Table des matières

1 Introduction générale **9**

I Les algorithmes itératifs parallèles 13

1 Modèles des algorithmes itératifs parallèles **15**
1.1 Les itérations parallèles asynchrones 15
1.1.1 Modèle informatique 15
1.1.2 Modèle mathématique 15
1.1.3 Exemple et interprétation 17
1.2 Les itérations parallèles asynchrones avec communication flexible 19
1.2.1 Sous-applications et sur applications 20
1.2.2 Un premier modèle d'itérations asynchrones avec communication flexible 23
1.2.3 Un second modèle d'itérations asynchrones avec communication flexible 24
1.3 Quelques résultats de convergence 24
1.3.1 Problèmes de point fixe 25
1.3.1.1 Contraction 25
1.3.1.2 Ordre partiel 26
1.3.2 Systèmes d'équations non linéaires avec ordre partiel 26
1.3.3 Systèmes singuliers 28
1.4 Conclusion 31

2 Mise en œuvre parallèle **33**
2.1 Introduction 33
2.2 Classification des architectures parallèles utilisées 34
2.2.1 Architecture parallèle à mémoire partagée 35
2.2.2 Architecture parallèle à mémoire distribuée 36
2.2.3 Réseau de machines 36
2.3 Mise en œuvre sur CRAY T3E 37
2.3.1 Description matérielle 37
2.3.2 Mise en œuvre utilisant la bibliothèque MPI 38
2.3.3 Mise en œuvre utilisant la bibliothèque SHMEM 42

2.4 Mise en œuvre sur une SMP . . . 43
2.4.1 Description matérielle . . . 43
2.4.2 Mise en oeuvre en utilisant les processus legers . . . 46
2.5 Mise en œuvre sur un réseau de SMP . . . 47
2.5.1 Description matérielle . . . 47
2.5.2 Mise en œuvre avec passage de message et multithreading . . . 47
2.5.2.1 Conception générale . . . 47
2.5.2.2 Cas de l'utilisation de MPICH . . . 50
2.5.2.3 Cas de l'utilisation de LANDA . . . 50
2.5.2.4 Cas de l'utilisation d'ATHAPASCAN . . . 51
2.6 Mesures de performance et paramètres importants . . . 51
2.7 Conclusion . . . 52

II Applications et étude de performance 53

1 Résolution d'un problème de flot convexe dans les réseaux 55
1.1 Introduction . . . 55
1.2 Les problèmes de flot à critère convexe . . . 55
1.2.1 Formulation du problème . . . 55
1.2.2 Une formulation duale . . . 57
1.2.3 L'ensemble des solutions duales optimales. . . . 58
1.3 Méthodes itératives de résolution. . . . 58
1.3.1 Principe de la méthode de relaxation . . . 59
1.3.2 Méthode de type gradient . . . 60
1.3.3 Test de terminaison . . . 60
1.4 Mise en œuvre séquentielle . . . 61
1.4.1 Présentation de l'algorithme séquentiel . . . 61
1.4.2 Problèmes traités . . . 61
1.4.3 Comparaison entre la méthode de type gradient et la méthode du gradient conjugué . . . 62
1.5 Etude de performance . . . 62
1.5.1 Décomposition du problème . . . 62
1.5.2 Etude de performance sur CRAYT3E . . . 63
1.5.3 Etude de performance sur SMP . . . 70
1.5.4 Etude de performance sur un réseau de SMP . . . 71
1.6 Les facteurs qui interviennent sur les performances de la mise en œuvre parallèle asynchrone . . . 73
1.6.1 Effet du déséquilibre des charges de calcul . . . 73
1.6.2 Effet de la granularité . . . 74
1.6.2.1 Effet du temps de latence . . . 74
1.6.2.2 Effet du débit de communication . . . 75
1.6.2.3 Effet de l'environnement de programmation parallèle . . . 75

1.6.3 Effet du degré d'asynchronisme 75
1.7 Conclusion 77

2 Résolution d'un problème de commande optimale 79
2.1 Introduction 79
2.2 Equations canoniques d'un problème de commande optimale 80
2.3 Etude de la régulation d'un processus thermique 80
2.3.1 Position du problème 80
2.3.2 Discrétisation du problème 81
2.3.2.1 Cas d'un problème sans contrainte 81
2.3.2.2 Cas d'un problème avec contrainte 83
2.3.3 Détermination de la commande optimale en séquentiel 83
2.3.3.1 Schémas itératifs à deux niveaux 84
2.3.3.2 Mise en œuvre séquentielle 85
2.3.3.3 Résultats expérimentaux 85
2.4 Schémas itératifs parallèles asynchrones 86
2.4.1 Cas d'une application de point fixe contractante 87
2.4.1.1 Décomposition par point du problème 87
2.4.1.2 Décomposition par bloc du problème 87
2.4.1.3 Résultats de convergence de la méthode de l'état adjoint avec échange asynchrone 88
2.4.2 Cas d'une application de point fixe monotone 88
2.4.2.1 Sous-solution et sur solution de départ 89
2.4.2.2 Un exemple d'application de point fixe monotone 91
2.4.2.3 Un résultat de convergence des itérations asynchrones avec communication flexible 92
2.4.2.4 Terminaison des itérations asynchrones avec communication flexible 95
2.5 Etude de performance 95
2.5.1 Décomposition en sous-domaines du problème 96
2.5.2 Etude de performance sur CRAYT3E 97
2.5.3 Etude de performance sur SMP 99
2.5.4 Etude de performance sur un réseau de SMP 101
2.6 Conclusion 103

