts des produits des sous-vecteurs de π par les matrices M_i. Plus précisémment $b_i\ x_1$ représente $(b_1\ div\ p + 1)\ \frac{(\prod_{j=2}^{N} b_j)}{b_i}$ multiplication sous-vecteur matrice qui est de l'ordre de b_i^2. Le terme $r_{cc}\ \frac{p-1}{4}\ x_1$ est le coût d'un *échange total* pour réaliser le mélange parfait. Où x_1 est la taille maximale des sous-vecteurs échangés. Le dernier terme de C_1 correspond au coût de la multiplication des sous-vecteurs de π par la matrice M_1, les sous-vecteurs considérés sont le résultat du mélange parfait des composantes du vecteur π arrangées selon un ordre adapté à la multiplication par la matrice M_1.

Calculons C_2 :

$$C_2 = b_N\, x_2 + b_{N-1}\, x_2 + \cdots + b_3\, x_2 + r_{cc}\, \frac{p-1}{4}\, x_2 + (\frac{T}{b_2}\ div\ p + 1)\, b_2^2 + r_{cc}\, \frac{p-1}{4}\, (\frac{T}{b_2}\ div\ p + 1)\, b_2 + (\frac{T}{b_1}\ div\ p + 1)\, b_1^2$$

avec

$$x_2 = (b_1\, b_2\ div\ p + 1) \prod_{j=3}^{N} b_j.$$

Calculons C_k :

$$C_k = \sum_{j=k+1}^{N} b_j\, x_k + r_{cc}\, \frac{p-1}{4}\, x_k + \sum_{j=1}^{k} (\frac{T}{b_j}\ div\ p + 1)\, b_j^2 + r_{cc}\, \frac{p-1}{4} \sum_{j=2}^{k} (\frac{T}{b_j}\ div\ p + 1)\, b_j$$

Avec

$$x_k = (\prod_{j=1}^{k} b_j\ div\ p + 1) \prod_{j=k+1}^{N} b_j$$

Remarque :

La valeur de C_k dépend fortement de l'ordre lexicographique choisi. On rappelle que le choix de l'ordre lexicographique ne change en rien le problème probabiliste. Dans ce qui suit nous allons choisir le meilleur ordre lexicographique pour résoudre notre problème.

6.4.4.3 Critère de choix de l'ordre lexicographique

Etant donné le coût d'une affectation C_k, nous allons déduire l'ordre lexicographique le mieux adapté qui fait décroitre le coût C_{k+1} par rapport à C_k. Pour ce faire, calculons la différence Δ_k=$C_{k+1} - C_k$ pour k tel que $x_k \neq 0$. Notons que si Δ_k est négative, l'affectation C_{k+1} est meilleure que l'affectation C_k.

$$\Delta_k = (\sum_{j=k+2}^{N} b_j\, x_{k+1}) - (\sum_{j=k+1}^{N} b_j\, x_k) + r_{cc}\, \frac{p-1}{4}\, (x_{k+1} - x_k)$$

$$+ (\sum_{j=1}^{k+1}(\frac{T}{b_j}\ div\ p\ +\ 1)\ b_j^2) - (\sum_{j=1}^{k}(\frac{T}{b_j}\ div\ p\ +\ 1)\ b_j^2)$$
$$+ r_{cc}\ \frac{p-1}{4}\ \{(\sum_{j=2}^{k+1}(\frac{T}{b_j}\ div\ p\ +\ 1)\ b_j) - (\sum_{j=2}^{k}(\frac{T}{b_j}\ div\ p\ +\ 1)\ b_j)\}$$

Après développement, on obtient :

$$\Delta_k = -\ b_{k+1}\ x_k\ +\ (x_{k+1}\ -\ x_k\ )\ (\sum_{j=k+2}^{N}\ b_j)\ +\ r_{cc}\ \frac{p-1}{4}\ (x_{k+1}\ -\ x_k)$$
$$+\ (\ \frac{T}{b_{k+1}}\ div\ p\ +\ 1)\ b_{k+1}^2\ +\ r_{cc}\ \frac{p-1}{4}\ (\ \frac{T}{b_{k+1}}\ div\ p\ +\ 1)\ b_{k+1}$$

Calculons la quantité $(x_{k+1}\ -\ x_k)$:

$$x_{k+1}\ -\ x_k\ =\ ((\prod_{j=1}^{k+1}\ b_j)\ div\ p\ +\ 1)(\ \prod_{j=k+2}^{N}\ b_j)\ -\ ((\prod_{j=1}^{k}\ b_j)\ div\ p\ +\ 1)(\ \prod_{j=k+1}^{N}\ b_j)$$

Après développement, on obtient :

$$x_{k+1}\ -\ x_k\ =\ (\ \prod_{j=k+2}^{N}\ b_j)(1\ -b_{k+1})\ +\ (\ \prod_{j=k+2}^{N}\ b_j)(\prod_{j=1}^{k+1}\ div\ p)\ -\ (\ \prod_{j=k+1}^{N}\ b_j)(\prod_{j=1}^{k}\ b_j\ div\ p)$$

On note

$$D\ =\ (\ \prod_{j=k+2}^{N}\ b_j)(\prod_{j=1}^{k+1}\ div\ p)\ -\ (\ \prod_{j=k+1}^{N}\ b_j)(\prod_{j=1}^{k}\ b_j\ div\ p)$$

Pour calculer D nous allons comparer les quantités suivantes :

$$E\ =\ b(a\ div\ p)$$
$$F\ =\ (ba)\ div\ p$$

Avec les constantes a, b, p des entiers naturels. On peut toujours écrire :

$$a\ =\ p\ q_a\ +\ r_a$$

On a alors E qui s'écrit :

$$b(a\ div\ p)\ =\ b\ q_a.$$

D'autre part :

$$b\,a \;=\; p\,b\,q_a \;+\; b\,r_a.$$

Examinons le terme $(b\,r_a)$. La constante r_a est le reste de la division de a par p, on a $0 \leq r_a < p$, donc on peut écrire :

$$r_a\,b \;=\; (p \;-\; m)b \quad , 1 \leq m \leq p$$

Nous avons :

$$p\,b \;-\; m\,b \;=\; p\,b - m\,b \;+\; p\,l \;-\; p\,l \quad , l \in \mathcal{N}$$

On arrange les termes de la formule précédente et on obtient :

$$p\,b \;-\; m\,b \;=\; p(b \;-\; l) \;+\; p\,l \;-\; m\,b$$

Vérifions que $(\; p\,l \;-\; m\,b)$ est bien le reste de la division de $r_a\;b$ par p. En effet si $(\; p\,l \;-\; m\,b)$ est le reste de la division de $r_a\;b$ par p on a le système d'inéquations suivant:

$$S\begin{cases} 0 \leq p\,l \;-\; m\,b < p \\ 1 \leq m \leq p \\ 1 \leq l \leq b \end{cases}$$

Le système S peut s'écrire :

$$S'\begin{cases} p(l \;-\; 1) \leq m\,b \leq p\,l \\ 1 \leq m \leq p \\ 1 \leq l \leq b \end{cases}$$

En Multipliant la deuxième inéquation de S' par b, on obtient :

$$S''\begin{cases} p(l \;-\; 1) \leq m\,b \leq p\,l \\ b \leq m\,b \leq p\,b \\ 1 \leq l \leq b \end{cases}$$

D'après S'' en déduit que pour tout l tel que $1 \;\leq\; l \;\leq\; b$, la première inéquation est vraie, donc $(\; p\,l \;-\; m\;b)$ est le reste de la division de $r_a\;b$ par p. La quantité $F \;=\; (b\,a)\;div\;p$ est égale :

$$(b\ a)\ div\ p\ =\ b\ q_a\ +\ (b\ -\ l)$$

On a alors :

$$(b\ a)\ div\ p\ =\ b(a\ div\ p)\ +\ (b\ -\ l),\ \ 1\ \leq\ l\ \leq\ b \tag{6.26}$$

On utilisant l'équation 6.26, calculons :

$$(\prod_{j=k+2}^{N} b_j)(\prod_{j=1}^{k+1} b_j\ div\ p)\ =\ T\ div\ p\ -\ \prod_{j=k+2}^{N} b_j\ +\ l_1\ ,\quad 1\ \leq\ l_1\ \leq\ \prod_{j=k+2}^{N} b_j$$

De même, nous avons :

$$(\prod_{j=k+1}^{N} b_j)(\prod_{j=1}^{k} b_j\ div\ p)\ =\ T\ div\ p\ -\ \prod_{j=k+1}^{N} b_j\ +\ l_2\ ,\quad 1\ \leq\ l_2\ \leq\ \prod_{j=k+1}^{N} b_j$$

De ce fait, la quantité D devient :

$$D\ =\ (\prod_{j=k+2}^{N} b_j)(b_{k+1}\ -\ 1) + (l_1\ -\ l_2)$$

L'expression $(x_{k+1}\ -\ x_k)$ devient :

$$(x_{k+1}\ -\ x_k)\ =\ (l_1\ -\ l_2)$$

Avec

$$1\ \leq\ l_1\ \leq\ \prod_{j=k+2}^{N} b_j$$

$$1\ \leq\ l_2\ \leq\ \prod_{j=k+1}^{N} b_j$$

En combinant les deux inéquations précédentes, on obtient :

$$1\ -\ \prod_{j=k+1}^{N} b_j\ \leq\ (l_1\ -\ l_2)\ \leq\ \prod_{j=k+2}^{N} b_j\ -\ 1 \tag{6.27}$$

Injectons la valeur de $(x_{k+1} - x_k)$ dans l'expression de Δ_k :

$$\Delta_k = (-b_{k+1})(\prod_{j=1}^{k} b_j \; div \; p \; + \; 1) \; (\prod_{j=k+1}^{N} b_j) \; + \; (l_1 - l_2)(\sum_{j=k+2}^{N} b_j \; + \; r_{cc} \; \frac{p-1}{4})$$

$$+ \; (\frac{T}{b_{k+1}} \; div \; p \; + \; 1) \; b_{k+1}^2 \; + \; r_{cc} \; \frac{p-1}{4} \; (\frac{T}{b_{k+1}} \; div \; p \; + \; 1) \; b_{k+1}$$

Pour développer l'expression Δ_k , calculons les quantités suivantes :

$$A \; = \; (\prod_{j=k+1}^{N} b_j)(\prod_{j=1}^{k} b_j \; div \; p)$$

$$B \; = \; (b_{k+1})(\frac{T}{b_{k+1}} \; div \; p)$$

Nous avons les expressions A et B qui s'écrivent de la forme :

$$A \; = \; b'(a' \; div \; p)$$

$$B \; = \; b''(a'' \; div \; p)$$

Avec :

$$b' \; = \; (\prod_{j=k+1}^{N} b_j) \; , \; a' \; = \; (\prod_{j=1}^{k} b_j)$$

$$b'' \; = \; b_{k+1} \; , \; a'' \; = \; (\frac{T}{b_{k+1}})$$

D'après l'analyse précédente, l'équation 6.26 s'écrit :

$$(b \; a) \; div \; p \; = \; b \; q_a \; + \; b \; - \; l$$

$$1 \; \leq \; l \; \leq \; b$$

On déduit que :

$$A \; = \; (b' \; a') \; div \; p \; - \; b' \; + \; l'$$

$$B \; = \; (b'' \; a'') \; div \; p \; - \; b'' \; + \; l''$$

On obtient donc :

$$A \; = \; (T) \; div \; p \; - \; \prod_{j=k+1}^{N} b_j \; + \; l'$$

$$B \; = \; (T) \; div \; p \; - \; b_{k+1} \; + \; l''$$

Avec

$$1 \leq l' \leq \prod_{j=k+1}^{N} b_j$$
$$1 \leq l'' \leq b_{k+1}$$

Remplaçons les quantités A et B dans l'expression de Δ_k :

$$\begin{aligned}\Delta_k = (-b_{k+1})(T\ div\ p - \prod_{j=k+1}^{N} b_j + l') - (b_{k+1} \prod_{j=k+1}^{N} b_j) \\ + (l_1 - l_2)(\sum_{j=k+2}^{N} b_j + r_{cc}\frac{p-1}{4}) \\ + (b_{k+1})(T\ div\ p - b_{k+1} + l'') + b_{k+1}^2 \\ + r_{cc}\frac{p-1}{4} b_{k+1} + r_{cc}\frac{p-1}{4}(T\ div\ p - b_{k+1} + l'')\end{aligned}$$

Finalement , on obtient :

$$\begin{aligned}\Delta_k = - b_{k+1}\, l' + (l_1 - l_2)(\sum_{j=k+2}^{N} b_j + r_{cc}\frac{p-1}{4}) + (b_{k+1}\, l'') \\ + (r_{cc}\frac{p-1}{4}\,(T\ div\ p)) + (r_{cc}\frac{p-1}{4}\, l'')\end{aligned}$$

Remarque

D'après la formule de Δ_k, on remarque que le premier terme est d'autant plus fortement négatifs que b_{k+1} est choisi comme le plus grand en valeur dans l'ensemble $\{b_{k+1}, b_{k+2}, \cdots, b_N\}$, on note cette valeur a_{k+1}. Le deuxième terme dépend de la quantité $(l_1 - l_2)$ et d'après l'inéquation 6.27, si $(l_1 - l_2)$ est positive, pour espérer que Δ_k soit négative il faut que la borne maximal de $(l_1 - l_2)$ qui est $(\prod_{j=k+2}^{N} b_j)$ soit la plus petite possible. En effet si la valeur $a_{k+1} \notin \{ b_{k+2}, \cdots, b_N \}$, la borne maximal a la plus petite valeur. Dans le cas où $(l_1 - l_2)$ est négative, le choix de b_{k+1} n'influence en rien sa borne minimale qui est $(1 - \prod_{k+1}^{N} b_j)$. Il est de même si $(l_1 - l_2)$ est nulle. Le troisième et cinquième terme sont des constantes positives pas très importantes. Le quatrième terme est une constante positive assez grande.

D'après l'analyse de la formule Δ_k, on déduit le lemme suivant :

Lemme 3 *L'ordre lexicographique trie décroissant des tailles des matrices du modèle de b_1 à b_N permet de trouver l'affectation de coût minimun.*

Démonstration :

Soit un ordre lexicographique $\{b_1, b_2 \cdots, b_N\}$, l'idée et de renuméroter la suite $\{b_i\}_{i=1,\ N}$ de telle manière que pour chaque $k > kin$ (kin est tel que $x_{kin} \neq 0$), C_k à le meilleur coût. Or on vient de voir ci-dessus que pour réaliser ce but, il suffit que le terme de la suite de rang $k+1$ noté a_{k+1} doit être le plus grand élément de l'ensemble $\{b_{k+1}, b_{k+2}, \cdots, b_N\}$ des éléments qui ne sont pas encore renuméroter. Par ce procédé, la suite $\{a_i\}_{i=kin,N}$ est décroissante. Pour la renumérotation des $(kin-1)$ premiers termes de la suite $\{a_i\}_{i=1,kin-1}$ l'argument du choix est le suivant :

Notre objectif est déja d'assurer un coût C_{kin} le meilleur possible. Nous avons vu au paragraphe 6.4.4.1 que pour avoir un meilleur coût, il faut avoir le plus de sous-blocs possible (l'équilibrage de charge) avec kin le plus petit possible (nombre *d'échange total*). Pour réaliser cela , il suffit de choisir les $(kin-1)$ premiers termes de la suite comme les plus grands des b_i $i = 1, N$. Ainsi $(kin-1)$ est minimum et le nombre de sous-blocs correspondants au niveau $kin-1$ est maximum.

Remarque :

Le résultat de ce lemme nous réduit la recherche de l'affectation minimum d'un rapport $N!$. On remarque que la valeur absolue des termes négatifs de Δ_k sont strictement décroissants en fonction de k (ordre lexicographique trie décroissant des b_k). la valeur de Δ_k pour k de 1 à k_0 est négative puis devient positive pour k supérieur à k_0. Comme Δ_k est l'accroissement de la fonction C_k, on déduit que la fonction C_k est décroissante, atteint un minimum et recroit (voir annexe F). De ce fait la recherche de C_k minimum devient une simple recherche dichotomique.

Dans l'annexe F, nous avons comparé le coût des affectations selon plusieurs ordre lexicographique aléatoire, l'ordre lexicographique décroissant des $\{b_i\}_{i=1,N}$ donne toujours le meilleur coût.

6.4.4.4 Algorithme de la méthode mixte:

Une fois que le grain de parallélisme à été choisi (affectation des sous-blocs du vecteur π correspondant au niveau k_0 de l'arbre), la méthode mixte est présentée dans l'algorithme suivant :

Début
  Calcul de k_0 ;
  Calcul des positions des composantes pour l'affectation k_0

{* On effectue $N - k_0$ étape en utilisant la méthode du paragraphe 6.2.6 *}
Pour *etape* $= k_0 + 1,\ N$ faire
Algorithme décrit dans le paragraphe 6.2.6
FinPour
{* On effectue k_0 étape en utilisant la méthode du paragraphe 6.2.5 *}
Pour *etape* $= 1,\ k_0$ faire
Calcul de la permutation et de l'affectation *etape*;
Echange total pour effectuer le mélange parfait *etape*;
Produit des blocs de $s\pi^{\sigma_2}$ de taille b_{etape} par la matrice M_{etape}
FinPour
Fin

Remarque

Il faut noter que pour effectuer la somme $\Sigma_{i=1}^{C}\ \bigotimes_{i=1}^{N} M_{ti}$, il suffit d'exécuter l'algorithme ci-dessus pour chacun des C termes avec accumulation des résultats. Cependant, l'ordre de rangement des composantes dans le vecteur résultant est différent de l'ordre initial (les k_0 derniers mélange parfait de l'algorithme mixte, déplacent les composantes). Dans le cadre de l'intégration de cette méthode de multiplication vecteur matrice (descripteur d'un (RAS)) dans un processus itératif, une étape de communication du type échange total vient ensuite pour replacer les composantes du vecteur résultat selon la configuration initiale. Cette procédure de replacement des composantes selon l'ordre initial après les k_0 mélanges parfait, fait l'objet du paragraphe suivant.