3 Résolution d'un système Markovien 105
3.1 Introduction 105
3.2 Génération d'un système Markovien 105
3.2.1 Chaînes de Markov à temps discrets 106
3.2.2 Chaîne de Markov à temps continu 107
3.2.3 Passage des taux de transition aux probabilités de transition 107
3.2.4 Application à un processus de naissance et de mort 108
3.3 Méthode itératives de résolution 108

3.3.1 Schéma itératif de Gauss-Seidel 109
3.3.2 Méthode de sur relaxation 109
3.3.3 Méthode d'agrégation-désagrégation 110
3.3.4 Résultats numériques 111
3.4 Parallélisation du schéma itératif de Gauss-Seidel 112
3.4.1 Méthode d'Amodio et Mazzia 112
3.4.1.1 Transformation de la matrice de Gauss-Seidel 112
3.4.1.2 Schéma itératif de Gauss-Seidel modifié 113
3.4.2 Méthode de coloration en rouge et noir 113
3.5 Les itérations parallèles 115
3.5.1 Décomposition en sous-domaines 115
3.5.2 Terminaison des itérations asynchrones 117
3.5.2.1 Méthode de terminaison avec acquittement systématique . 117
3.5.2.2 Méthode de terminaison sans acquittement systématique . 119
3.6 Etude de performance 121
3.6.1 Etude de performance sur CRAY T3E 121
3.6.2 Etude de performance sur SMP 125
3.6.3 Etude de performance sur un réseau de SMP 125
3.7 Conclusion 126

4 Conclusion générale 129

A Rappels mathematiques 131
A.1 Rappels d'algèbre linéaire 131
A.2 Notions de M-fonctions 132
A.3 Notions de H-accrétivité 134
A.4 Applications contractantes 136
A.5 Méthode du gradient conjugué 137

Bibliographie 141

Table des figures

1.1 Comportement d'un algorithme asynchrone sur deux processeurs. 16
1.2 graphe d'évolution des composantes 18
1.3 Effet des différences de durée des phases de calcul 18
1.4 Effet des délais de communication 19
1.5 Comportement d'un algorithme synchrone sur deux processeurs. 20

2.1 Classification de Flynn des architectures parallèles 34
2.2 architecture MIMD à mémoire partagée 35
2.3 machine parallèle virtuelle 37
2.4 architecture de la machine CRAY T3E 38
2.5 modèles d'itérations synchrones et asynchrones avec communication flexible sur une SMP 47
2.6 modèle des itérations synchrones sur un réseau de SMP 48

1.1 Un graphe connexe orienté 56
1.2 décomposition du réseau sur les processeurs 64
1.3 Problème de flot turbulent 66
1.4 Problème de flot de communication 67
1.5 Problème de flot quadratique 68
1.6 Comparaison des performances des itérations asynchrones par sous-domaines pour les différents problèmes 68
1.7 Comparaison des performances des itérations asynchrones par sous-domaines et asynchrone par point 69
1.8 Comparaison des performances des itérations asynchrones par point et asynchrone avec communication flexible pour le cas d'un degré maximum égal à 4 69
1.9 Comparaison des performances des itérations asynchrones par point et asynchrone avec communication flexible pour le cas d'un degré maximum égal à 32 70
1.10 Variation du nombre moyen d'itérés partiels 76
1.11 Comparaison entre la meilleure politique d'envoi des itérés partiel avec shmem_put() et la mise en œuvre utilisant shmem_get() 77

2.1 Le sous-différentiel de la fonction indicatrice des commandes admissibles . 83

2.2 Commande optimale calculé en séquentiel en fonction du nombre d'itérations 86
2.3 Cas d'une application de point fixe contractante 90
2.4 Cas d'une application de point fixe qui engendre deux séquences monotones de sous-solutions et de sur solutions 91
2.5 algorithme de terminaison 96
2.6 Décomposition en sous-domaine 97
2.7 Temps d'exécution sur le CRAYT3E 100
2.8 Accélération des mises an œuvre parallèles sur le CRAYT3E 100

3.1 processus de naissance et de mort 108
3.2 Charge très déséquilibrée 117
3.3 équilibrage avec l'algorithme 10 117
3.4 Modèle de l'algorithme de détection de la terminaison avec acquittement . 118
3.5 Modèle de terminaison sans acquittement 119

Liste des tableaux

1.1 Problèmes de flot : comparaison de la méthode de type gradient et de la méthode du gradient conjugué 63
1.2 Performance sur une SMP 70
1.3 Le temps d'exécution et l'efficacité de la mise en œuvre parallèle sur une réseau Ethernet en utilisant MPICH 71
1.4 Le temps d'exécution et l'efficacité de la mise en œuvre parallèle sur une réseau Myrinet en utilisant MPICH-gmm 72
1.5 Le temps d'exécution et l'efficacité de la mise en œuvre parallèle sur une réseau Myrinet en utilisant LANDA 72
1.6 Le temps d'exécution et l'efficacité de la mise en œuvre parallèle sur une réseau Myrinet en utilisant ATHAPASCAN 73

2.1 nombre moyen d'itérés partiels par itération sur 4 processeurs du CRAYT3E 100
2.2 Performance sur une SMP 101
2.3 Performance sur un réseau Myrinet de SMP en utilisant MPICH-gmm . . 102
2.4 Performance sur un réseau Ethernet de SMP en utilisant MPICH 102

3.1 Comparaison en séquentiel des méthodes itératives de résolution 111
3.2 Performance des itérations parallèles synchrones sur CRAYT3E 121
3.3 Performance des itérations parallèls synchrones sur CRAYT3E en appliquant la méthode d'Amodio et Mazzia 122
3.4 Performance des itérations synchrones sur CRAYT3E avec une technique de coloration en rouge et noir 123
3.5 Performance des itérations parallèles asynchrones sur le CRAYT3E avec la méthode de terminaison de Bertsekas 124
3.6 Performance des itérations parallèles asynchrones sur le CRAYT3E avec la méthode de terminaison sans acquittement systématique 124
3.7 Comparaison de performance des itérations parallèles synchrones et asynchrones sur SMP 125
3.8 Comparaison de performance des itérations parallèles synchrones et asynchrones sur un réseau myrinet de SMP 126

Chapitre 1

Introduction générale

La rapidité de calcul représente un défit majeur dans le monde scientifique et industriel. L'accroissement des performances des calculateurs a été le fruit de l'évolution technologique en microélectronique. Toutefois, les performances des machines monoprocesseurs sont insuffisantes pour certaines applications industrielles qui nécessitent un très grand volume de calcul. Ces applications touchent à des domaines très sensibles à savoir la construction des avions ou des véhicules spatiaux, la prévision météorologique, etc....
En réponse à ces besoins de calcul, de nouvelles architectures vectorielles et multiprocesseurs ont été conues dans les années 80 afin de partager le volume de calcul sur plusieurs unités de traitement ou sur plusieurs processeurs. Simultanément, de nombreuses méthodes numériques parallèles ont été proposées. Parmi ces méthodes, les algorithmes itératifs parallèles tiennent une place particulièrement importante. Afin d'obtenir une parallélisation efficace, une grande partie des travaux de recherche sur le parallélisme s'est focalisée sur le choix du type de synchronisation et l'équilibrage des charges de calcul. Très rapidement des travaux de recherche ont démarré afin de proposer des algorithmes itératifs asynchrones performants.

Les méthodes itératives asynchrones ont été introduites afin d'éliminer les temps d'attente dus au déséquilibre des charges de calcul et de permettre aux processeurs de collaborer à la résolution d'un problème, chaque processeur allant à son propre rythme. Les premiers travaux théoriques sur les algorithmes itératifs asynchrones (cf. [CM69]) ainsi que les premières simulations (cf. [Ros69]) datent des années soixante et concernent essentiellement la résolution des systèmes linéaires. Depuis, de nombreuses études ont été menées afin d'appliquer ces algorithmes à la résolution d'équations aux dérivées partielles discrétisées (cf. [Mie75a], [Bau78], [Spi83], [MiS85b], [SBA89], [MSGL91], [GS91], [GS92], [FS94], [Fro95], [FS97], [FSc97], [FSS97], [FS98], et [FS00]) d'équations différentielles ordinaires (cf. [Mit87]), de problèmes aux deux bouts (cf. [Spi84] et [Spi86]), de problèmes de programmation dynamique (cf. [Ber82]), ou d'optimisation de type flot dans les réseaux (cf. [BeE87], [Elb93] et [Elb96]), à la résolution des systèmes d'équations non linéaires (cf.

[Elb90] et [Elb94b]), à la résolution des systèmes différentiels algébriques (cf. [Bah91] et [Bah96]), et à la résolution des systèmes singuliers (cf. [Bah98a], [Bah98b] et [Bah99a]).

Une nouvelle classe d'algorithmes itératifs asynchrones : les itérations asynchrones avec communication flexible a été proposée récemment dans [MES98] pour la résolution des systèmes d'équations non linéaires issus de la discrétisation d'équations aux dérivées partielles : problèmes de diffusion non linéaires ou d'Hamilton-Jacobi-Belman dans un contexte de M-fonction. Contrairement aux itérations parallèles asynchrones classiques pour lesquelles les échanges de données entre processeurs portent nécessairement sur le résultat de la réactualisation de la valeur d'une composante ou d'un bloc de composantes du vecteur itéré, les itérations asynchrones avec communication flexible autorisent des échanges de données relatifs à des réactualisations partielles entre les différents processeurs. Ainsi, chaque processeur peut avoir accès à n'importe quel instant et sans aucune règle fixée à la valeur courante des composantes du vecteur itéré réactualisées par les autres processeurs. Cette nouvelle classe a été aussi appliquée à la résolution d'un problème de flot convexe dans les réseaux dans un contexte un peu plus général (cf. [ESMG96]).