Procédure de calcul des adresses des composantes pour retrouver l'ordre initial après k_0 mélanges parfaits :

Le calcul des positions des éléments pour l'affectation k_0 sachant que les composantes du vecteurs ont été permuté successivement par les mélangés parfaits de 1 à k_0 se base sur ce qui suit :

On rappelle que pour effectuer un mélange parfait à une étape i, le vecteur π est vu comme b_i blocs de taille B_i éléments (voir le paragraphe 6.2.3). L'ordre des composantes dans le vecteur π correspond au parcours ligne par ligne des b_i blocs. Après la $(b_i,\ Bi)$-permutation, l'ordre des composantes dans le vecteur π^{σ_2} correspond au parcours colonne par colonne des b_i blocs. Pour avoir une vision des déplacements des éléments nous allons regarder comment sont positionner les composantes du vecteur π (ordonné selon un ordre lexicographique fixé) après chaque mélange parfait :

- Après le premier mélange parfait :

Etant donné qu'après la permutation les composantes (numérotées selon un ordre lexicographique donné) sont ordonnées selon un parcours colonne par colonne des b_1 blocs. On déduit que deux composantes consécutives x_l et x_{l+1} dans le vecteur π seront dans l'ordre respectif x_{l_1} et $x_{(l_1+b_1)[T]}$ dans le vecteur $\pi^{\sigma_2^{(1)}}$ pour tout $l \in [\,1..T\,]$.

- Après le deuxième mélange parfait :

  La composante x_{l_1} et sa composante consécutif dans $\pi^{\sigma_2^{(1)}}$ qui est x_{l_1+1} seront dans l'ordre respectif x_{l_2} et $x_{(l_2+b_2)[T]}$ dans le vecteur $\pi^{\sigma_2^{(2)}}$. Si on applique ce même raisonnement sur toutes les composantes successives de x_{l_1} jusqu'à la composante $x_{(l_1+b_1)[T]}$, cette dernière serait localisées par $x_{(l_1 b_1\ b_2)[T]}$ dans le vecteur $\pi^{\sigma_2^{(2)}}$.

- Après le i^{eme} mélange parfait :

  On déduit que deux composantes consécutives dans π seront consécutives par un pas de $\prod_{j=1}^{i} b_j$ modulo la taille T dans le vecteur $\pi^{\sigma_2^{(i)}}$.

Remarque

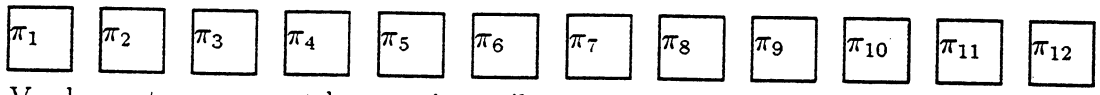
D'après ce qu'on vient de voir, une façon de retrouver l'ordre initial des composantes du vecteur π après k_0 mélanges parfaits, est de parcourire le vecteur $\pi^{\sigma_2^{(k_0)}}$ par pas de $\prod_{j=1}^{k_0} b_j$, ce qui est décrit par l'algorithme suivant.

Algorithme

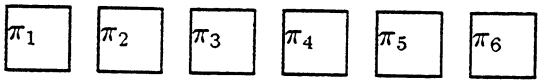
Début
$tb = \prod_{j=1}^{k_0}$;
$nbtb = T\ div\ tb$;{* nombre de blocs de taille tb *}
  Pour $posit = 1,\ T$ faire
    $j = (posit - 1) * tb + 1$;
    $l = j\ mod\ T$;
    $m = posit\ div\ nbtb$;
    $kk = l\ +\ m$;
    Si ($posit\ mod\ nbtb = 0$) Alors $kk = kk - 1$;
    $adr[posit] = kk$;
    {* adr[posit] contient la position de la composantes numéro
    $posit$ (l'ordre initial) dans le vecteur $\pi^{\sigma_2^{k_0}}$ *}
  FinPour
Fin

Un exemple illustrant le déplacement des composantes par les mélanges parfaits successifs est donné dans la figure 6.14.

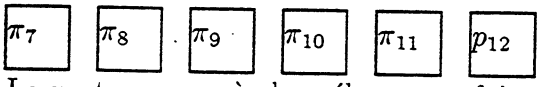
Le rangement des composantes dans le vecteur π avant le premier mélange parfait est :



Vu du vecteur π avant le premier mélange parfait comme $b_1 = 2$ blocs de $B_2 = 6$ éléments

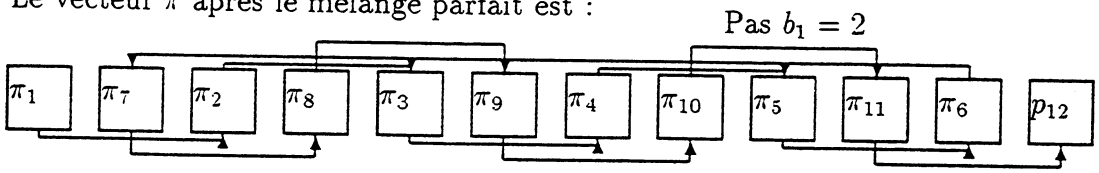


Bloc 1

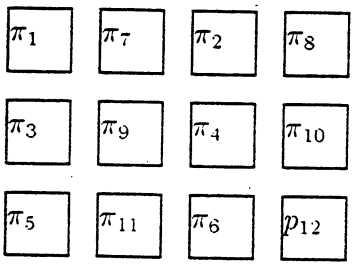


Bloc 2

Le vecteur π après le mélange parfait est :



Vu du vecteur π avant le deuxieme mélange parfait comme $b_2 = 3$ blocs de $B_2 = 4$ éléments



Le vecteur π après le mélange parfait est :

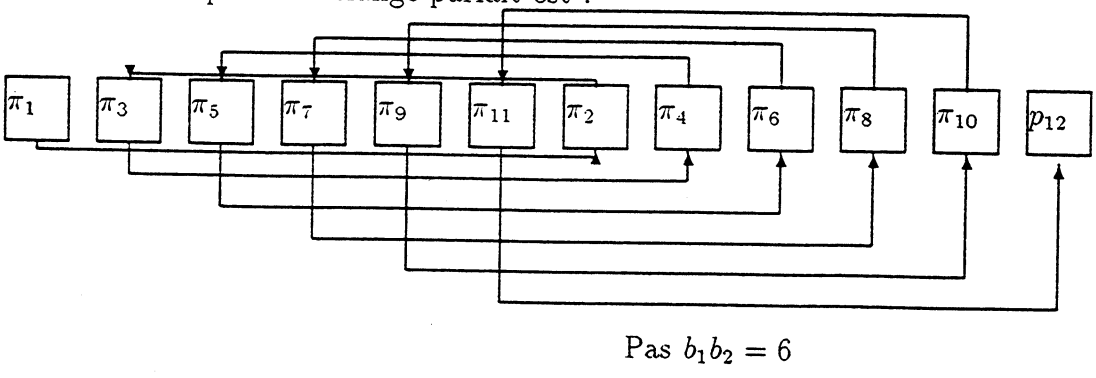


Figure 6.14: Schéma du déplacement des composantes par les mélanges parfaits successives

Dans ce qui suit, on va évaluer l'accélération de la parallélisation de cette approche utilisant la méthode mixte.

Accélération de la méthode mixte

On rappelle que dans cette approche de parallélisation du terme, on veut exploiter le maximum de mémoire pour traiter de très gros problème. Pour évaluer notre stratégie de parallélisation, nous définissons l'accélération de la méthode comme le rapport du temps du meilleur algorithme séquentiel sur le temps parallèle de la méthode mixte (pour un seul terme) :

$$Accel_{mixte} = \frac{T \sum_{j=1}^{N} b_j}{\frac{T \sum_{j=1}^{N} b_j}{p} + (k_0) \frac{(p-1)}{4} \frac{T\, r}{p}}$$

Le numérateur représente le coût du produit sur un processeur. Le deuxième terme du dénominateur correspond au coût des communications pour effectuer les k_0 mélanges parfaits en utilisant la procédure d'échange total. La constante r représente le rapport d'une opération de calcul élémentaire sur le coût de communication d'un réel.

On remarque que le coût des communication dans la méthode mixte est réduit d'un facteur de k_0 au lieu de N dans la méthode décrite dans le paragraphe 6.4.2.2.

6.5 Deuxième approche : parallélisation par distribution des termes de la somme $\pi\ [\Sigma_{t=1}^{C}\ \ (\otimes_{i=1}^{N} M_{ti})]$

Dans cette approche, nous allons paralléliser la somme $\pi[\Sigma_{t=1}^{C}\ \ (\otimes_{i=1}^{N} M_{ti})]$. On suppose que C le nombre de terme de la somme est inféniment grand devant p le nombre de processeurs.

L'idée est d'assigner à chaque processeur de la machine $\frac{C}{p}$ termes de la somme. A chaque instant de l'algorithme, on exécute en parallèle p termes. Dans cette approche, le calcul des termes n'est pas parallélisé. Etant donné que chaque processeur contient le vecteur de probabilités en entier, on ne génére pas de communication pour effectuer les mélanges parfaits. Après que chaque processeur termine l'exécution des termes qui lui sont assignés, une étape de communication du type *échange total* vient ensuite pour former la somme totale sur chaque processeur (pour éventuellement une autre itération). Le coût de cette dernière est donné par :

$$Comm_{somm} = \frac{p-1}{4} T\ r$$

L'accélération de cette approche de parallélisation est donnée dans le paragraphe suivant.

Accélération de la parallélisation de la somme

On définit de même que précédemment, l'accélération de la méthode par :

$$Accel_{somm} = \frac{C\ T\ \sum_{j=1}^{N} b_j}{\frac{C\ T\ \sum_{j=1}^{N} b_j}{p} + \frac{(p-1)}{4} T\ r}$$

Le numérateur de cette expression, représente le coût en séquentiel des C termes. Le deuxième terme du dénominateur représente les communications pour effectuer la somme sur tous les processeurs de la machine.

On écrit sous une autre forme l'expression de l'accélération :

$$Accel_{somm} = \frac{1}{\frac{1}{p} + \frac{p\ r}{C\ \sum_{j=1}^{N} b_j}}$$

Il faut noter que pour C infini, l'accélération de la méthode tend vers p.

Remarque :

L'inconvénient de cette méthode est que chaque processeur doit avoir le vecteur de probabilités en entier. Cela limite la taille des problèmes à résoudre. Ce qui présente un sérieux désavantage.

6.6 Troisième approche : parallélisation de la somme et des termes de la somme

Dans ce cas nous allons étudier une méthode de parallélisation qui est un juste milieu entre les deux méthodes précédentes. On veut exploiter le plus de parallélisme possible de l'expression $\pi[\Sigma_{t=1}^{C} \otimes_{i=1}^{N} M_{ti}]$. L'idée est de former g groupes de processeurs, chaque groupe réalise la parallélisation d'un terme (selon la méthode mixte). Ainsi, nous allons traiter g termes simultanément. Une étape de communication vient enfin pour effectuer la somme totale qui est distribué dans les g groupes. L'efficacité de cette parallélisation dépend du choix des algorithmes et des topologies de communication qu'on doit utiliser pour résoudre les problèmes suivants :

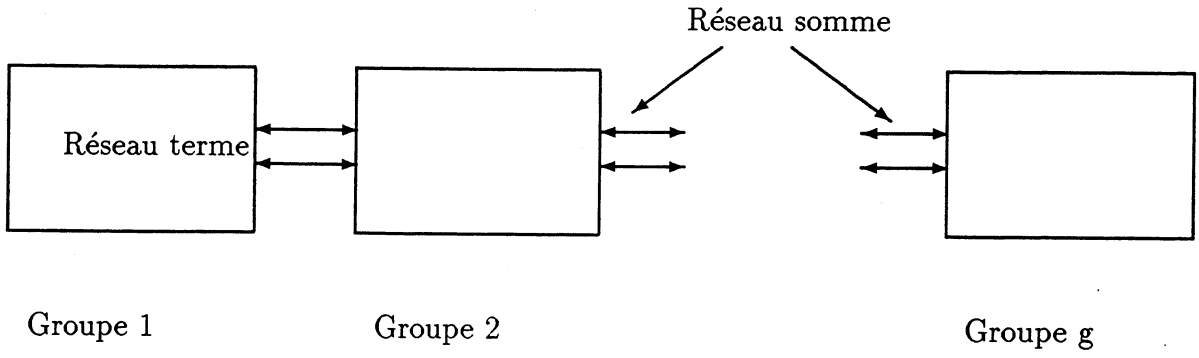


Figure 6.15: Réseaux somme et terme

- Communications pour effectuer les mélanges parfaits dans chaque groupe (réseau de parallélisation des termes) (figure 6.15).
- Communication pour effectuer la somme totale des termes (réseau somme) (figure 6.15)
- Le nombre de groupe de processeur (accélération de la somme)
- Le nombre de processeur dans un groupe $np_g = \frac{p}{g}$ (accélération des termes).

Dans ce qui suit nous allons proposer différentes topologies de connexion pour résoudre au mieux les points ci-dessus.

6.6.1 Réseaux toriques pour effectuer les termes et la somme

La motivation principal du choix du réseau de parallélisation des termes (réseau terme) comme un tore carré est justifier par le fait qu'on sait résoudre l'échange total en temps optimal sur le tore (figure 6.16).

Pour effectuer la somme global, il suffit que le réseau soit configurer en un tore de p processeurs pour effectuer un échange total afin de former la somme total sur chaque processeur (figure 6.17).

Evaluation du coûts des communications dans cette solution

- Le coût d'un mélange parfait est :

$$CoutReseau_{term_1} = \frac{Np_g - 1}{4} \frac{T\,r}{Np_g} \simeq \frac{T\,r}{4}$$

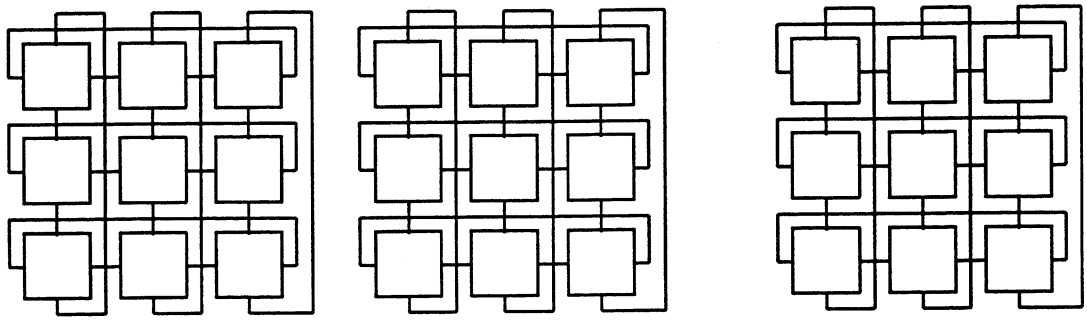


Figure 6.16: Réseaux de communication pour effectuer les termes

- Le coût de la somme total :

$$CoutReseau_{somm_1} = \frac{p-1}{4}\frac{T\,r}{Np_g} \simeq \frac{g\,T\,r}{4}$$

Remarque :

Il faut faire une reconfiguration dynamique de la machine pour passer du réseau terme au réseau somme. Ce qui n'est pas possible pour l'instant sur notre machine.

6.6.2 Réseaux semi-toriques pour effectuer les termes et réseaux anneaux pour la somme

Dans cette stratégie chaque groupe de processeur est configuré en un semi-tore (grille fermé sur une seule dimension)(figure 6.18). Ces semi-tores sont mis côte à côte pour former le réseau somme qui est constitué des anneaux horizontaux (voir la figure 6.19).

Evaluation du coûts des communications dans cette solution

- Le coût d'un mélange parfait est :

$$CoutReseau_{term_2} = \frac{T\,r}{2}$$

Ce qui correspond au temps de l'échange total sur le semi-tore [15].