La mise en œuvre d'algorithmes itératifs parallèle a été effectuée sur de nombreuses architectures parallèles. En particulier la mise en œuvre des itérations asynchrones a été effectuées sur la mémoire partagée CMM (cf. [Bau78]), sur un réseau de 6 micro-ordinateurs à base de processeurs M6809 de Motorola et sur une machine à base de transputer Tnode de Telmat (cf. [Elb84], [GBEB82], [BACEB84], [Elb89], [Elb93] et [ESMG96]), sur les machines à mémoire partagée Allient FX/8 (cf. [CZ91], [GS91] , [GS92] et [BMPS95]) et IBM 3090-400 (cf. [GS91] et [GS92]) ainsi que sur une machine SMP (cf. [Gaz98], [EGJSM98] et [JEG98]), sur la machine à mémoire distribuée IBM SP2 (cf. [Gui97]), le CRAY T3D et le CRAY J90 (cf. [FSS97]) et sur une station SGI Power Challenge R10000 à 8 processeurs (cf. [REB98]).

La diversité des études théoriques, applications et mises en œuvre montre que ce domaine de recherche est très actif. Vu l'évolution technologique continue des architectures parallèles, il est particulièrement motivant de mettre en œuvre ces algorithmes parallèles sur les architectures les plus récentes. De même il est de première importance d'étudier les performances des algorithmes itératifs parallèles en fonction de l'architecture et de l'environnement de programmation utilisés ceci afin de valider les schémas de calcul.

Notre contribution porte sur la mise en œuvre des algorithmes itératifs parallèles sur trois types d'architecture : un supercalculateur CRAY T3E à mémoire distribuée, une machine multiprocesseur symétrique "SMP" et un réseau rapide de machines SMP. Nous avons envisagé d'utiliser les "clusters", car ils représentent actuellement une nouvelle orientation des applications parallèles qui permettent de considérer un réseau comme étant une machine parallèle virtuelle. Le choix de ces différentes architectures nous a permis d'étudier les facteurs qui interviennent dans les performances des itérations parallèles synchrones, asynchrones et asynchrones avec communication flexible. Pour chaque type d'architecture, nous donnons les primitives employées pour mettre en œuvre efficacement le parallélisme entre les différents processeurs.

Afin d'étudier les performances des itérations parallèles, nous avons considéré trois types d'applications de nature différente et qui permettent de mettre en évidence les différents problèmes liés à la parallélisation des méthodes itératives de résolution : un problème de flot convexe dans les réseaux, un problème de commande optimale et un système markovien. Ce travail représente une suite des travaux de recherche effectués en partie dans le cadre d'une collaboration avec les professeurs Jean-Claude Miellou du Laboratoire de Calcul Scientifique de Besanon et Pierre Spiteri de l'ENSEEHIT-IRIT de Toulouse ainsi que dans le cadre d'un BQR inter INP en collaboration avec les professeurs Pierre Spiteri et Brigitte Plateau de l'IMAG.

Le manuscrit est divisé en deux parties :

la première partie concerne les algorithmes itératifs parallèles synchrones, asynchrones et asynchrones avec communication flexible. Nous donnons au chapitre I.1 un modèle de chaque type d'algorithme itératif parallèle et nous présentons à la fin de ce chapitre quelques résultats théoriques de convergence des itérations asynchrones et asynchrones avec communication flexible. Le chapitre I.2 porte sur les différentes mises en œuvre de ces algorithmes itératifs parallèles sur les trois types d'architectures citées ci-dessus.

Dans la seconde partie nous présentons une étude de performances des algorithmes itératifs parallèles pour les trois types d'applications citées plus haut. En particulier le chapitre II.1 concerne la résolution d'un problème de flot convexe dans les réseaux. Nous mettons en œuvre dans ce chapitre des algorithmes de type gradient afin de résoudre un problème dual réduit. Le chapitre II.2 présente la résolution d'un problème de commande optimale relatif à la régulation d'un four. Nous effectuons la résolution en appliquant la méthode de l'état adjoint par relaxation introduite dans [Mie72], [Spi84], [LMS86] et [Jac77]. Nous proposons pour ce cas d'application un résultat de convergence des itérations asynchrones avec communication flexible. Dans le chapitre II.3 nous considérons un système markovien et l'on résout plus précisément un système singulier à l'aide de la méthode de relaxation et la méthode d'agrégation-désagrégation (cf. [Leg87], [BLA87] et [Leg92]). Nous comparons les performances des itérations parallèles en utilisant un algorithme de Gauss Seidel parallèle modifié (cf. [AM95]) et en utilisant la technique de coloration en rouge et noir.

Première partie

Les algorithmes itératifs parallèles

Chapitre 1

Modèles des algorithmes itératifs parallèles

Nous présentons dans ce chapitre des modèles mathématiques et informatiques des différents schémas itératifs parallèles ainsi que quelques rappels des principaux résultats théoriques de convergence des itérations asynchrones.

1.1 Les itérations parallèles asynchrones

1.1.1 Modèle informatique

Pour illustrer les algorithmes itératifs asynchrones nous considérons la résolution d'un problème de point fixe $x^* = F(x^*)$, où F est une application de $\mathbb{R}^n$ dans $\mathbb{R}^n$. Sur la figure (1.1) le réseau de Petri montre que chaque processeur itère à son propre rythme sans ordre ni synchronisation entre les processeurs. Sur le diagramme temporel les rectangles blancs numérotés représentent les phases de réactualisation et les rectangles hachurés représentent les phases de communication. Le numéro d'itération est incrémenté au commencement de chaque nouvelle phase de réactualisation. Un algorithme asynchrone n'admet pas de période d'inactivité et les phases de réactualisation s'enchaînent rapidement.

1.1.2 Modèle mathématique

Dans ce sous-paragraphe nous présentons un modèle d'algorithmes itératifs asynchrones qui est dérivé des modèles de Chazan et Miranker (cf. [CM69]), de Miellou (cf. [Mie75b]) et de Baudet (cf. [Bau78]).

Définition 1.1.1 *Une itération asynchrone associée à l'application F et au point initial $x(0) \in \mathbb{R}^n$ est une suite $\{x(k)\}$, $k = 0,1,...$ de vecteurs de $\mathbb{R}^n$ définie de manière récursive par :*

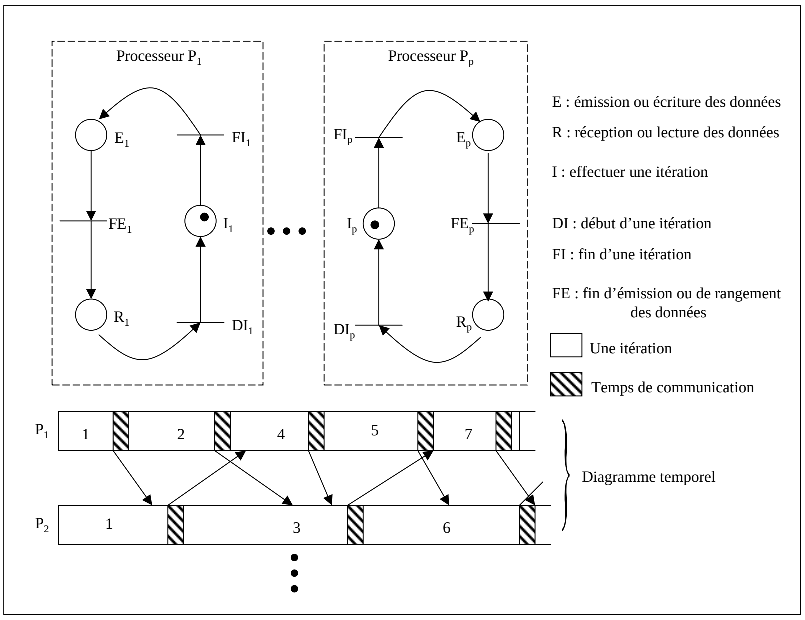

Fig. 1.1 – *Comportement d'un algorithme asynchrone sur deux processeurs.*

$\forall k \in \mathbb{N}, \forall l \in \{1,2,...,\alpha\}$,

$$x_l(k+1) = \begin{cases} F_l(x_1(\tau_1(k)),...,x_\alpha(\tau_\alpha(k))) \ si \ l \in s(k), \\ x_l(k) \ si \ l \notin s(k), \end{cases} \tag{1.1}$$

où les suites s et τ satisfont les conditions suivantes pour $l = 1,...,\alpha$:

a) $0 \leq \tau_l(k) \leq k$, $k = 1,2,...$, c'est à dire que les valeurs du vecteur itéré utilisées à l'itération $k+1$ proviennent aux maximum de l'itération k,

b) $lim_{k \to \infty} \tau_l(k) = +\infty$, c'est à dire que les valeurs trop anciennes sont écartées au fur et à mesure que les calculs progressent.

c) $\forall l \in \{1,...,\alpha\}$, l'ensemble $\{k \in \mathbb{N} | l \in s(k)\}$ est infini, signifie qu'aucun bloc de composante ou aucune composante n'est abandonnée définitivement lors du processus itératif ; on doit réactualiser une infinité de fois chaque composante ou bloc de composantes.

Remarque: Le modèle (1.1) décrit des méthodes itératives par bloc. Dans le cas où $\alpha = n$, on trouve le cas particulier des méthodes par point.