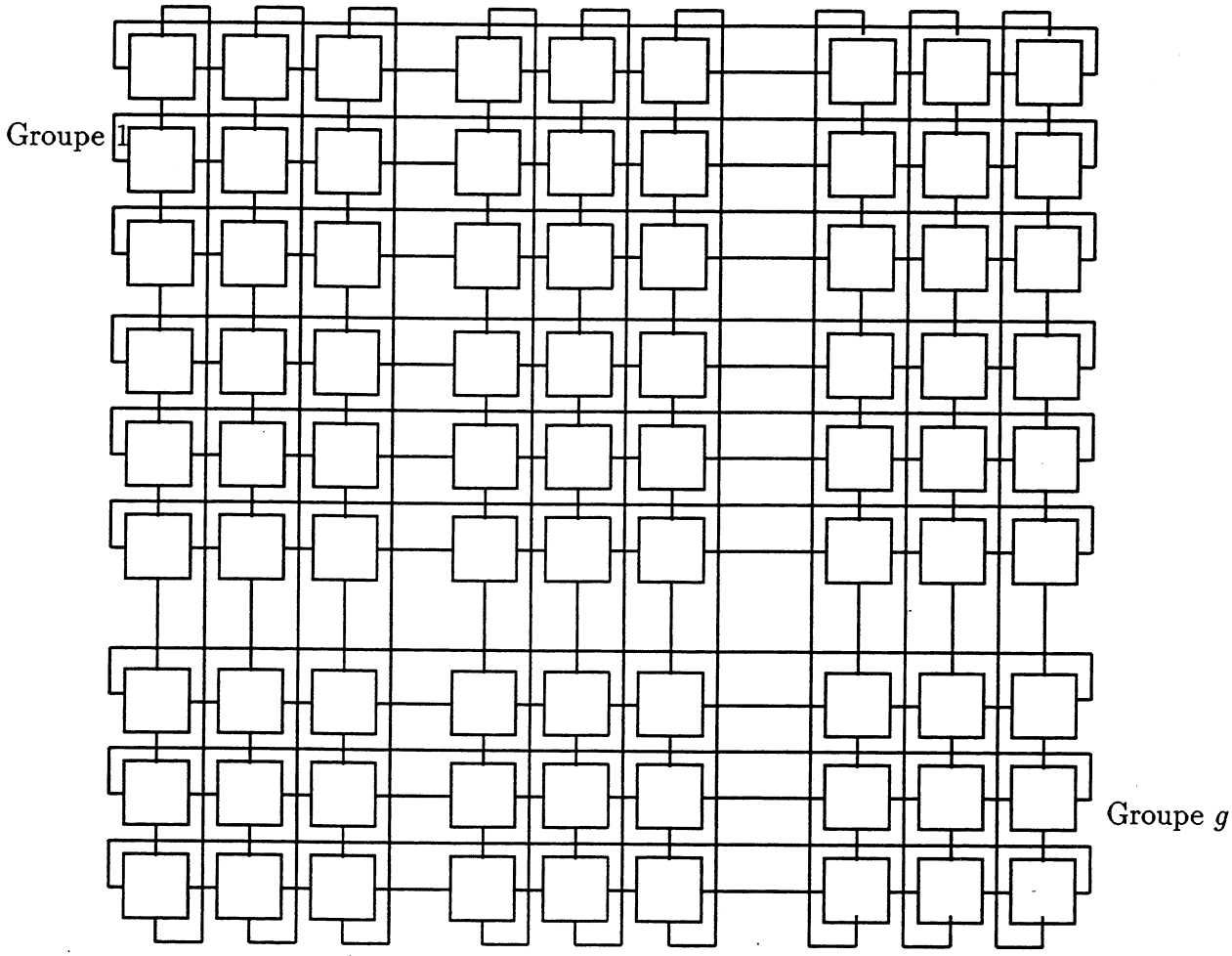


Figure 6.17: Réseau de communication pour effectuer la somme total

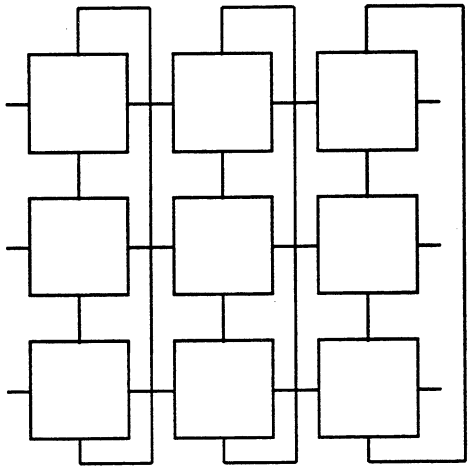


Figure 6.18: Réseau semi-tore pour effectuer les termes

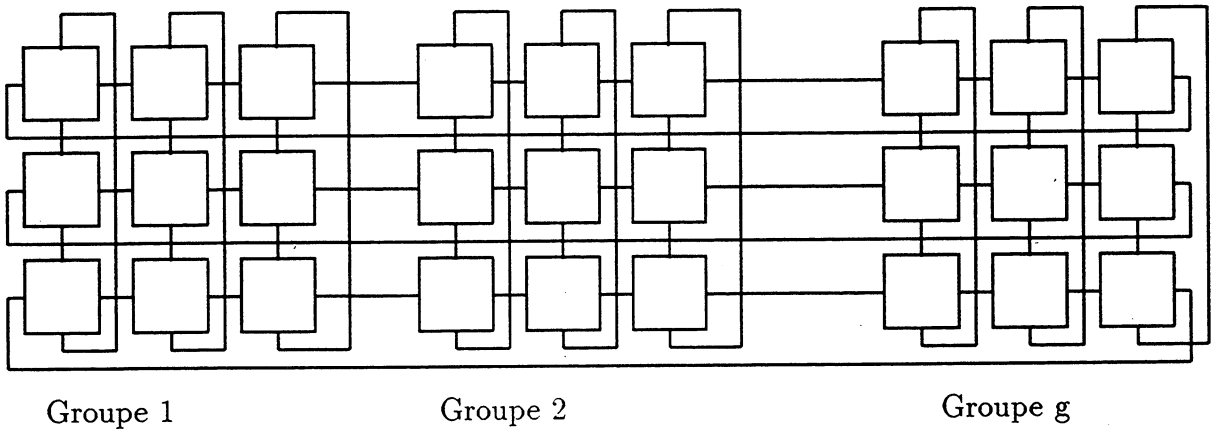


Figure 6.19: Réseau anneaux de communication pour effectuer la somme total

- Le coût de la somme total : si on note $l_g \times L_g$ la taille d'un semi-tore, on a :

$$\frac{p}{g} = l_g \, L_g.$$

Les anneaux horizontaux sont de taille $g \, L_g$.

$$CoutReseau_{somm_2} = \frac{g \, L_g}{2} \frac{T \, r}{Np_g} = \frac{g^2 \, L_g \, T \, r}{2 \, p}$$

6.6.3 Réseaux anneaux pour effectuer les termes et réseaux anneaux pour la somme

Dans cette stratégie, on configure notre machine en tore. Les anneaux horizontaux, constituent les groupes (parallélisation des termes), les anneaux verticaux constituent le réseau somme (figure 6.20).

Evaluation du coûts des communications dans cette solution

- Le coût d'un mélange parfait est :

$$CoutReseau_{term_3} = \frac{T \, r}{2}$$

Temps d'un échange total sur un anneau de taille $\frac{p}{g}$.

- Le coût de la somme total :

$$CoutReseau_{somm_3} = \frac{g}{2} \frac{T \, r}{Np_g} = \frac{g^2 \, T \, r}{2 \, p}$$

6.6.4 Comparaison entre les trois stratégies et choix de la meilleure solution

La première stratégie donne le meilleur temps pour effectuer un mélange parfait, mais le temps pour effectuer la somme total est très mauvais. De plus, la machine doit changer de topologie pendant l'exécution. Par contre la deuxième et troisième sont des solutions à topologie fixe. La stratégie 3, s'avère meilleure car le temps pour effectuer les mélanges parfaits est le même que dans la stratégie 2, mais avec un algorithme plus simple. De plus le temps pour réaliser la somme total est L_g fois plus petit que dans la stratégie 2.

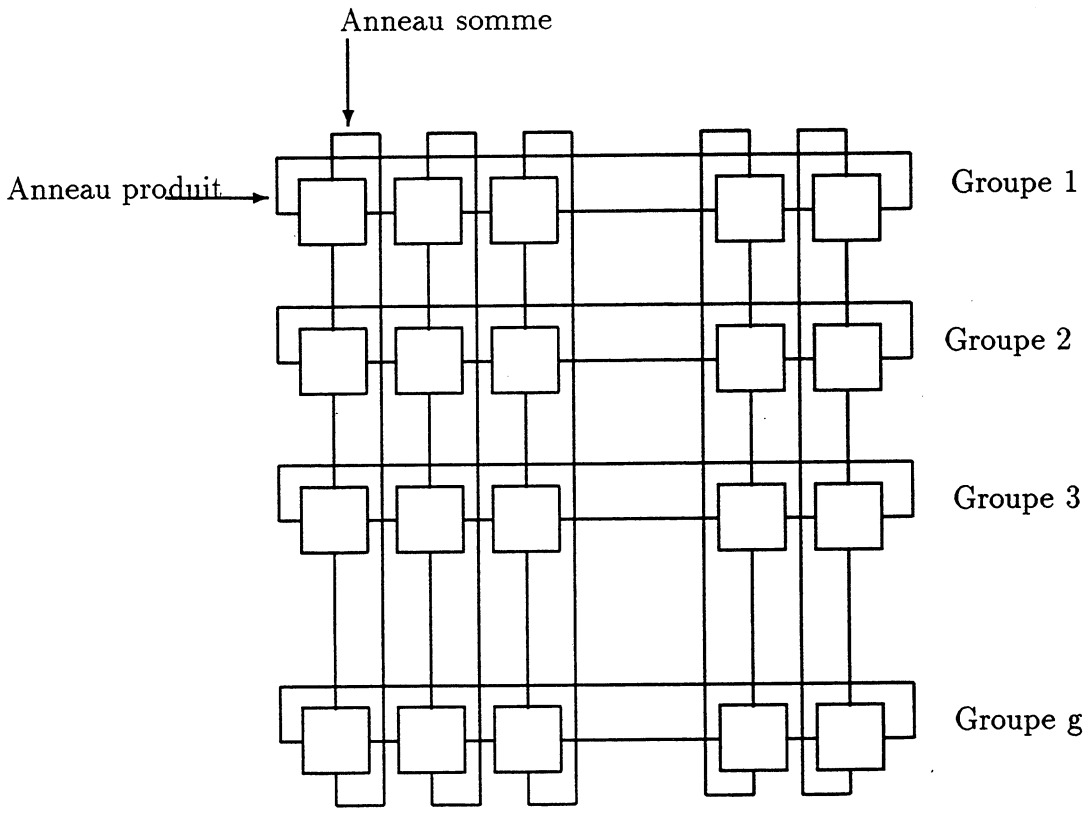


Figure 6.20: Deux réseaux anneaux de communication pour effectuer les termes de la somme

6.6.5 Etude de l'accélération pour la solution choisie

Dans ce paragraphe, nous allons évaluer l'approche de parallésation qui est basé sur la stratégie 3.

$$Accel_{strat3} = \frac{C\ tseq}{\frac{C}{g}\frac{tseq}{Np_g} + \frac{C}{g}\ Comm_{term} + Comm_{somm}}$$

- C représente le nombre de termes de la somme.
- g est nombre de groupe de processeur.
- np_g est le nombre de processeur par groupe.

$$np_g = \frac{p}{g}$$

- $Comm_{term}$ est le temps de communication pour faire (k_0) mélanges parfaits :

$$Comm_{term} = k_0\ (\frac{T\ r}{2})$$

- $Comm_{somm}$ est le temps de communication pour faire une somme total sur chaque processeur :

$$Comm_{somm} = \frac{g^2\ T\ r}{2\ p}$$

- $tseq$: le temps d'exécution d'un terme en séquentiel.

$$tseq = T \sum_{j=1}^{N} b_j$$

Ecrivons sous une autre forme l'expression de l'accélération :

$$Accel_{strat3} = \frac{1}{\frac{1}{p} + \frac{k_0\ r}{2\ g\ \sum_{j=1}^{N} b_j} + \frac{g^2\ r}{2\ C\ p\ \sum_{j=1}^{N} b_j}}$$

$$\lim_{c \to \infty} Accel_{strat_3} = \frac{p}{1 + \frac{k_0\ np_g\ r}{2\ \sum_{j=1}^{N} b_j}} \qquad (6.28)$$

D'après l'expression 6.28, on remarque :

- L'accélération limite est inférieur à p.
- L'accélération est d'autant plus mauvaise que np_g le nombre de processeur dans un groupe est grand.

Remarque :

Cette approche de parallélisation ne marche bien que pour np_g le plus petit possible, ce qui implique que les problèmes qu'on peut traiter efficacement par cette méthode sont relativement petits. Cette solution rejoint l'approche de parallélisation de la somme. On rappelle que l'accélération de l'approche de parallélisation de la somme (paragraphe 6.5) dans les même conditions atteint p et est donc meilleure que la solution proposé dans ce paragraphe.

6.7 Comparaison entre les 3 approches de parallélisation et choix de la plus appropriée

On rappelle que notre objectif est de résoudre des problèmes de grande taille de l'ordre de plusieurs **millions** d'états. Or on vient de voir que la deuxième approche (parallélisation par distribution des termes de la somme) est très efficace mais la taille des problèmes qu'on peut traiter est réduite à cause des restriction matérielles et logicielles de la machine. Nous avons vu aussi que la troisième approche qui consiste à distribuer les termes de la somme sur des groupes de processeurs de telle façon que chaque groupe parallélise le terme courant n'est efficace que si le nombre de processeur par groupe est le plus petit possible. Cette approche tend vers la deuxième approche, on arrive donc à la même conclusion. La seule approche qui supporte les problèmes de grandes tailles est la première approche.

Dans ce qui suit, nous allons proposer deux améliorations concernant la première approche. La première amélioration consiste à modifier la procédure qui fait les mélanges parfaits. La deuxième amélioration utilise le recouvrement des communications par les calculs.

6.8 Amélioration et expérimentation de l'approche choisie

Dans le cadre de l'amélioration de la première approche, nous allons commencer par la modification de la procédure de communication qui effectue le mélange parfait, en se

basant sur les résultats des propriétés des mélanges parfaits décrites dans le paragraphe 6.2.3 et la procédure de communication échange total partiel personnalisé, développée dans le paragraphe 2.6. La deuxième amélioration utilise l'effet de recouvrement des communications par des calculs.

6.8.1 Amélioration de la méthode en utilisant les propriétés des mélanges parfaits

1. On rappelle que dans les conditions de divisibilité suivante :

   - T est divisible par p.
   - B_i est divisible par p.

   On n'a la formule suivante :

   $$p_a = (p_d\ b_i + c)\ [p] \quad 0 \leq c < b_i$$

   qui exprime que dans chaque phase i du mélange parfait, chaque processeur, doit envoyer les éléments de son sous-vecteur $s\pi$ à b_i processeurs.

2. On rappelle aussi, qu'avant le mélange parfait les composantes affectées à un processeur sont formés de groupes (gcc) (voir le paragraphe 6.2.3). Nous savons aussi que la taille d'un groupe (nombre de composantes consécutives qui vont aller sur un processeur donné après le mélange parfait) est exactement

   $$pas = \frac{(\frac{T}{p})}{b_i}$$

   dans le cas de bonne divisibilité de T, B_i par p. Dans ce cas là :
   Les $\frac{taillevect}{b_i}$ premiers éléments (premier groupe $g_{Numproc,p_{d.b_i}}$ vont migrer sur le processeur $p_{d.b_i}$, les $\frac{taillevect}{b_i}$ suivants vont migrer sur le processeur $p_{d.b_i+1}$ et ainsi de suite jusqu'aux derniers $\frac{taillevect}{b_i}$ qui vont migrer sur le processeur $p_{d.bi+b_i-1}$.
   Dans le cas de non divisibilité de T, B_i par p, la notion de groupe (gcc) reste valable par contre le nombre de groupes diffère d'un processeur à un autre (selon l'équilibrage de la charge).

L'amélioration de la réalisation des mélanges parfaits, se base sur l'utilisation de la procédure de communication échange partiel personnalisé présentée dans le paragraphe 2.6 et de ces résultats qu'on vient de voir qui sont très utiles pour la formation des sous-vecteurs à envoyer (groupes (gcc)) au processeurs destinataires. On rappelle brièvement les principes de la procédure d'échange partiel personnalisé :

- De chaque noeud est issu un arbre de recouvrement du tore, cet arbre est structuré en 4 branches linéaires (voir le paragraphe 2.3).
- Sur chaque noeud et à chaque branche de l'arbre de recouvrement, on associe une zone mémoire tempon : chaque zone mémoire va contenir les groupes $g_{Numproc,*}$ si (*) le numéro de processeur, appartient à branche correspondante.
- Sur chaque noeud, formation des 4 zones de mémoire tempons.
- Envoi sur les arbres de recouvrement des zones de mémoires tempons.

La phase de formation des sous-vecteurs (zones tempons) à envoyer est présenté dans l'algorithme suivant :

Algorithme de formation des sous-vecteurs à envoyer au processeurs à la phase i

Début
$l = indideb$;
$p_a = NumProcArriv(l)$; {* $l \in g_{Numproc,\ p_a}$ (premier groupe) *}
$pas = \frac{(\frac{T}{p})}{b_i}$; {* taille probable du groupe *}
$l = l + pas$; {* à la recherche du prochain groupe *}
Tant que ($l \leq indif$) faire
$nouvp_a = NumProcArriv(l)$;
Si $nouvp_a \neq p_a$;
Alors
Décroitre l jusqu'a trouver le premier indice l : $nouvp_a \neq p_a$;
(.) Formation des groupes destinataire du processeur p_a
$p_a = nouvp_a$
$l = l + pas$; {* à la recherche du prochain groupe *}
Sinon
$l = l + 1$
FinTanque
Fin

Avec $NumProArriv(l)$ qui représente le numéro du processeur qui va contenir la composante l après le mélange parfait. L'instruction étiquetée par (.) dans l'algorithme ci-dessus, consiste à l'ajout du nouveau groupe détecté (qui à pour destination le processeur p_a), dans le tempon mémoire qui contient les groupes destinés aux processeurs

qui sont dans la même branche de l'arbre de recouvrement du tore de racine le processeur courant.