1.1.3 Exemple et interprétation

En guise d'illustration nous reprenons avec plus de détail l'exemple simple (cf. [BT89], p. 429) d'une application de point fixé F de $\mathbb{R}^2$ dans $\mathbb{R}^2$ définie par :

$$\begin{cases} x_1 = F_1(x) = 0.5x_1 + 0.5x_2 \\ x_2 = F_2(x) = 0.2x_1 + 0.7x_2 \end{cases}$$

La mise en œuvre asynchrone est effectuée par deux processeurs P_1 et P_2. Le processeur P_1 réactualise la composante x_1 et envoie sa valeur au processeur P_2 qui réactualise la composante x_2 et envoie sa valeur au processeur P_1. On note par x^1 une instance du vecteur x sur le processeur P_1 et par x^2 une instance du vecteur x sur le processeur P_2. On suppose qu'aux instants
* $t = 0$: le processeur P_1 réactualise x_1^1 et envoie sa valeur au processeur P_2. Cette valeur est reue par P_2 entre les instants $t = 1$ et $t = 2$. P_2 réactualise x_2^2 et envoie sa valeur à P_1. Cette valeur est reue entre les instants $t = 1$ et $t = 2$.
* $t = 1$: P_1 réactualise x_1^1 et envoie sa valeur à P_2. Cette valeur est reue entre les instants $t = 2$ et $t = 3$. P_2 ne réactualise pas x_2^2, mais x_1^2 change suite à la réception de sa nouvelle valeur sur l'intervalle de temps [1,2]; c'est pour cette raison que $x^2(1) \neq x^2(2)$ (cf. figure (1.2)).
* $t = 2$: P_2 réactualise x_2^2 et communique sa valeur à P_1. Cette valeur est reue entre les instants $t = 3$ et $t = 4$. P_1 ne réactualise pas x_1^1 et ne reoit aucune valeur du processeur P_2 entre les instants $t = 2$ et $t = 3$; c'est pour cette raison $x^1(2) = x^1(3)$ (cf. figure (1.2)).
* $t = 3$: P_1 reoit $x_2^2(1)$ entre $t = 1$ et $t = 2$, réactualise x_1^1 et communique sa valeur à P_2. Cette valeur est reue entre les instants $t = 4$ et $t = 5$. P_2 reoit $x_2^2(1)$, réactualise x_2^2 et communique sa valeur qui est reue entre les instants $t = 4$ et $t = 5$.
* $t = 4$: P_1 a déjà reu $x_2^2(3)$ et il ne fait plus de réactualisation.
* $t = 5$: P_1 a déjà reu $x_2^2(4)$ et P_2 a déjà reu $x_1^1(5)$.

Dans la formulation des algorithmes itératifs asynchrones (cf. équation (1.1)), le terme $k - \tau_l(k)$ représente le retard dans l'accès au l-ème sous-vecteur du vecteur itéré pour les calculs effectués à l'itération $k + 1$. Les retards permettent de représenter l'effet de deux types de phénomènes : la différence de durée des phases de calcul et les délais de communication.

La différence de durée des phases de calcul est due soit à la différence de puissance des processeurs soit à la différence des charges de calcul. Le déséquilibre des charges peut être déterministe comme par exemple dans le cas où le travail à effectuer par les divers processeurs est assigné d'une manière non équitable. Le déséquilibre peut aussi être non déterministe si le travail à effectuer par les différents processeurs varie d'une manière aléatoire d'une itération à l'autre. Ainsi, entre le commencement et la fin d'une réactualisation effectuée par un processeur, d'autres processeurs peuvent effectuer un nombre indéterminé de réactualisations.

Pour illustrer ce phénomène on peut considérer un exemple simple d'équations de point fixe $x^* = F(x^*)$, ou F est une application de $\mathbb{R}^2 \to \mathbb{R}^2$ et $x^* \in \mathbb{R}^2$. On considère que chaque processeur utilise pour ses réactualisations les valeurs des composantes disponibles en début

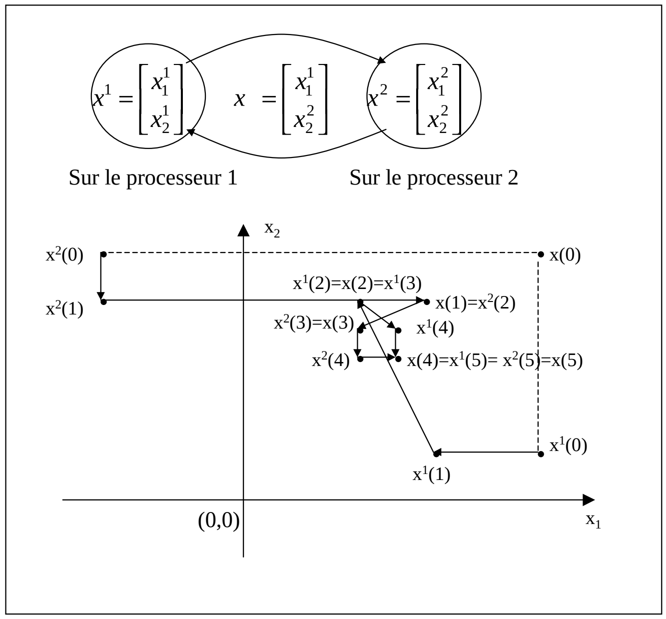

FIG. 1.2 – *graphe d'évolution des composantes*

de calcul. Un exemple de déroulement des calculs est donné par la figure 1.3 en supposant que les communications sont instantanées. On remarque que $x_1(k) = F_1(x_1(k-1), x_2(k-6))$.