La procédure de recopie des composantes se base sur l'algorithme recopie version 2 (voir le paragraphe 6.4.2.2). A cet algorithme, on rajoute une phase de compactage du tempon. Ainsi les tempons de données reçus par les processeurs, se voient leurs tailles diminuer de plus en plus de leurs traverser dans les branches correspondantes des arbres (procédure d'échange partiel personnalisé).

Dans le pragraphe qui suit, nous allons comparer cette nouvelle méthode pour réaliser les mélanges parfaits qu'on note "mélange parfait amélioré" , avec la méthode précédente (mélange parfait par l'utilisation de l'échange total) qu'on note par "mélange parfait naif".

6.8.1.1 Comparaison expérimentale entre le mélange amélioré et le mélange parfait naif

Dans le tableau suivant, nous comparons les performances de cette nouvelle méthode pour effectuer les mélanges parfait.

Légende du tableau

- $MelAm$, $MelNa$: représente le temps total en secondes respectivement du mélange parfait amélioré et du mélange parfait naif.
- $ComAm$, $ComNa$: représente le temps de communication pure en secondes respectivement du mélange parfait amélioré et du mélange parfait naif.
- $\%ComAm$, $\%ComNa$: représente le pourentage des communications pures dans le temps total respectivement du mélange parfait amélioré et du mélange parfait naif.

T	$MelAm$	$ComAm$	$\%ComAm$	$MelNa$	$ComNa$	$\%ComNa$	$\frac{MelNa}{MelAm}$
422500	1.402944	1.17408	83.69	2.877184	2.770048	96.28	2
562500	1.689920	1.37830	81.56	3.002048	2.866176	95.47	1.77
640000	1.914624	1.52320	79.57	3.705344	3.553447	95.91	1.93
810000	2.294976	1.79705	78.31	4.451456	4.265088	95.81	1.94
1000000	2.900992	2.317376	79.88	5.546432	5.320704	95.93	1.91

On remarque d'après le tableau ci-dessus que le temps du mélange parfait amélioré est de l'ordre de la moitie que celui du mélange parfait naif. On note aussi que le temps

de communication pure dans le mélange parfait représente environ 96% de celui-ci. Par contre le pourcentage de communication pure dans le mélange parfait amélioré n'est que d'environ 80%. Ce qui s'explique par le coût supplémentaire de la phase de compactage.

Dans le paragraphe suivant, nous allons comparer les temps des mélanges parfaits avec le temps des produits correspondants. Les résultats de cette comparaison vont nous permettre (paragraphe 6.8.1.3) d'avoir un ordre de grandeur de l'accélération esperer de cette méthode.

6.8.1.2 Comparaison expérimentale entre le temps du mélange parfait et le temps du produit correspondant

Devant la difficulté de quantification analytique du coût du mélange parfait amélioré, nous allons essayer de le quantifier expérimentalement en le comparant avec le coût du produit correspondant ($Prod$). Le résultat de cette comparaison est donné dans le tableau suivant :

T	$Prod$	$MelAm$	$\frac{MelAm}{Prod}$	$MelNa$	$\frac{MelNa}{Prod}$
422500	0.400190	1.402944	3.50	2.877184	7.20
562500	0.610048	1.689920	2.77	3.002048	4.90
640000	0.739008	1.914624	2.59	3.705344	5.00
810000	1.046848	2.294976	2.19	4.451456	4.25
1000000	1.433790	2.900992	2.02	5.546432	3.86

Ainsi, d'après cette expérimentation, on a une petite idée sur l'ordre de grandeur du temps des mélanges parfaits. On remarque que le temps du mélange parfait amélioré coût en moyenne 2.6 fois le temps du produit correspondant. Par contre le temps du mélange parfait naif est de l'ordre de 5 fois le temps du produit correspondant.

6.8.1.3 Performances de cette approche de parallélisation

Le tableau suivant résume les performances de l'algorithme de la première approche qui se base sur la méthode mixte du paragraphe 6.4.4 (pour un seul terme).

Deux affectations sont expérimentés, correspondant à deux niveaux de coupe de l'arbre des mélanges parfaits successifs (le niveau $k_0 = 2$, $k_0 = 3$). Dans les deux cas, on utilise les deux versions de l'algorithme du mélange parfait. Le nombre de processeur est $p = 121$.

T	version naive $k_0 = 2$ / $k_0 = 3$	version amél $k_0 = 2$ / $k_0 = 3$	Accél (naive) $k_0 = 2$ / $k_0 = 3$	Accél (amél) $k_0 = 2$ / $k_0 = 3$
302500	5.349544	4.042599	33.39	44.19
	6.286226	4.643199	28.42	38.47
422500	7.847352	5.544720	33.97	48.08
	8.183943	5.587176	32.57	47.71
490000	8.342731	6.041246	38.24	52.81
	9.679091	6.581260	32.96	48.48
562500	9.077351	6.580205	41.60	57.39
	11.038445	7.533893	34.21	50.13
810000	13.464790	9.576648	44.03	61.92
	16.146424	11.105688	36.72	53.38
902500	14.307122	10.379127	47.45	65.40
	18.157438	12.420265	37.38	54.65
100000	17.354954	12.494541	44.50	61.8
	20.271712	13.960476	38.10	55.32
1102500	18.256453	13.140419	47.86	66.50
	22.425241	15.317111	38.96	57.05

Dans le tableau précédent, on confirme qu'il faut choisir k_0 le plus petit possible $k_0 = kin$. On constate un gain de la version avec mélange parfait amélioré d'environ 20%.

Dans le paragraphe 6.8.1.2, nous avons comparé le temps des mélanges parafait avec le temps des produits correspondants, nous allons exploiter les résultats de ces comparaison pour évaluer l'ordre de grandeur de l'accélération de cette approche par un modèle de calcul simpliste avec les hypothèses suivantes :

- Le temps d'un mélange parfait amélioré est 2.6 fois le temps du produit correspondant.
- Le temps d'un mélange parfait naif est 5 fois le temps du produit correspondant.
- Toutes les matrices du modèles sont de même taille.
- L'équilibrage de charge est parfait.

L'accélération est donnée par :

$$s = \frac{C\ N\ Prod}{\frac{C\ N\ Prod}{p} + C\ k_0\ Mel} \tag{6.29}$$

Le numérateur de l'expression 6.29, représente le coût séquentiel des C terme, où $Prod$ correspond au temps du produit à la phase i . Le deuxième terme du dénominateur correspond aux communications, Mel correspond au temps d'un mélange parfait.

Soit en remplaçant le temps Mel par sa valeur dans 6.29 :

$$s = \frac{C\ N\ Prod}{\frac{C\ N\ Prod}{p} + C\ k_0\ f\ \frac{Prod}{p}} \tag{6.30}$$

où f représente le rapport du temps d'un mélange parfait sur le temps du produit correspondant. On peut écrire s sous une autre forme :

$$s = \frac{p}{1 + \frac{k_0\ f}{N}} \tag{6.31}$$

Pour avoir un ordre de grandeur de l'accélération dans les mêmes conditions que dans l'expérience résumé dans le tableau précédent, on pose $p = 121$, la taille des matrice (b_i) de l'ordre de 10, $N = 6$ (1 million d'états) et enfin en pose $k_0 = 2$. On obtient :

$$s_{amel} = \frac{121}{1 + \frac{2\ 2.6}{6}} = 64.8$$

et

$$s_{naif} = \frac{121}{1 + \frac{2\ 5}{6}} = 45.35$$

Ces résultats sont proches de ceux données par le tableau précédent.

Pour mieux observer le déséquilibre du calcul et des communication qui est affecte l'accélération, nous allons choisir des modèles qui contiennent de gros automates pour augmenter le coût du calcul. En effet, le coût du calcul qui $(T\ \sum_{i=1}^{N}\ b_i)$ est de plus en plus grand quand les b_i sont grand à T constant. Pour T constant, le coût des communications reste constant, par contre le volume de calcul croit avec la taille des b_i.

T	version naive	version amélio	Accél(naive)	Accél(amélio)
20^4=160000	2.985400	2.252416	45.65	60.49
25^4=390625	7.715392	5.901440	53.34	69.74
30^4=810000	16.32025	12.7376	62.32	79.84
32^4=1048576	21.98304	16.914688	63.74	82.85
33^4=1185221	23.674752	19.04660	68.96	85.75

On remarque que l'accélération est d'autant meilleure que b_i est grand.

6.8.2 Amélioration de la méthode en utilisant le recouvrement des communications par les calculs

On rappelle que le descripteur d'un (RAS) est de la forme :

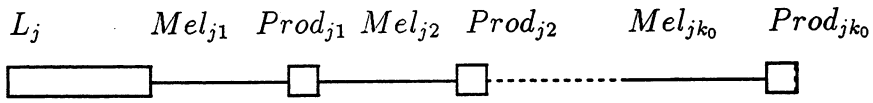
$$P = \Sigma_{t=1}^{C} \otimes_{i=1}^{N} M_{ti}.$$

La méthode de base que nous allons utiliser est la méthode mixte décrite dans le paragraphe 6.4.4. On rappel briévement que cette méthode se compose de deux parties:

- La première partie comporte $(N - k_0)$ mélanges parfaits internes et leur produit correspondant (voir le paragraphe 6.2.6). On désigne cette partie par : la phase des produits locaux (pas besoin de faire des communications pour effectuer les mélanges parfaits).
- La deuxième partie consiste en k_0 mélanges parfaits externes et leur produit correspondant (voir le paragraphe 6.2.5). On désigne cette partie par : la phase des produits externes (les mélanges parfaits nécessitent des communications entre processeurs).

La nouvelle méthode qu'on va proposer utilise le recouvrement des communications (mélanges parfaits externe) par les calculs (produits locaux). Les motivations de cette méthode de recouvrement sont :

- Le calcul des termes est indépendant (le calcul d'un terme n'est pas influencer par le terme précédent).
- La machine permet un bon entrelacement des calculs et des communication (voir le paragraphe 1.5).



□ Représente une tâche de calcul

— Représente une tâche de communication

Figure 6.21: Contraintes de précédence des tâche dans le calcul d'un terme j

L'idée de base est de recouvrire les communications pour effectuer les mélanges parfaits externes d'un terme par les calculs des produits locaux d'un autre terme.

Pour dériver un algorithme général, nous allons étudier deux cas de figures :

- Le temps des produits locaux d'un terme est de l'ordre du temps d'un mélange parfait externe d'un autre terme.
- Le temps des produits locaux d'un terme est supérieur ou égal au temps des produits externes d'un autre terme.

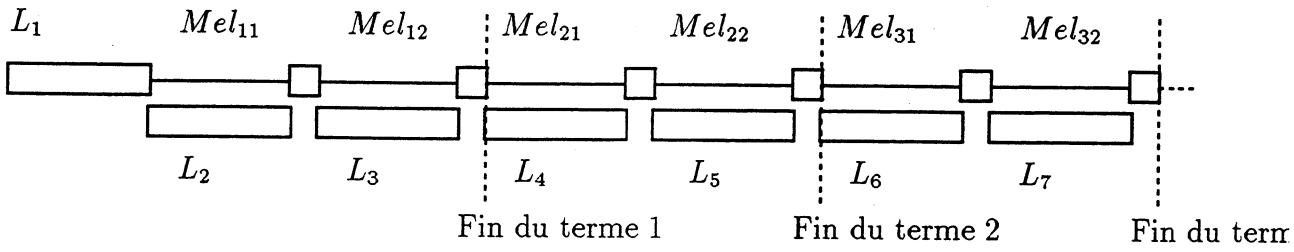
Dans ce qui suit, nous allons étudier ces deux cas de figure. Avant de commencer cette étude, nous allons donner quelques notations.

Notations

Soit un terme j, $j \in [1..C]$. On note par L_j la tâche qui effectue les produits locaux du terme j. On note par Mel_{ji} la tâche qui effectue le mélange parfait externe i du terme j et $Prod_{ji}$ la tâche qui effectue le produit correspondant. La figure 6.21 montre les contraintes de précédence à l'exécution d'un terme j.

6.8.2.1 Premier cas:le temps des produits locaux est du même ordre que le temps d'un mélange parfait externe

L'algorithme qu'on propose dans ce cas est le suivant :



☐ Représente une tâche de calcul $Prod_{ji}$

— Représente une tache de communication

Figure 6.22: Schéma de recouvrement dans le premier cas, pour $k_0 = 2$

On lance L_1, puis en parallèle, on lance (Mel_{11} et L_2). A la fin de ces deux tâches en lance $Prod_{11}$. En suite, on lance (Mel_{12} et L_3), puis $Prod_{12}$ et ainsi de suit. A la fin du terme 1, on aura traiter le terme 1 complétement et terminé les tâches L_j , $j \in [1..k_0+1]$. On fait de même pour les produits externe du deuxième terme à savoir :

En lance (Mel_{21} et L_{k_0+2}) puis en lance $Prod_{22}$ etc... (voir figure 6.22).

Remarque

Par ce procédé de recouvrement, on remarque :

La fin du terme j coincide avec la fin des produits locaux du terme $(j.k_0 + 1)$.

De cette remarque on déduit que la fin des produits locaux du terme C coincide avec la fin du terme $(\frac{C-1}{k_0})$. Si on néglige 1 devant C, il en résulte qu'après avoir terminer les $\frac{C}{k_0}$ premiers termes, il reste à finir les $(C - \frac{C}{k_0})$ derniers sans aucun recouvrement des communications.

Gain de cette méthode

Dans cette méthode de recouvrement qui correspond au premier cas de figure cité précédemment, les mélanges parfaits externes de $\frac{C}{k_0}$ premiers termes sont recouverts par des calculs (produits locaux), soit un gain de :

$$GainRecouv1 = \frac{C}{k_0} k_0 \; tr \; Mel = C \; tr \; Mel$$

avec Mel qui représente le temps d'un mélange parfait externe et tr le taux de recouvrement d'un mélange parfait par les produits locaux.

Evaluation du coût mémoire de cette solution

Nous allons essayer de donner une borne supérieure de la mémoire utilisée par cette solution. Cette borne correspond au nombre de termes maximum qu'on traitent simultanément pendant l'exécution de cette méthode. On note par mem_j le sous-vecteur qui contient le résultat du terme j et mem_x un sous-vecteur où on accumule les résultats des termes déja terminer. Pour quantifier cette borne, nous allons compter le nombre de sous-vecteurs nécessaires entre les instants de fin de deux termes :

Début du terme 1
  On a besoin des sous-vecteurs : mem_1, mem_2, ..., mem_{k_0+1}
Fin du terme 1
  On a besoin des sous-vecteurs : mem_x, mem_2, ..., mem_{k_0+1}, mem_{k_0+2}, ...mem_{2k}
Fin du terme 2
  On a besoin des sous-vecteurs : mem_x, mem_3, ..., mem_{k_0+2}, ...mem_{2k_0+1}, ...mem
Fin du terme 3
.
.
.
Fin du terme $j-1$
  On a besoin des sous-vecteurs : mem_x, mem_j, ..., $mem_{j.k_0+1}$
Fin du terme j

Cette borne maximale est donc de l'ordre de $\frac{C}{k_0}$ qui est atteinte pendant le lancement de la tâche L_C . Comme C est généralement très grand, cette solution est donc pratiquement impossible à réaliser.

Pour remédier à ce problème de mémoire, une autre stratégie consiste à prendre un groupe de terme et d'appliquer le procédé de recouvrement précédent sur ce groupe. Le

nombre de sous-vecteur nécessaire dépend de la taille de ce groupe (le nombre de terme). A la fin de ce groupe, on obtient un seul sous-vecteur résultat des termes déja traiter et on fait de même avec les groupes de termes suivants.

Groupement des termes pour le recouvrement

Dans cette stratégie, on groupe les termes et on effectue la méthode précédente sur chacun des groupes.

Soit ng le nombre de groupe. Chaque groupe contient $\frac{C}{Ng}$ termes. Calculons le gain de cette stratégie par groupe :

$$Gain_{groupe} = \frac{C}{Ng}\ tr\ Mel.$$

Soit un gain total sur tous les groupe de :

$$GainRecouvl_{gr} = \frac{C}{Ng}\ ng\ tr\ Mel\ =\ C\ tr\ Mel.$$

Le gain de cette stratégie est le même que dans la précédente, de plus, elle consomme moins de mémoire soit un maximum de sous-vecteur de l'ordre de $\frac{C}{k_0\ ng}$.

L'idée est d'augmenter ng (le nombre de groupe) pour minimiser l'espace mémoire. Pour que cette méthode fonctionne, le nombre de terme minimum est $\frac{C}{Ng}\ = 5$ et le nombre de sous-vecteurs nécessaire est de 5.

Estimation de l'accélération dans ce cas

On rappelle que le nombre de terme est de $\frac{C}{5}$, le gain en mélange parfaits externes par groupe est de $(4\ tr\ Mel)$. Le gain total sur tous les groupes est de $(\frac{C\ 4\ tr\ Mel}{5})$:

$$Accel_{recouvl} = \frac{C\ N\ Prod}{\frac{C\ N\ Prod}{p} + C\ k_0\ Mel - \frac{4\ C\ tr\ Mel}{5}}$$

On remplace Mel par sa valeur en fonction de $Prod$ (voir le paragraphe 6.8.1.2) :

$$Accel_{recouvl} = \frac{C\ N\ Prod}{\frac{C\ N\ Prod}{p} + C\ k_0\ (\frac{f\ prod}{p}) - \frac{4\ C}{5}\ tr\ (\frac{f\ prod}{p})}$$

et on obtient après simplification :

$$Accel_{recouvl} = \frac{p}{1 + \frac{f}{N}(k_0 - \frac{4\ tr}{5})}$$

Pour $p = 121$, $N = 6$, $k_0 = 2$, $f_{am} = 2.6$ et $tr_{am} = 0.67$ (mélange parfait amélioré), on estime l'accélération à :

$$Accel_{recouv1} = \frac{121}{1 + \frac{2.6}{6}(2 - \frac{4\ 0.67}{5})} = 74$$

Pour le mélange parfait naif: $p = 121$, $N = 6$, $k_0 = 2$, $f_{na} = 5$ et $tr_{na} = 0.795$, on estime l'accélération à :

$$Accel_{recouv1} = \frac{121}{1 + \frac{5}{6}(2 - \frac{4\ 0.795}{5})} = 56$$

Les valeurs des constantes tr_{am} et tr_{na} sont déduites de l'expérimentation du paragraphe 6.8.1.1. On rappelle la proportion de communication pure (qu'on peut recouvrire) est respectivement de 80%, 96% du temps des mélanges parfait respectivement amélioré et naif. On rappelle aussi que dans le paragraphe 1.5, le recouvrement des communications par le calcul n'est pas parfait, il présente une dégradation de l'ordre de 20%, d'où les valeurs des constantes tr_{am} et tr_{na}.

6.8.2.2 Deuxième cas : Le temps des produits locaux est supérieur au temps total de tous les produits externes

Dans ce cas, on propose un algorithme de recouvrement qui pipeline les produits locaux de chaque terme :

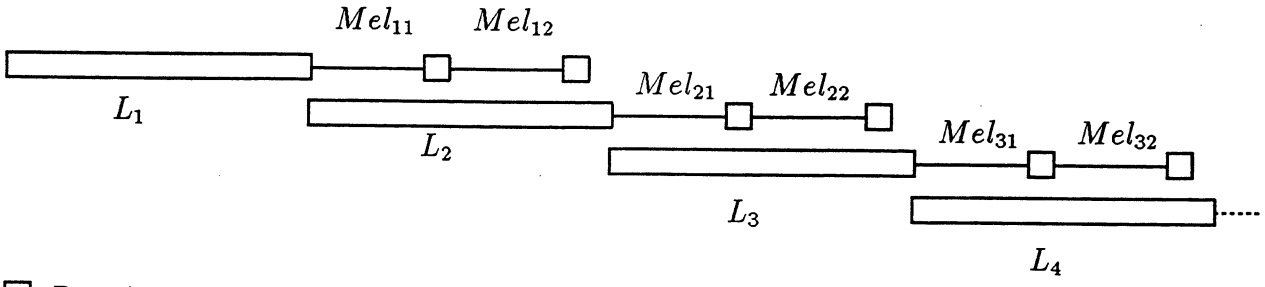
Pour initialiser le pipeline, en lance la tâche L_1, à la fin de celle-ci, en lance L_j en parallèle avec les produits externes du terme $j-1$ pour $j \in [2..C]$ (figure 6.23). A la fin du pipeline, en lance les produits externes du terme C (qui ne sont pas recouverts par aucun calcul).

Coût mémoire dans ce cas

Dans ce cas, à chaque instant de l'algorithme, on exécute 2 termes simultanément. Le coût en mémoire est donc deux sous-vecteurs plus un pour cumuler les termes déja traités.

Estimation de l'accélération dans ce cas

$$Accel_{recouv2} = \frac{C\ N\ Prod}{\frac{C\ N\ Prod}{p} + C\ k_0\ Mel - (C-1)\ k_0\ Mel\ tr}$$



☐ Représente une tâche de calcul $Prod_{ji}$

— Représente une tâche de communication

Figure 6.23: Schéma de recouvrement dans le deuxième cas, pour $k_0 = 2$

on néglige 1 devant C :

$$Accel_{recouv2} = \frac{C\ N\ Prod}{\frac{C\ N\ Prod}{p} + C\ k_0\ \frac{f\ prod\ (1-tr)}{p}}$$

et on obtient après simplification :

$$Accel_{recouv2} = \frac{p}{1 + \frac{k_0\ f\ (1-tr)}{N}}$$

Pour $p = 121,\ N = 6,\ k_0 = 2,\ f_{am} = 2.6$ et $tr_{am} = 0.67$ (mélange parfait amélioré), on estime l'accélération à :

$$Accel_{recouv2} = \frac{121}{1 + \frac{2\ (2.6)\ (1-0.67)}{6}} = 94$$

Pour le mélange parfait naif: $p = 121,\ N = 6,\ k_0 = 2,\ f_{na} = 5$ et $tr_{na} = 0.795$ (mélange parfait naif), on estime l'accélération à :

$$Accel_{recouv2} = \frac{121}{1 + \frac{2\ (5)\ (1-0.795)}{6}} = 90$$

6.8.2.3 Algorithme général utilisant la technique de recouvrement

L'algorithme que nous proposons dans ce paragraphe prend en compte les deux cas précédents, pour surpasser le problème des tailles des produits locaux et des produits locaux, on rajoute des points de synchronisations. Cet algorithme est décrit par ce qui suit :

On initialise l'algorithme par le lancement de la tâche L_1. Puis en lance en parallèle L_j et les produits externes du terme $j-1$, on attend que ces deux tâches terminent (point de synchronisation), puis en refait ce procédé pour $j \in [2..C]$. A la fin de l'algorithme, en lance les produits externes du terme C.

Coût mémoire

A chaque instant de l'algorithme, on exécute 2 termes simultanément. Le nombre de sous-vecteurs nécessaires est 3.

Estimation de l'accélération

Il est clair que l'accélération de cette méthode à pour borne inférieure l'accélération du premier cas et pour borne supérieure l'accélération du deuxième cas.

6.9 Conclusion

Dans ce chapitre, nous avons étudié la parallélisation du produit vecteur par le descripteur d'un (RAS), l'approche de parallélisation des termes (première approche) donne des résultats satisfaisant sur les deux points de vues : taille des problème à traiter (de l'ordre du plusieurs **Millions** d'états) et son efficacité (accélération) (en utilisant des techniques de recouvrement des communications par les calculs).

L'étude de la parallélisation des méthodes de résolution itératives qui utilisent des techniques de projections tel que (Arnoldi, GMRES), intégrant cette méthode de produit vecteur descripteur, constitue un des axes de nos futures recherches.

Conclusion

Dans cette thèse, nous avons mis le parallélisme au service de l'évaluation des performances. Nous avons vu que les machines du type (MIMD) à mémoire distribuée s'apprêtent bien à la résolution numérique des modèles Markoviens, de par leur capacité mémoire et leur parallélisme massif, qui offre une puissance de calcul considérable.

Nous avons vu aussi que le coût des mouvements de données dans les algorithmes parallèles sur ces machines constitue un facteur de dégradation des performances important qu'il faut réduire. Nous avons proposé des procédures de communications efficaces qui résolvent un type particulier de mouvement de données qu'on retrouve fréquemment dans l'algorithmique numérique matricielle. Ces procédures peuvent être automatisées pour leur utilisation transparente par le programmeur (sous forme de bibliothèque).

Dans le cadre de la résolution numérique des modèles Markoviens, nous avons présenté des méthodes itératives qui s'adaptent bien au calcul parallèle, dans le sens où elles présentent de faibles contraintes de synchronisation. Nous les avons expérimentées et certaines de ces méthodes donnent de très bons résultats.

Dans le cas particulier de la modélisation par réseaux d'automates stochastiques, nous avons présenté une méthode de parallélisation du produit de base (vecteur-descripteur) des méthodes itératives de résolution. Cette méthode permet de traiter de très gros modèles de l'ordre de plusieurs **Million d'états** en un temps satisfaisant. Pour le choix de la méthode itérative à utiliser, des comparaisons entre différentes méthode (dans le cas séquentiel), montrent que les méthodes de projection (Arnoldi, GMRS) donnent de bons résultats. La parallélisation de ces méthodes nous semble comme une suite logique de cette thèse.

Aussi, dans le cadre de recherche ultérieure, il est intéressant d'étudier la parallélisation des méthodes d'agrégation désagrégation qui donnent de très bons résultats dans les cas des modèles (NCD).

Un autre axe de recherche qui nous tient à coeur est l'écriture de nouveaux algorithme de routage se basant sur le modèle "wormhole", à l'heure où plusieurs machines parallèles sont munis de support sophistiqués pour le développement des routeurs.

Annexe A

Modèle de calcul asynchrone

Dans ce paragraphe nous allons passer en revue les conditions et théorèmes d'applications des méthodes *chaotiques* ou itérations asynchrones et quelques exemples de modèles relatifs aux chaines de Markov.

Le modèle suivant est développé dans [15], et on retrouve les mêmes fondements théoriques dans [11] [31] [3] [38] [39] [8] :

Soit $X_1, X_2, X_3, ..., X_T$ des ensembles et $X=X_1 \times X_2 \times X_3 ... \times X_T$ leur produit cartésien. Les éléments de X sont des T-uples tels que pour $x \in X$ on écrit $x=(x_1, x_2, ..., x_T)$, avec $x_i \in X_i$ pour $i=1,T$.
Soit $f_i : X \rightarrow X_i$ une fonction donnée, et soit $f : X \rightarrow X$ la fonction définie par :

$$f(x) = (f_1(x),\ f_2(x), ...,\ f_T(x)), \forall\, x\ \in\ X. \qquad (A.1)$$

Le problème est de trouver un point fixe de f, $\bar{x} \in X$ avec

$$\bar{x}=f(\bar{x}) \Leftrightarrow \bar{x}_i=f_i(\bar{x})\ \forall i=1, T.$$

Soit $x_i(t)$ la valeur de la i^{eme} composante au temps t. On suppose que l'ensemble $T_e=\{0, 1, 2, ...\}$ représente les dates auxquelles une ou plusieurs composantes x_i de x sont mises à jour par un processeur durant un calcul distribué. Soit T_e^i l'ensemble des dates auxquelles la composante x_i est mise à jour. On suppose que le processeur qui met à jour la composante x_i peut ne pas avoir accès aux plus récentes valeurs des autres composantes de x.

On écrit l'équation A.1 comme suit :

$$x_i(t+1) = \begin{cases} f_i(x_1(\tau_1^i(t)), ..., x_T(\tau_T^i(t))), & \text{si } \forall t \in T_e^i; \\ x_i(t), & \text{si } \forall t \notin T_e^i. \end{cases} \quad (A.2)$$

$\tau_j^i(t)$ représente la date à laquelle la composante x_j est disponible sur le processeur qui met à jour la valeur de x_i.

Remarque :

$$0 \leq \tau_j^i(t) \leq t \quad \forall\, t \in T_e.$$

La différence (t-$\tau_j^i(t)$) peut être vue comme le délai de communication correspondant à l'envoi du processeur qui met à jour la composante x_j vers le processeur qui modifie la composante x_i.

Exemple :

On considère un réseau de T processeurs, chaque processeur dispose d'une mémoire locale. Le processeur numéro i stocke le vecteur $x^i(t)$=($x_1^i(t)$, ..., $x_T^i(t)$) et met à jour sa i^{eme} composante $x_i^i(t)$ aux dates $t \in T_e^i$ selon le procédé itératif :

$$x_i^i(t+1)=f_i(x^i(t)).$$

Occasionnellement, le processeur i envoie aux autres processeurs la nouvelle valeur de x_i^i. Qand le processeur j reçoie la nouvelle valeur x_i^i (après un certain délai), il stocke cette valeur dans son vecteur x^j à l'indice i. Juste après cette opération on aura :

$$x_i^j(t+1)=x_i^i(\ \tau_i^j(t)).$$

Les composantes du vecteur $x(t+1)$ sont :

$$x(t+1)=(x_1^1(t+1), ..., x_T^n(t+1)).$$

A.1 Asynchronisme total

Hypothèse 1 :

On suppose que chaque ensemble T_e^i est infini et que si $\{t_k\}$ est une séquence infinie d'éléments de T_e^i, alors $\lim_{k\to\infty} \tau_j^i(t_k)=\infty$ pour tout j.

Cette hypothèse garantit que chaque composante est mise à jour infiniment souvent. D'une autre manière, la quantité $(t\text{-}\tau_j^i(t))$ peut être non bornée quand t croit.

Hypothèse 2

Soit une suite d'ensembles non vides $\{X(k)\}$ tel que :

$$... \subset X(k+1) \subset X(k) ... \subset X .$$

Vérifiant les conditions suivantes :

1. Condition de convergence synchrone

   $$f(x) \in X(k+1), \forall k \text{ et } x \in X(k).$$

   De plus, si une suite $\{y^k\}$ telle que $y^k \in X(k)$ pour tout k, alors la limite de $\{y^k\}$ est un point fixe de f.

2. Condition de fermeture

   Pour tout k, il existe des ensembles $X_i(k) \subset X_i$ tel que :

   $$X(k)=X_1(k) \times X_2(k) \times ... X_T(k).$$

Il faut noter que la condition de convergence synchrone implique, que les points limites des suites générées par les itérations synchrones sont des points fixe de f, en supposant que le vecteur initial x appartienne à $X(0)$. Remarquer aussi que la condition de fermeture implique que si $x \in X(k)$ et $x' \in X(k)$, et si on remplace la i^{eme} composante de x par la i^{eme} de x', on obtient un élément de $X(k)$. Typiquement la condition de fermeture s'établit quand $X(k)$ est une sphère dans $\Re^T$ par rapport à la norme $\|x\|_\infty^w=\max_i \frac{|x_i|}{w_i}$, avec $w \in \Re^T$.

Théorème 6 *Sous les hypothèses 1 et 2 et si une solution initiale estimée :*

$$x(0) = (x_1(0), x_2(0), ..., x_T(0))$$

appartient à l'ensemble X(0), alors toute limite de $\{x(t)\}$ est un point fixe de f.

Ce théorème est un outil théorique très puissant, mais en pratique, une bonne identification des suites d'ensembles $\{X(k)\}$ requiert une analyse assez profonde. Dans ce qui suit nous allons voir l'application du théorème de convergence dans le cas des systèmes linéaires et plus particulièrement dans le cas de systèmes linéaires dont la matrice est stochastique.

A.2 Application au système d'équations linéaires

On considère le cas :

$$f(x)=Ax+b,$$

avec A une matrice carrée d'ordre T dont les éléments sont notés a_{ij}, b un vecteur de $\Re^T$. On veut trouver un vecteur $\bar{x}$ tel que :

$$\bar{x}=A\bar{x}+b,$$

Le modèle de calcul de l'itération asynchrone est le suivant :

$$x_i(t+1) = \begin{cases} \sum_{j=1}^{n} a_{ij}\ x_j(\tau_j^i(t)) + b_i & \text{si } t \in T_e^i \\ x_i(t) & \text{si } t \notin T_e^i \end{cases}$$

Proposition 1 [15] :

Soit A une matrice carrée d'ordre T telle que, la matrice (I-A) est inversible. Alors les assertions suivante sont équivalentes :
(i) Le rayon spectral $\rho(|A|) < 1$.
(ii) Pour tout initialisation $x(0)$ et tout vecteur b de $\Re^T$, tout choix de l'ensemble T_e^i tel que T_e^i est infini, et pour tout choix des variables $\tau_j^i(t)$ qui satisfasse :

$$t-2 \ \leq \ \tau_j^i\ (t) \ \leq \ t \quad \forall\, t$$

La suite $\{x(t)\}$ produite par l'itération asynchrone converge vers le vecteur $(I-A)^{-1}b$.

Dans cette proposition on retrouve la condition de fermeture comme conséquence de l'assertion (i) (le rayon spectral de A et strictement inférieur à l'unité).

Exemple de recherche du vecteur de probabilités stationnaires d'une matrice stochastique :

Rappellons que le vecteur de probabilités π d'un système Markovien de matrice de transition P est donné par la solution à : $\pi P{=}\pi$. Etant donné que P est stochastique, le rayon spectral de P est égal à 1. Pour se ramener à l'hypothèse d'application de la proposition 1, nous allons utiliser l'itération suivante :

$$\pi(t+1) = \pi(t)\tilde{P} + \pi_1(0)a$$
$$\pi_1(t+1) = \pi_1(t)$$

lorsque la matrice P s'écrit :

$$P = \begin{pmatrix} p_{11} & a \\ v & \tilde{P} \end{pmatrix}$$

avec a un vecteur ligne d'ordre $(T-1)$, v un vecteur colonne d'ordre $T-1$.

Considérons la version asynchrone suivante :

$$\pi_i(t+1) = \begin{cases} \sum_{j=1}^{T} \tilde{P}_{ij}\ \pi_j(\tau_j^i(t)) + \pi_1(0)\ a & \text{si } i > 1,\ t \in T_e^i \\ \pi_i(t) & \text{si } i > 1,\ t \notin T_e^i \\ \pi_i(0) & \text{si } i = 1 \end{cases}$$

Il clair que le rayon spectral de $\tilde{P}$ est strictement inférieur à un. La condition nécessaire de convergence est établie. Si de plus, on choisit le retard τ tel que :
$t-2 \leq \tau_j^i(t) \leq t\ \forall t$, l'itération asynchrone précédente converge vers le vecteur de probabilités à une constante de normalisation près.

Dans ce schéma de convergence, on ne permet que des retards de 2 entre les itérations. Dans ce qui suit, nous allons voir un modèle asynchrone appelé asynchronisme partiel, où on borne le retard par une valeur B [15] [3].