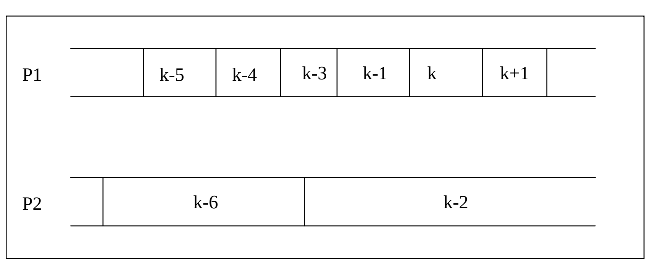

FIG. 1.3 – *Effet des différences de durée des phases de calcul*

Les délais de communication résultent essentiellement de l'architecture étudiée. Dans le cas d'une architecture à mémoire distribuée et pour une granularité faible, les délais peuvent être particulièrement importants ; ainsi entre l'émission de la valeur réactualisée d'une composante et la réception de cette valeur, le processeur récepteur peut avoir réactualisé plusieurs fois une autre composante. Un exemple de déroulement des calculs est donné par la figure 1.4 où les extrémités des flèches représentent respectivement le début et la fin des communications. On remarque que $x_1(k) = F_1(x_1(k-2), x_2(k-3))$. Le processeur P_1 recevant la valeur $x_2(k-1)$ après l'instant $t(k)$ correspondant au commencement de la k-ème itération.

Remarque: Outre son caractère novateur, la formulation des algorithmes itératifs asynchrones est aussi très générale car les conditions (a), (b) et (c) sont très faibles. La formulation (1.1) des algorithmes itératifs asynchrones englobe tous les algorithmes classiques.

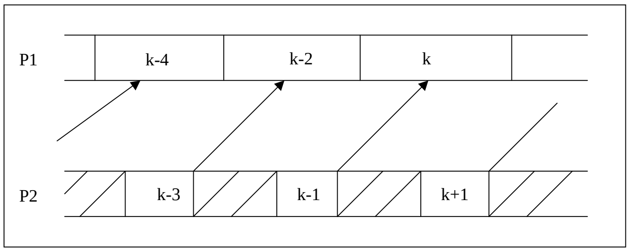

FIG. 1.4 – *Effet des délais de communication*

L'algorithme de Jacobi peut être représenté par exemple de la manière suivante avec le modèle (1.1) :

$$s(k) = \{1,...,\alpha\},\ k = 1,2,...$$

$$\tau_i(k) = k - 1, \forall i \in \{1,...,\alpha\},\ k = 1,2,...$$

Les autres algorithmes : Gauss-Seidel, "free-steering", itérations chaotiques série-parallèle (cf. [RCM75]), algorithmes asynchrones avec retards bornés peuvent être représentés à l'aide de cette formulation.

Dans le cas où les retards sont identiquement nuls, les itérations deviennent synchrones. Ainsi dans le cas d'une schéma itératif de type Jacobi, $x_i(k+1) = F_i(x_1(k),x_2(k))$, $\forall i \in \{1,2\}$. Un exemple type de déroulement des calculs est donné par la figure 1.5 qui montre une spécification des itérations synchrones sur deux processeurs. Initialement chaque processeur effectue une itération et envoie ses données à l'autre processeur ou les range en mémoire commune. Avant de faire une autre itération, chaque processeur doit attendre que les données relatives à l'autre processeur soient disponibles. Sur le diagramme temporel, les rectangles blancs numérotés représentent les phases de réactualisation, les rectangles hachurés les phases de communication ou d'attente et les flèches délimitent le début et la fin des communications.

1.2 Les itérations parallèles asynchrones avec communication flexible

Nous présentons dans ce paragraphe une nouvelle classe d'algorithmes itératifs asynchrones : les itérations asynchrones avec communication flexible qui ont été introduites dans [MES98]. Dans un premier temps nous donnons un exemple qui explique le concept des itérations asynchrones avec communication flexible, ensuite nous présentons les modèles mathématiques proposés dans [MES98], [ESMG96] (voir aussi [Elb98]).
Les itérations asynchrones avec communication flexible sont fortement liées aux concepts de $\mathcal{A}$-sous-application, de $\mathcal{A}$-sur application, de sous-application et de sur application.