A.3 Modèle de calcul asynchrone partiel

On définit de la même manière que dans [3] le modèle asynchrone total l'itération de base. En plus, on pose trois hypothèses principales :

(a) $\forall i$, $\forall t \geq 0$ au moins un élément de $\{t, t+1, ..., t+B-1\}$ appartient à T_e^i.
(b) $t - B \leq \tau_j^i(t) \leq t$ pour tout i, j et tout $t \geq 0$ appartenant à T_e^i.
(c) $\tau_i^i(t)=t$ pour tout i et $t \in T_e^i$.

Autrement dit :

(a) Chaque processeur met à jour les composantes dont il est chargé au moins une fois dans un intervalle de temps de durée B.
(b) L'information utilisée par chaque processeur date d'au plus B unité de temps.
(c) Une composante du vecteur n'est mise à jour que par un seul processeur.

Nous allons considérer l'itération A.2 en vérifiant les trois hypothèses citées ci-dessus. L'information est modifiée par l'algorithme après au plus B unités de temps.

On définit, l'états de l'algorithme à un instant donné t par le vecteur $z(t) \in X^B$ défini par :

$$z(\mathrm{t})=(x(\mathrm{t}), x(t-1), ..., x(t-B+1))$$

On suppose que l'application f qu'utilise l'itération est continue est admet un ensemble non vide $\bar{X} \in X$ de points fixes. Soit $\bar{Z}$ l'ensemble de tous les vecteurs $\bar{z} \in X^B$ de la forme $\bar{z}=(\bar{x}, \bar{x}, ..., \bar{x})$, avec $\bar{x} \in \bar{X}$.

Proposition 2 (Bertsekas et Tsitsiklis) :

On suppose qu'il existe un entier positif $\bar{t}$ et une fonction $d : X^B \rightarrow [0, \infty]$ avec les propriétés suivantes :
Pour chaque initialisation $z(0) \notin \bar{Z}$ de l'itération et tout suite d'événements ultérieurs (conforme au hypothèses (a) , (b), (c)) on a $d(z(\bar{t})) < d(z(0))$ et $d(z(1)) \leq d(z(0))$. Donc tout point limite de la suite $\{z(t)\}$ générer par l'algorithme asynchrone partiel, appartient à l'ensemble $\bar{Z}$.

De plus, dans le cas où $X=\Re^T$, on choisi la fonction d de la forme $d(z)=inf_{\bar{z} \in \bar{Z}} \|z - \bar{z}\|$, avec $\|.\|$ une norme de vecteur. Si f est de la forme $f(x)=Ax+b$, alors, $d(z(t))$ converge géometriquement vers 0.

Dans le paragraphe suivant, nous allons voir un schéma de calcul du type asynchrone partiel, appliqué dans le cas de système linéaire dont la matrice est stochastique, en vérifiant quelques conditions très simples à mettre en oeuvre pratiquement.

A.4 Application au calcul du vecteur stationnaire de probabilités

Hypothèses

1. Il existe au moins une coordonnée $\bar{i}$, $1 \leq \bar{i} \leq n$ telle que

$$\begin{array}{c} P_{\bar{i}\bar{i}} > 0 \\ \tau_{\bar{i}}^{i}(t)=t \\ x_{\bar{i}}(0) > 0 \end{array}$$

2. Il existe un entier B tel que :

$$0 \leq \tau_j^i(t) \leq B < \infty \text{ pour } t \geq 0 \text{ et } 1 \leq i,j \leq n$$

3. Chaque composante du vecteur est mise à jour infiniment souvent.

Dans [3], on montre que sous les hypothèses (1), (2) et (3) le schéma asynchrone partiel converge avec une vitesse geometrique vers le vecteur de probabilités stationnaires. La preuve est similaire que pour la proposition 2, en choisissons la fonction $d(z(t)) = \max_i \max_{t-B < \tau \leq t} x_i(\tau) - \min_i \min_{t-B < \tau \leq t} x_i(\tau)$.

L'hypothèse (1) semble assez lourde. En fait, même si tous les éléments diagonaux de P sont nuls, il suffit d'une simple transformation pour que l'hypothèse (1) soit satisfaite :

$$\bar{P} = \alpha P + (1-\alpha) I \text{ avec } 0 < \alpha \leq 1$$

Nous avons $\pi P = \pi \Leftrightarrow \pi \bar{P} = \pi$, et tous les éléments diagonaux de $\bar{P}$ sont non nuls. La condition $\tau_i^i(t)=t$ est satisfaite pour tout t si la composante i est mise à jour par un seul processeur.

Annexe B

Terminaison des algorithmes itératifs parallèles

B.1 Convergence globale

Dans le chapitre 5, nous avons présenté différentes schémas itératifs pour résoudre le problème :

$$\pi^{(t+1)} = \pi^{(t)} P.$$

On rappelle qu'une itération (t) consiste à effectuer un produit parallèle πP selon la méthode (C) (voir le paragraphe 2.4.1), suivit d'une procédure de communication du type échange total (voir le paragraphe 2.4), pour pouvoir entamer l'itération $(t+1)$.

Dans les algorithmes itératifs, chaque processeur itère sur les composantes dont il est résponsable. Le problème est de savoir quand est-ce que les algorithmes terminent , ie. les sous-vecteurs de chaque processeur ont tous atteint une certaine précision, on parle alors de convergence globale.

Dans un algorithme distribué il est difficile de déterminer qu'une condition globale de terminaison est satisfaite, car chaque processeur ne connait qu'une information partielle sur le déroulement de l'algorithme. Généralement le problème de détection de la terminaison se décompose en deux phases [27] [14] [15] :

1. Détection de la terminaison locale.

2. Une procédure spéciale est utilisée pour détecter la terminaison en un temps fini.

Nous allons proposer un algorithme de détection de la terminaison dont le coût est négligeable car il est intégré avec l'algorithme de calcul. Pour ce faire, on utilise une structure de donnée tableau qu'on met à jour pendant le déroulement de l'algorithme. Le principe de l'algorithme de terminaison est le suivant :

Soit *term* un tableau booléen de dimension p sur chaque processeur. Pour $l = 0, p-1$, $term[l] = vrai$ si le processeur de numéro l a détecté la convergence locale par un test de précision, $faux$ sinon. Pendant le déroulement de l'algorithme de calcul et plus précisément lors de l'exécution du processus $AllToAll()$, on rajoute à l'envoi du sous-vecteur, le booléen $term[MonNumero]$. Inversement, chaque fois qu'un processeur reçoie un nouveau sous-vecteur de données et le booléen, il met à jour son tableau de booléens : $term[NumeroRecu]$ prend la valeur du booléen reçu. Dès qu'un processeur donné voie tous son tableau *term* à vrai, alors il détecte la convergence globale.

Algorithme *TestConvergence.*

```
Début
Booleen bool;
Entier l;
(* Le tableau term est initialement à faux *)

bool = vrai;
(* Test de convergence Globale *)
   Pour l = 0 a p − 1
     bool = bool and term[l];
(* Test de convergence Local *)
   Si ( non bool)
   Alors
         CalculPrecisionLocal();
         Si Precision < Epsilon
         Alors term[MonNumero] = vrai;
         Sinon term[MonNumero] = faux;
   Sinon ConvergengeGlobal;
Fin
```

L'autre phase de mise à jour du tableau *term* est réalisé dans la procédure $AllToAll()$. On rappelle que la procédure de communication $AllToAll()$ est constituée de $\frac{p-1}{4}$ étapes de communication. Une étape de communication est décrite par ce qui suit :

- Envoyer-et-Recevoir sur les 4 liens des données.
- Recopier les 4 données reçues dans le vecteur local aux positions correspondantes.
- Mettre à jour les 4 booléens reçues dans le tableau *term* aux positions correspondantes.

Remarque :

Une remarque importante à été faite dans [16] concernant la détection de terminaison locale d'un algorithme asynchrone : si un processeur donné s'arrête de calculer à une précision locale donnée, la convergence de l'algorithme asynchrone peut être mise en cause. En effet, si un groupe de processeurs s'arrête de calculer pendant que des processeurs d'un autre groupe n'ont pas atteint leur convergence locale, ces derniers utilisent dans chaque itération les anciennes valeurs des composantes délivrées par l'autre groupe. On remarque que le retard d'itération devient alors croissant non borné, donc une violation de l'hypothèse (2) du schéma asynchrone partiel (voir dans l'annexe A).
Pour pallier à ce problème, quand une convergence locale est détectée, le processeur, le signale aux autres processeurs et continue à améliorer la solution tant que la convergence globale n'est pas détecter.

Dans ce qui suit, nous allons voir le test de convergence locale que nous avons utiliser dans l'expérimentation des différents schémas itératifs.

B.2 Convergence locale

Dans le cas d'un calculateur séquentiel, les vecteurs approximés à l'itération (t) et $(t-1)$ par une méthode itératif sont en entier dans la mémoire du calculateur. Pour savoir que la convergence est atteinte, on fait le test de précision suivant :

$$\sum_{i=1}^{T} \frac{\pi_i^{(t)} - \pi_i^{(t-1)}}{\pi_i^{(t)}} < \epsilon \qquad \text{(B.1)}$$

Dans le cas d'une machine à mémoire distribuée à p processeur, chaque processeur p_l contient les sous-vecteurs $\pi_{Jl}^{(t)}$ et $\pi_{Jl}^{(t-1)}$, avec Jl l'ensemble des indices des composantes du vecteur de probabilités affectées au processeur p_l. Le test de convergence locale est

$$\sum_{i \in Jl} \frac{\pi_i^{(t)} - \pi_i^{(t-1)}}{\pi_i^{(t)}} < \frac{\epsilon}{p}.$$

La convergence globale est atteinte si toutes les convergences locales sont atteintes car

$$\sum_{i=1}^{T} \frac{\pi_i^{(t)} - \pi_i^{(t-1)}}{\pi_i^{(t)}} \leq \sum_{l=0}^{p-1} \sum_{il \in Jl} \frac{\pi_{il}^{(t)} - \pi_{il}^{(t-1)}}{\pi_{il}^{(t)}} < (\frac{\epsilon}{p})p$$

Il faut remarquer que le temps du test de convergence dans le cas parallèle est p fois plus petit que dans le cas séquentiel, mais le test parallèle (local) est plus fort.

Dans certain cas de convergence lente, si la différence entre les vecteurs itérés successives est plus petite que le seuil spécifie (ϵ), le vecteur peut être loin de la solution. Pour éviter se genre de situation, il suffit de comparer entre les vecteurs approximés à l'itération $(t-m)$ et (t), $m > 1$. Idéalement m doit être une fonction de la vitesse de convergence [48].

Annexe C

Expérimentation des méthodes de puissances classique et réactualisée en séquentiel

C.1 Plan d'expérience

Dans ce paragraphe nous allons expérimenter deux méthodes itératives, la méthode des puissances et la méthode des puissances réactualisé sur 3 types de matrices. En premier lieu, on s'intéresse aux matrices aléatoires avec un pourcentage d'éléments non nuls variable (matrice creuses) . Le deuxième type de matrices sont les matrices associés aux processus dits processus presque décomposable (NCD). Le dernier type de matrices sont les matrices tridiagonales par bloc avec des tailles de bloc variables.

Les matrices qu'on va générer (voir annexe G paragraphe D) sont au moins tridiagonales pour assurer l'irréductibilité de la chaine de Markov associée, de plus tous les états de la chaine de Markov contiennent une boucle pour assurer l'apériodicité. La précision du test de convergence est $\epsilon = 0.000001$ (voir annexe F paragraphe B.2). Les expériences suivantes représentent le nombre d'itérations moyen et le gain moyen d'un échantillon aléatoire de taille 5. Le gain est calculer comme la différence entre le nombre d'itérations de la méthode classique et la méthode réactualisée divisé par le nombre d'itérations de la méthode classique et multiplier par 100 pour avoir un pourcentage. Les expériences ont été faites sur une station SUN 360.

C.2 Matrices creuses

L'expérience consiste à observer le nombre d'itération moyen et le gain moyen. On varier le pourcentage d'éléments non nuls pour diverses tailles de la matrice (voir les figure C.1, C.2).

Dans la figure C.2, on remarque un gain de d'au moins 20 % de la méthode réactualisée par rapport à la méthode classique.

C.3 Matrices du type presque décomposable (NCD)

Dans cette expérience on génère des matrices bloc diagonale, tous les éléments qui n'appartiennent pas aux blocs diagonaux sont inférieurs à 0.0001. On fait varier la taille de la matrice et la taille des blocs en pourcentage de la taille de la matrice.

La figure C.3 donne la vitesse de convergence des deux méthodes, la figure C.4 donne le gain de la méthode réactualisée par rapport à la la méthode classique qui est de l'ordre de 30 % au moins.

C.4 Matrices triagonales par bloc

Dans cette expérience on génère des matrices tridiagonales par bloc et on fait varier de la même manière que dans le paragraphe précédent la taille du bloc. la figure C.5 montre la vitesse de convergence des deux méthodes. La figure C.6 montre le gain de la méthode réactualisée par rapport à la méthode classique qui est d'au moins 35 %.

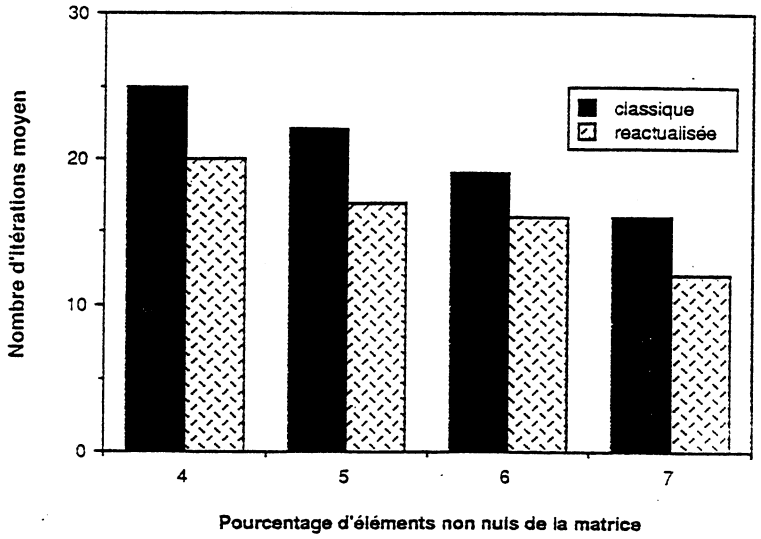


Figure C.1: Nombre d'itérations moyen pour des matrices de taille 200 (creuse)

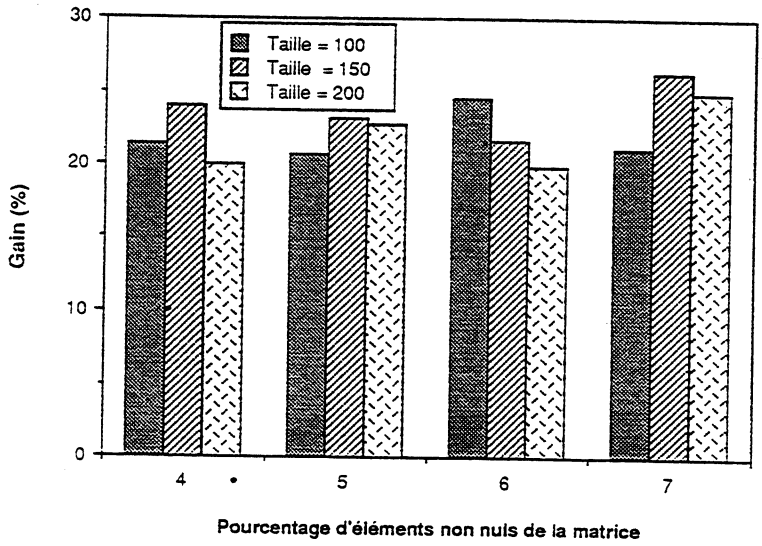


Figure C.2: Gain de la méthode avec réactualisation par rapport à la méthode classique (creuse)

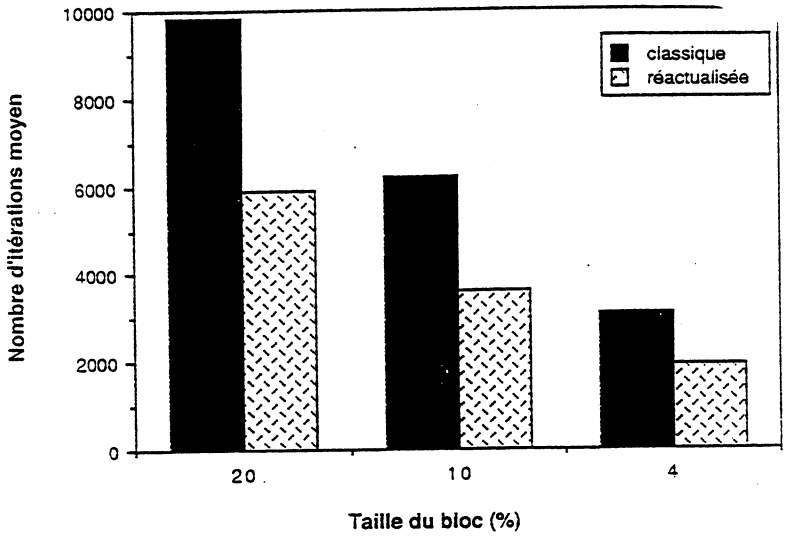


Figure C.3: Nombre d'itérations pour des matrices de taille 200 du type NCD

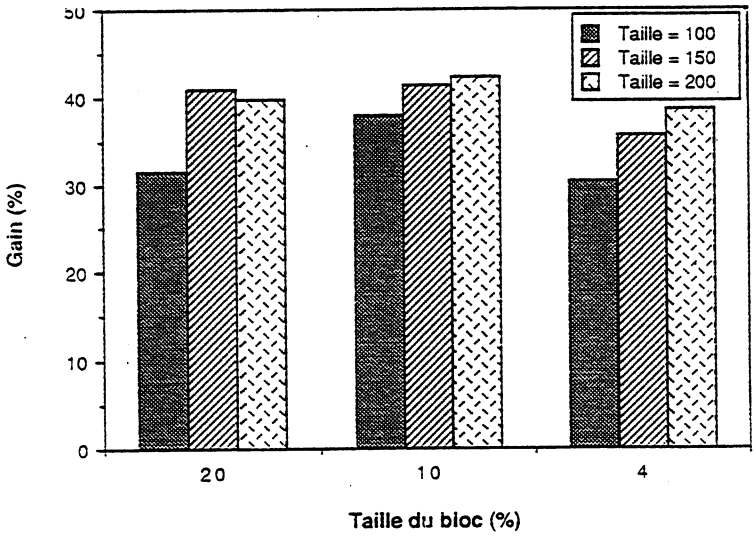


Figure C.4: Gain de la méthode réactualisée (NCD)

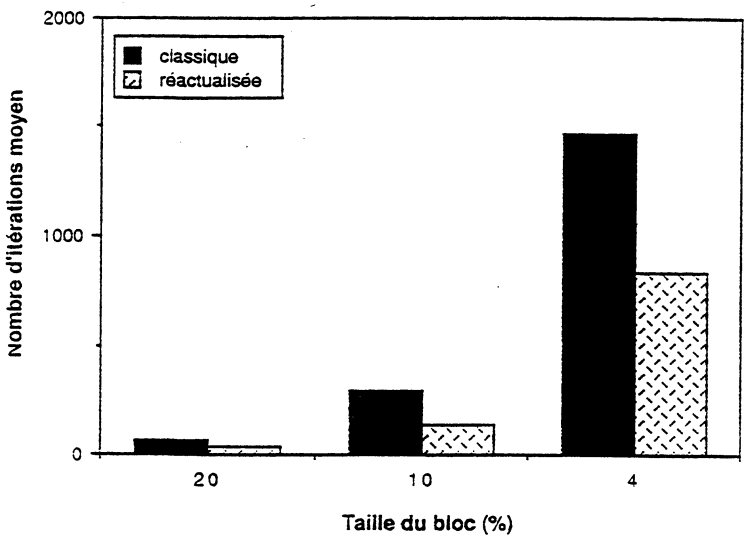


Figure C.5: Nombre d'itérations pour des matrices de taille 200 (tridiagonale blocs)

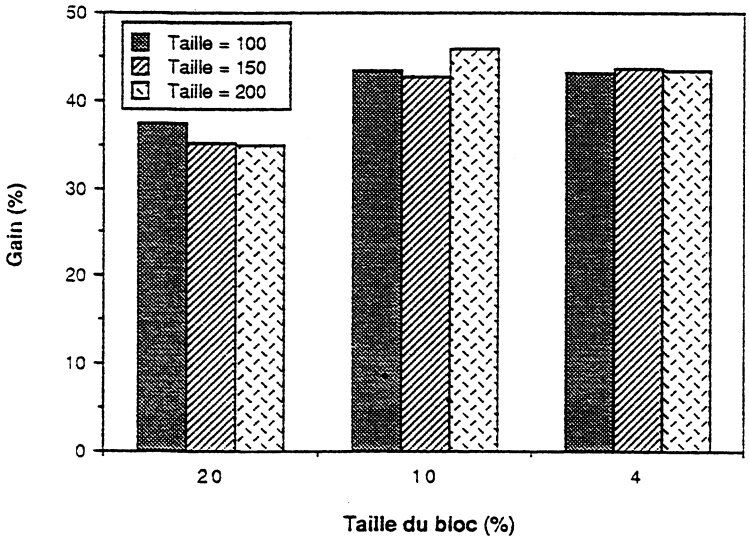


Figure C.6: Gain de la méthode réactualisée (tridiagonale blocs)

Annexe D

Génération des matrices

D.1 Matrices creuses

Pour générer une matrice de taille T creuse à $s\%$ d'éléments non nuls, on calcule le nombre d'éléments non nuls dans une colonne de la matrice qui est $NonNul=\frac{s\ T}{100}$ puis on génère aléatoirement les indices lignes (nombres aléatoires compris entre 1 et T) et les éléments non nuls correspondants (nombres aléatoires compris entre 0 et 1, les bornes sont exclues). Ainsi chaque processeur génère les colonnes de la matrice qui lui sont affectées indépendamment des autres processeurs.

D.2 Matrices du type presque décomposable (NCD)

On génère des matrices bloc diagonale qui ont pour éléments aléatoires compris strictement entre 0 et 1. Tous les éléments qui n'appartiennent pas aux blocs diagonaux sont inférieurs à ϵ=0.00001.

D.3 Matrices du type tridiagonales blocs

On génère des matrices bloc tridiagonale qui ont pour éléments aléatoires compris strictement entre 0 et 1. Tous les éléments qui n'appartiennent pas aux blocs diagonaux sont nuls.

D.4 Matrice de transition de la marche aléatoire sur une grille 2D

Soit une grille de taille $(\frac{T}{p} \times p)$, chaque noeud de la grille représente un état de la chaine de Markov (probabilité du promeneur d'être sur un noeud donné de la grille). On fixe les probabilités de transitions d'un état donné comme suit :

Pour les 4 noeuds situés aux coins de la grille, la probabilité de rester sur le même noeud est 0.2. La probabilité d'aller à un noeud voisin est 0.4. Pour les noeuds situés sur la frontière de la grille (coins exclus), la probabilité de rester sur le même noeud est 0.1. La probabilité d'aller à un état voisin est 0.3. Pour les autres noeuds, la probabilité de rester sur le même état est 0.2, la probabilité d'aller à un état voisin est 0.2.

Annexe E

Algorithmes de placement

E.1 Algorithme de Bokhari "échange de pairs"

Algorithme de Bokhari "échange de paires"

L'algorithme par échange de paires "pairwise exchange" est décrit dans [4], le principe de cet algorithme est le suivant :

Etant donné un placement initial, l'algorithme consiste à trouver la meilleure paire de processeurs, telle que l'échange de leurs sous-vecteurs fait décroitre la fonction de coût. On adopte alors ce nouveau placement pour itérer ce processus tant que qu'on peut améliorer la fonction de coût. pour éviter les minimums locaux, l'algorithme génère un placement aléatoire en échangant aléatoirement $\sqrt{p}$ paires.

Algorithme

Début
  $ameliore = vrai$
{ * placement initial *}
  Pour $k = 1$ à p
      $mapi[k] = k$
  FinPour
{* Calcul du coût du placement initial *}
  $anccout = calculer$ ($cout\ initial$)
  Tant que $ameliore$
        $coutopt = anccout$

```
        ameliore = faux
{* Début de la recherche du meilleur permuttant *}
        Pour i = 1 à p
            Pour j = 1 à p
                nouvcout = calculer le coût si on interchange les places de sπ_i^σ et s
                Si nouvcout =< anccout
                Alors meilleuri = i
                      meilleurj = j
                      anccout = nouvcout
                Fsi
            FinPour
        FinPour
{* Fin de recherche du meilleur permuttant *}
        Si nouvcout < coutopt
        Alors
              interchanger map[meilleur] avec map[meilleurj]
              ameliore = vrai
        Sinon
              Faire √p permutations aléatoire sur les p sous-vecteurs
              nouvcout = calculer le coût du placement obtenu
              Si nouvccout < anccout
              Alors on itère avec le placement obtenu
              Fsi
        Fsi
 FinTantque amélioration
Fin
```

E.2 Algorithme de récuit simulé

Le principe de cet algorithme repose sur une analogie entre certains effets thermodynamiques et la résolution de certains problèmes d'optimisation [41].

Le recuit est un abaissement contrôlé de température d'un ensemble de particules en interaction. Il permet d'obtenir une configuration des particules à énergie minimale lorsque la température est devenue suffisamment basse. La température imposée contrôle les états du système (placements donnés) accessibles à partir de l'état courant (placement courant).

L'énergie d'une configuration du système représente la fonction de coût d'un placement, la température est simulée par une variable de contrôle que l'on fait décroitre lentement à chaque itération. Pour cela, à chaque itération la température est multipliée par un coefficient constant $a \in [\,0..1\,[$. L'algorithme s'arrête lorsque la température est proche de 0. Le changement d'état du système s'exprime par une perturbation de l'état courant en permuttant aléatoirement les sous-vecteurs de deux processeurs. Si l'énergie de l'état résultant est inférieure à celle de l'état courant alors on itère ce processus avec l'état obtenu. Si l'énergie de l'état résultant est supérieure à celle de l'état courant, on accepte sous probabilité l'état obtenu. La probabilité d'acceptation est directement liée à la température courante du système. Plus la température est élever et plus on a des chances d'autoriser des états de plus grande énergie. Cette technique de remonté permet de sortir des extrêmums locaux.

Algorithme

Début
{* placement initial *}
  Pour $k = 1$ à p
      $mapi[k] = k$
  FinPour
{* Calcul du coût du placement initial *}
  $anccout$ = *calculer* (*cout initial*)
  $temp = 1000$ /* Température initiale */
  Tant que $temp \geq 0.0001$
        $iteration$ =1
        Tant que $iter < p$ {* itérations à température constante *}
            Choisir aléatoirement deux permuttant parmi p
            $nouvcout$ = calculer le coût si on interchange les deux permuttants
            $deltacout = nouvcout$ - $anccout$
            Si $deltacout = < 0$
            Alors itérer avec le nouveau placement
                $anccout = nouvcout$
            Sinon tirer un nombre aléatoire r
                Si $r < exp\ (- \frac{deltacout}{t}\ )$
                Alors itérer avec le nouveau placement
                    $anccout = nouvcout$
                Fsi
            Fsi
            $iteration = iteration + 1$
        FinTantque
        $temp = temp * a$

Annexe F

Résultats expérimentaux sur le choix de l'ordre lexicographique

Dans le paragraphe 6.4.4.3 , nous avons montrer que l'ordre lexicographique décroissants des tailles (b_i) des matrices du modèle (RAS), donne l'affectation des tâches qui à le coût minimum. Dans ce paragraphe, nous allons valider cela par des expérimentations.

L'expérience consiste à tester 100 modèles différents, Pour chaque modèle, on modifie aléatoirement 100 fois l'ordre lexicographique (100 échantillons). A chaque ordre lexicographique on calcul les coûts C_k et on les compare avec ceux de l'ordre lexicographique trie décroissant. Sur les 100 modèles expérimentés, l'ordre lexicographique trie décroissant à toujours le meilleur coût. Dans ce qui suit nous allons expérimenter 5 modèles différents. Pour chaque modèle, on modifie 5 fois l'ordre lexicographique, le 5^{ieme} ordre lexicographique correspond à l'ordre trie décroissant des tailles des matrices. Dans toutes les expériences, en suppose que la constantes r_{cc} est égal à 1 (cas où le coût de communication unitaire est du même qu'un opération de calcul).

1^{ier} modèle

La taille du modèle est $T = 11340$

Nombre de matrice $N = 4$

1^{ier} échantillon aléatoire:

b_1 b_2 b_3 b_4

(14) (15) (6) (9)

$C_2 = 4780$
$C_3 = 4714$
$C_4 = 4810$

C_k min : 4714 , le meilleur coût calculé : 4714

2ieme échantillon aléatoire

b_1	b_2	b_3	b_4
(15)	(6)	(14)	(9)

$C_3 = 4707$
$C_4 = 4810$

C_k min : 4707 , le meilleur coût calculé : 4707

3ieme échantillon aléatoire

b_1	b_2	b_3	b_4
(6)	(9)	(14)	(15)

$C_3 = 4716$
$C_4 = 4810$

C_k min : 4716 , le meilleur coût calculé : 4707

4ieme échantillon aléatoire

b_1	b_2	b_3	b_4
(9)	(6)	(15)	(14)

$C_3 = 4713$
$C_4 = 4810$

C_k min : 4713 , le meilleur coût calculé : 4707

5^{ieme} échantillon ordre lexicographique trié :

b_1	b_2	b_3	b_4
(15)	(14)	(9)	(6)

$C_2 = 4773$
$C_3 = 4707$
$C_4 = 4810$

C_k min : 4707 , le meilleur coût calculé : 4707

2^{ieme} modèle

La taille du modèle est $T = 24255$

Nombre de matrice $N = 6$

1^{ier} échantillon aléatoire:

b_1	b_2	b_3	b_4	b_5	b_6
(3)	(11)	(5)	(7)	(3)	(7)

$C_3 = 9633$
$C_4 = 8275$
$C_5 = 8393$
$C_6 = 8594$

C_k min : 8275 , le meilleur coût calculé : 8275

2^{ieme} échantillon aléatoire

b_1	b_2	b_3	b_4	b_5	b_6
(5)	(11)	(7)	(3)	(7)	(3)

$C_3 = 8685$
$C_4 = 8271$
$C_5 = 8389$

$C_6 = 8594$

C_k min : 8271 , le meilleur coût calculé : 8271

3^{ieme} échantillon aléatoire

b_1	b_2	b_3	b_4	b_5	b_6
(7)	(7)	(11)	(3)	(3)	(5)

$C_3 = 8253$
$C_4 = 8247$
$C_5 = 8391$
$C_6 = 8594$

C_k min : 8247 , le meilleur coût calculé : 8247

4^{ieme} échantillon aléatoire

b_1	b_2	b_3	b_4	b_5	b_6
(5)	(3)	(7)	(3)	(11)	(7)

$C_4 = 8646$
$C_5 = 8389$
$C_6 = 8594$

C_k min : 8389 , le meilleur coût calculé : 8247

5^{ieme} échantillon ordre lexicographique trié :

b_1	b_2	b_3	b_4	b_5	b_6
(11)	(7)	(7)	(5)	(3)	(3)

$C_3 = 8247$
$C_4 = 8226$
$C_5 = 8385$
$C_6 = 8594$

C_k min : 8226 , le meilleur coût calculé : 8226

3^{ieme} modèle

La taille du modèle est $T = 244944$

Nombre de matrice $N = 8$

1^{ier} échantillon aléatoire:

b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8
(6)	(9)	(3)	(3)	(7)	(4)	(3)	(6)

$C_3 = 113094$
$C_4 = 101538$
$C_5 = 94090$
$C_6 = 95338$
$C_7 = 97294$
$C_8 = 99319$

C_k min : 94090 , le meilleur coût calculé : 94090

2^{ieme} échantillon aléatoire

b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8
(6)	(3)	(3)	(4)	(9)	(6)	(3)	(7)

$C_4 = 97476$
$C_5 = 95172$
$C_6 = 95361$
$C_7 = 97294$
$C_8 = 99319$

C_k min : 95172 , le meilleur coût calculé : 94090

3^{ieme} échantillon aléatoire

b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8
(7)	(6)	(3)	(3)	(3)	(4)	(9)	(6)

$C_3 = 137594$
$C_4 = 104222$
$C_5 = 95906$
$C_6 = 95678$
$C_7 = 97292$
$C_8 = 99319$

C_k min : 95678 , le meilleur coût calculé : 94090

4^{ieme} échantillon aléatoire

b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8
(9)	(6)	(3)	(7)	(6)	(4)	(3)	(3)

$C_3 = 113097$
$C_4 = 93481$
$C_5 = 93529$
$C_6 = 95272$
$C_7 = 97297$
$C_8 = 99319$

C_k min : 93481 , le meilleur coût calculé : 93481

5^{ieme} échantillon ordre lexicographique trié :

b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8
(9)	(7)	(6)	(6)	(4)	(3)	(3)	(3)

$C_3 = 100501$
$C_4 = 91585$
$C_5 = 93247$
$C_6 = 95272$
$C_7 = 97297$
$C_8 = 99319$

C_k min : 91585 , le meilleur coût calculé : 91585

4^{ieme} modèle

La taille du modèle est $T = 680400$

Nombre de matrice $N = 10$

1^{ier} échantillon aléatoire:

b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}
(7)	(2)	(5)	(3)	(5)	(3)	(6)	(3)	(3)	(4)

$C_4 = 274518$
$C_5 = 262908$
$C_6 = 267912$
$C_7 = 270336$
$C_8 = 275688$
$C_9 = 281284$
$C_{10} = 286908$

C_k min : 262908 , le meilleur coût calculé : 262908

2^{ieme} échantillon aléatoire

b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}
(4)	(3)	(2)	(5)	(5)	(7)	(3)	(6)	(3)	(3)

$C_5 = 259778$
$C_6 = 265112$
$C_7 = 270602$
$C_8 = 275663$
$C_9 = 281288$
$C_{10} = 286908$

C_k min : 259778 , le meilleur coût calculé : 259778

3^{ieme} échantillon aléatoire

b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}
(3)	(3)	(7)	(5)	(6)	(3)	(2)	(5)	(3)	(4)

$C_4 = 273669$

C_5 = 261225
C_6 = 264645
C_7 = 270237
C_8 = 275691
C_9 = 281287
C_{10} = 286908

C_k min : 261225 , le meilleur coût calculé : 259778

4^{ieme} échantillon aléatoire

b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}
(5)	(4)	(2)	(3)	(3)	(3)	(7)	(3)	(5)	(6)

C_5 = 259867
C_6 = 265357
C_7 = 270691
C_8 = 275821
C_9 = 281287
C_{10} = 286908

C_k min : 259867 , le meilleur coût calculé : 259778

5^{ieme} échantillon ordre lexicographique trié :

b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}
(7)	(6)	(5)	(5)	(4)	(3)	(3)	(3)	(3)	(2)

C_3 = 268062
C_4 = 257100
C_5 = 259462
C_6 = 264952
C_7 = 270118
C_8 = 275680
C_9 = 281284
C_{10} = 286908

C_k min : 257100 , le meilleur coût calculé : 257100

5^{ieme} modèle

La taille du modèle est $T = 276480$

Nombre de matrice $N = 12$

1^{ier} échantillon aléatoire:

b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}
(4)	(4)	(2)	(2)	(4)	(5)	(2)	(2)	(3)	(3)	(3)	(2)

$C_5 = 113788$
$C_6 = 97474$
$C_7 = 99580$
$C_8 = 101038$
$C_9 = 102892$
$C_{10} = 105178$
$C_{11} = 107464$
$C_{12} = 109749$

C_k min : 97474 , le meilleur coût calculé : 97474

2^{ieme} échantillon aléatoire

b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}
(2)	(3)	(3)	(2)	(5)	(2)	(4)	(4)	(2)	(3)	(4)	(2)

$C_5 = 111012$
$C_6 = 96366$
$C_7 = 98590$
$C_8 = 100814$
$C_9 = 103064$
$C_{10} = 105184$
$C_{11} = 107466$
$C_{12} = 109749$

C_k min : 96366 , le meilleur coût calculé : 96366

3^{ieme} échantillon aléatoire

b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}
(2)	(3)	(2)	(2)	(4)	(4)	(2)	(3)	(5)	(3)	(4)	(2)

$C_6 = 107912$
$C_7 = 102530$
$C_8 = 102314$
$C_9 = 103064$
$C_{10} = 105184$
$C_{11} = 107466$
$C_{12} = 109749$

C_k min : 102314 , le meilleur coût calculé : 96366

4^{ieme} échantillon aléatoire

b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}
(2)	(4)	(2)	(4)	(5)	(3)	(2)	(4)	(3)	(2)	(2)	(3)

$C_5 = 99860$
$C_6 = 96332$
$C_7 = 98582$
$C_8 = 100806$
$C_9 = 102942$
$C_{10} = 105180$
$C_{11} = 107466$
$C_{12} = 109749$

C_k min : 96332 , le meilleur coût calculé : 96332

5^{ieme} échantillon ordre lexicographique trié :

b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}
(5)	(4)	(4)	(4)	(3)	(3)	(3)	(2)	(2)	(2)	(2)	(2)

$C_4 = 97585$
$C_5 = 94057$
$C_6 = 96289$
$C_7 = 98521$

$C_8 = 100627$
$C_9 = 102909$
$C_{10} = 105191$
$C_{11} = 107467$
$C_{12} = 109749$

C_k min : 94057 , le meilleur coût calculé : 94057

Annexe G

Recouvrement des communications par les calculs

Dans ce paragraphe, nous poursuivons les expérimentations du paragraphe 1.5 pour mesurer le taux de recouvrement des communications par des calculs. On va s'intéresser au recouvrement dans le cas où la communication est des types suivants :

- Communication sur un lien en bi-directionnel.
- Communication sur 2 liens en mono et bi-directionnel.
- Communication sur 3 liens en mono et bi-directionnel.
- communication sur 4 liens en mono- directionnel.

Les cas manquant sont déja traiter dans le chapitre 1. On rappelle les notations suivantes:

Notation :

- On note t_{calc} : le temps d'exécution de la partie calcul.
- t_{comm} : le temps d'exécution de la partie communication.
- $t_{calc+comm}$: le temps résultant de la partie calcul suivit séquentiellement par la partie communication.
- $t_{calc//comm}$: le temps résultant de l'exécution en parallèle de la partie calcul et la partie communication (primitive ProcPar).

On défini le taux de recouvrement du calcul par les communications comme :

$$Recouv = \frac{t_{calc+comm} - t_{calc//comm}}{t_{comm}} * 100$$

Dans les expériences qui sont résumés dans les tableaux suivants, nous avons mesuré le taux de recouvrement de communication par du calcul. Ces mesures représentent des moyennes sur un échantillon de taille 10 avec resynchronisation des Transputers émetteur et récepteur.

Recouvrement du calcul par une communication sur 1 lien bi-directionnel

$t_{calc+comm}$ (s)	0.228992	0.344190	0.574656	0.804992	1.035392	1.265792
$t_{calc//comm}$ (s)	0.120512	0.235712	0.466112	0.696436	0.926836	1.157236
t_{comm} (s)	0.113792	0.113792	0.113792	0.113792	0.113792	0.113792
$Rapport\ \frac{t_{calc}}{t_{comm}}$	1	2	4	6	8	10
$Recouv(\%)$	95.3	95.3	95.4	95.4	95.4	95.4

D'après le tableau précédent, on remarque un bon recouvrement des communication par le calcul de l'ordre de 95%.

Recouvrement du calcul par une communication sur 2 liens mono-directionnel

$t_{calc+comm}$ (s)	0.165184	0.251584	0.424384	0.597248	0.769984	0.942848
$t_{calc//comm}$ (s)	0.090752	0.177152	0.349888	0.522760	0.695488	0.868285
t_{comm} (s)	0.078784	0.078784	0.078784	0.078784	0.078784	0.078784
$Rapport\ \frac{t_{calc}}{t_{comm}}$	1	2	4	6	8	10
$Recouv(\%)$	94.4	94.4	94.5	94.5	94.5	94.6

D'après le tableau précédent, on remarque un taux recouvrement des communication par le calcul de l'ordre de 94%.

Recouvrement du calcul par une communication sur 3 liens mono-directionnel

$t_{calc+comm}$ (s)	0.165248	0.251584	0.424384	0.597184	0.770048	0.942784
$t_{calc//comm}$ (s)	0.093358	0.179648	0.352256	0.525056	0.697961	0.87069
t_{comm} (s)	0.078784	0.078784	0.078784	0.078784	0.078784	0.078784
$Rapport\ \frac{t_{calc}}{t_{comm}}$	1	2	4	6	8	10
$Recouv(\%)$	91.3	91.3	91.5	91.5	91.5	91.5

D'après le tableau précédent, on remarque un taux recouvrement des communication par le calcul de l'ordre de 94%.

Recouvrement du calcul par une communication sur 3 liens bi-directionnel

$t_{calc+comm}$ (s)	0.130306	0.245277	0.475392	0.705826	0.936200	1.166603
$t_{calc//comm}$ (s)	0.090752	0.177152	0.349888	0.522760	0.695488	0.868285
t_{comm} (s)	0.114432	0.114432	0.114432	0.114432	0.114432	0.114432
$Rapport\ \frac{t_{calc}}{t_{comm}}$	1	2	4	6	8	10
$Recouv(\%)$	86.8	87.8	87.2	87.2	87.2	87.2

D'après le tableau précédent, on remarque un taux recouvrement des communication par le calcul de l'ordre de 87%.

Recouvrement du calcul par une communication sur 4 liens mono-directionnel

$t_{calc+comm}$ (s)	0.165248	0.251584	0.423840	0.597248	0.770048	0.942848
$t_{calc//comm}$ (s)	0.094993	0.180864	0.353648	0.526400	0.699202	0.871978
t_{comm} (s)	0.078848	0.078848	0.078848	0.078848	0.078848	0.078848
$Rapport\ \frac{t_{calc}}{t_{comm}}$	1	2	4	6	8	10
$Recouv(\%)$	89.1	89.7	89.7	89.8	89.8	89.8

D'après le tableau précédent, on remarque un taux recouvrement des communication par le calcul de l'ordre de 89%.

Remarque

La dégradation de performance du recouvrement et principalement dû à la bande passante du bus d'accés à la mémoire. L'accés au bus est partagé entre l'unité de calcul

et les 4 DMA sur les liens de communications. Il est clair alors que cette dégradation et d'autant plus forte que le nombre de liens utilisés en parallèle est grand et qu'ils soient utiliser en bi-directionnel.

Bibliographie

[1] Brigitte Plateau Abderezak Touzene. Optimal multinode broadcast on a mesh connected graph with reduced bufferization. In *Distributed Memory Computing, 2nd European Conference, EDMCC2*, Munich,FRG, 1991.

[2] K. Atif. *Thèse INPG.* INP Grenoble, Sept 1992.

[3] B.Lubachevsky and D.Mitra. A chaotic asynchrounous algorithm for computing the fixed point of nonnegative matrix of unit spectral radius. *Journal of ACM*, 33(1), Jan 1986.

[4] S. Bokhari. On the mapping problem. *IEEE Transactions on Computers*, C-30(3), March 1981.

[5] W-L. Cao and W.J. Stewart. Iterative aggregation/disaggregation technics for nearly uncoupled markov chains. *J. Asso. Comp. Math*, 32(3, 702-719), 1985.

[6] K.M. Chandy and Misra. Distributed simulation : A case study in design and verification of distributed programs. *IEEE. Trans. Software Eng.*, 5(440-452), 1979.

[7] K.M. Chandy and Misra. Asynchronous distributed simulation via a sequentiel of parallel computations. *Comm of ACM*, 24(4), 1981.

[8] M. Charnay. Iteration chaotic sur un produits déspace metrique. *Thèse de* 3^{ieme} *cycle U Claude Bernard, Lyon*, 1975.

[9] G. Chiola. Greatspn 1.5 software architecture. *In G. Balbo (Ed), Modelling Techniques and Tools for Computers Performance evaluation, Italy*, 1991.

[10] P.J. Courtois. *Decomposabilite Queueing and Computer System Applications.* Academic Press, Orlando, Florida, 1977.

[11] W. Miranker D. Chazan. Chaotic relaxation. *Linear Algebra and Appl*, (2), 1969.

[12] W. Dally and C. Seitz. Deadlock-free message routing in multiprocesseur interconnetion networks. *IEEE Transactions on Computers*, 36(5), 1987.

[13] M. Davio. Kronecker products and shuffle algebra. *IEEE Transactions on Computers*, C-30(2), February 1981.

[14] C.S.Schlten and E.W. Dijktra, Termination detection for diffusing computations. *Information Processing letter*, (11), 80.

[15] J.N.Tsitsiklis D.P.Bertsekas. *Parallel and Distributed Computation.* Printice-Hall. International Edition, 1989.

[16] J.N.Tsitsiklis D.P.Bertsekas. A survey of some aspect of parallel and distributed iterative algorthm. *CICS Publication Office M.I.T*, Juin 90.

[17] E. Gelenbe J. Labetoulle R. Marie G. Pujolle et W.J. Stewart. *Réseaux de files d'attente - modélisation et traitement numérique.* Ed. des hommes et techniques, Monographies informatiques de l'AFCET, 1980.

[18] P .Fraigniaud. *Communications dans les architectures distribuées.* Thèse , ENSL (Lyon), 1991.

[19] Hervé Frydlender. *Parallélisation de l'algorithme de Rétropropagation du Gradient Récursif.* Rapport téchnique L.M.C, 1991.

[20] S.M. Hedetniemi, S.T Hedetneimi, and A.L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18, 1986.

[21] Telmat Informatique. T.node overview. *Telmat Research*, Juin 1989.

[22] INMOS. *The transputer Data Book.* Printed at Press Ltd, Bath, 1989.

[23] D.R. Jefferson and H.A. Sowizaral. Fast concurrent simulation using the time warp mechanism, part 1 : Local control. *Tech. Rep. The Rand Corp., Santa Monica, CA*, 1982.

[24] A.Touzene K. Atif. *Résolution des systèmes QBD.* DEA université de Paris 11 Orsay, 1988.

[25] P. Kermani and L. Kleinrock. Virtuel cut-through : a new computer communication switching technique. *Computers Networks*, 3, 1979.

[26] L. Kleinrock. *Queueing Systems, Vol 1. : Theory.* J. Wiley, 1975.

[27] L. Lamport K.M. Chandy. Distributed snapshots : determining global states of distributed systems. *Journal of ACM Transation on Computing systems*, (3), 1985.

[28] D. Kumar. Systems with low distributed simulation overhead. *J. IEEE. Trans. on Parall. and Dist. Syst.*, 3(2, 155-165), 1992.

[29] G. Balbo M.A. Marsan and G. Conte. A class of generalised stochastic petri nets. *ACM. Trans. Comput. Systems*, 2(93-122), 1984.

[30] D. Mailles. *Files d'attente descreptives pour la modélisation de la synchronisation dans les systèmes informatique, Thèse de doctorat d'etat.* Universite de Paris 6, 1987.

[31] Gerard M.Baudet. Asynchronous iterative methods for multiprocessors. *Journal of ACM*, 25(2), April 1978.

[32] A M.Farley and A.Proskurowski. Gossip in grid graphs. *Journal of Combinatorics information and System Sciences*, 5(2), 1980.

[33] S. Petiton. Parallel qr algorithms for iterative subspace methods on connection machine. In *In proceedings of the fourth SIAM conference on parallel processing for sientific computing*, Dec 1989.

[34] A. Touzene B. Plateau. Mesures de performance des communications du meganode à 128 transputers. *La lettre du transputer et des calculateur distribués*, (7), Sept 1990.

[35] B. Plateau. On the stochastic structure of parallelism and synchronization models for distributed algorithms. In *ACM Sigmetrics Conference on Measurement and Modelling of Computer systems*, Austin, August 1985.

[36] B. Plateau and J.M. Fourneau. A methodology for solving markov models of parallel systems. *Journal of Parallel and Distributed Computing*, 12(370-387), 1991.

[37] D.F. McAllister R. Koury and W.J. Stewart. Methods for computing stationary distibutions of nearly completely markov chains. *SIAM , J. Alg. Disc. Math*, 5(2, 164-186), 1984.

[38] F. Robert. Contration en norme vectorielle, convergence d'itération chaotique pour des équations non linéares de points fixes à plusieurs variables. *Linear Algebra and its Applications*, (13), 1976.

[39] F. Robert. Iterations discretes asynchrones. Technical Report Rapport Thechnique 671M, IMAG, Universite de Grenoble , France, 1987.

[40] J. L. Roch. *Calcul formel et parallélisme : L'architecture du système PAC et et son arithmetique rationnelle.* Thèse INP Grenoble, Dec 1985.

[41] M.P. Vecchi S. Kirkpatrick, C.D. Gelatt. Optimization by simulating annealing. *Science*, 220, May 1983.

[42] Y. Saad. Gaussian elimination on hybercubes. *Parallel Algorithms and Architecture, Eds M. Cosnard et all, North-Holland*, 1986.

[43] Y. Saad. Data communication in parallel architectures. *Parallel Computing*, (11), 1989.

[44] Y. Saad and M. Shultz. A generalised minimal residual algorithm for solving non-symetric lineair systems. *SIAM, J. Sci. Stat. Comput.*, 7(3 : 856-869), 1986.

[45] P.J. Schweitze. Methods for large markov chains. *International workshop on applied mathemathics and performance reliablility models of computer connection systems, univesity of Pisa , Italy*, (225-234), 1983.

[46] H.A. Simon and A. Ando. Aggregation of variables dynamic systems. *Econometrica*, 29(111-138), 1961.

[47] W.J Stewart. Iterative methods . *Technical Report RR282, Departement of Computer Science, N.Carolina University, USA*, 1989.

[48] W.J Stewart. Itérative methods. *Departement of Computer Science, N.C Univ Raleigh*, 89.

[49] D.M. Topkis. All-to-all broadcast by flooding in communication networks. *IEEE Transactions on Computers*, 38(9), September 1989.

[50] B. Tourancheau. *Algorithmique Parallèle pour machines à mémoires distribuée.* Thèse INP Grenoble, Dec 1989.

[51] B. Plateau D. Trystram. Optimal total exchange on 3d-grid. *Information Processing Lettres, North-Holland*, 42, 1992.

[52] M. Cosnard M. Marrakchi Y. Robert D. Trystram. Parallel gaussian elimination on mimd computers. *Parallel Computing, North-Holland*, (6), 1988.

[53] R. S. Varga. *Matrix Iteratives Analysis.* Prentice-hall, Englewood Cliffs, N.J, 1963.

[54] G. Villard. *Calcul formel et parallélisme : Résolution de Systèmes linéaire.* Thèse INP Grenoble, Dec 1988.

[55] P.Bouvry H. Frylender J. Prevost J. Roch A. Touzene G. Villard. Manuel du meganode. Technical Report Rapport de Recherche RT-63, LMC-IMAG, 1991.

[56] D. Delesalle D. Trystram D. Wenzek. Optimal total exchange on simd distributed memory hypercube. In *Distributed Merory Computing*, Portland, May 1991.

[57] B. Philippe W.J Stewart, Y. Saad. Numerical methods in markov chains modelling. *Technical Report 495, IRISA, Rennes, France*, 1991.

Grenoble, le 10 Septembre 1992

INPG

:OLE DOCTORALE

aire suivie par Michèle SIMEON
l. 76.57
4525

Réf. :

ojet :

AUTORISATION de SOUTENANCE

Vu les dispositions de l'arrêté du 23 Novembre 1988 relatif aux Etudes Doctorales
Vu les rapports de présentation de :

Mr BERNARD Guy Professeur à l'I.N.T. Habilité à diriger des recherches EVRY
Mr KONIG J.Claude Maître de conférence Habil. à diriger des recherches ORSAY

Mr TOUZENE Abderezak

est autorisé(e) à présenter une thèse en soutenance en vue de l'obtention du diplôme de DOCTEUR de l'INSTITUT NATIONAL POLYTECHNIQUE de GRENOBLE* spécialité :

Informatique

M. GARNIER
Vice-Président Recherche