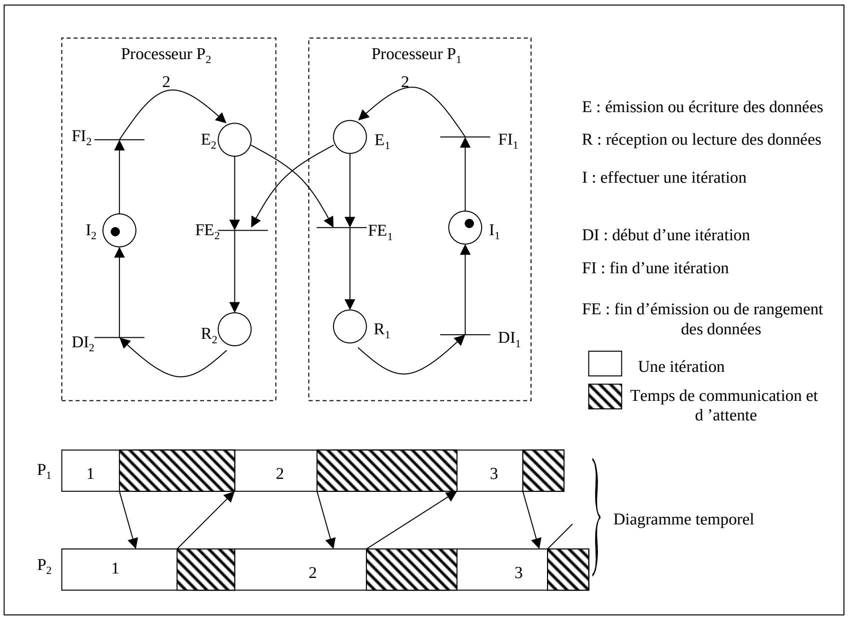

FIG. 1.5 – *Comportement d'un algorithme synchrone sur deux processeurs.*

1.2.1 Sous-applications et sur applications

Soit $n,\alpha \in \mathbb{N}$, $\alpha \leq n$ et considérons la décomposition de $\mathbb{R}^n$ en $\prod_{i=1}^{\alpha} \mathbb{R}^{n_i}$, $\sum_{i=1}^{\alpha} n_i = n$. Soit le vecteur $x \in \mathbb{R}^n$, on a alors la décomposition suivante de x :

$$x = \{x_1, \ldots, x_i, \ldots, x_\alpha\} \in \prod_{i=1}^{\alpha} \mathbb{R}^{n_i},$$

et la décomposition suivante de $\mathcal{A}$:

$$\mathcal{A}(x) = \{\mathcal{A}_1(x), \ldots, \mathcal{A}_i(x), \ldots, \mathcal{A}_\alpha(x)\} \in \prod_{i=1}^{\alpha} R^{n_i}.$$

Nous notons aussi par $(y_i; x)$ le vecteur $(x_1, \ldots, x_{i-1}, y_i, x_{i+1}, \ldots, x_\alpha)$.
Si $\mathcal{A} : \mathbb{R}^n \to \mathbb{R}^n$ est une M-fonction continue surjective (cf. Définition A.2.5 de l'annexe), alors pour tout $i \in \{1,...,\alpha\}$ et tout $x \in \prod_{i=1}^{\alpha} \mathbb{R}^{n_i}$, l'application de $\mathbb{R}^{n_i}$ dans $\mathbb{R}^{n_i}$ qui associe $\mathcal{A}_i(y_i; x)$ au sous-vecteur y_i est une M-fonction continue surjective sur $\mathbb{R}^{n_i}$ (cf. [Rhe70] et [OR70]). De plus pour tout $i \in \{1,...,\alpha\}$, le problème

$$\mathcal{A}_i(\hat{x}_i; x) = 0, \tag{1.3}$$

a une solution unique $\hat{x}_i$. L'application F de $\mathbb{R}^n$ dans $\mathbb{R}^n$ qui satisfait :

$$F(x) = \hat{x} = \{\hat{x}_1, \ldots, \hat{x}_i, \ldots, \hat{x}_\alpha\}, \tag{1.4}$$

où $\hat{x}_i$ est solution de (1.3) est bien définie. On a ainsi une formulation de type point fixe du problème $\mathcal{A}x^* = 0$ puisque $F(x^*) = x^* \Leftrightarrow \mathcal{A}(x^*) = 0$.

Définition 1.2.1 *(cf. [Mie86]) Soit $\mathcal{A}$ une M-fonction, alors l'application $\check{F}$ de composantes $\check{F}_i$, $i = 1,...,\alpha$, est une $\mathcal{A}$-sous-application associée à l'application de point fixe F si pour tout $i \in \{1,\ldots,\alpha\}$ et x élément du domaine de définition de $\check{F}_i : \{x \in R^n \mid \mathcal{A}_i(x) \leq 0\}$, on a $\check{F}_i(x) \geq x_i$, $\mathcal{A}_i(\check{F}_i(x); x) \leq 0$ et $\check{F}_i(x) \neq x_i$ si $F_i(x) \neq x_i$.*

Définition 1.2.2 *Soit $\mathcal{A}$ une M-fonction, alors l'application $\hat{F}$ de composantes $\hat{F}_i$, $i = 1,...,\alpha$, est une $\mathcal{A}$-sur application associée à l'application de point fixe F si pour tout $i \in \{1,\ldots,\alpha\}$ et x élément du domaine de définition de $\hat{F}_i$: $\{x \in R^n \mid \mathcal{A}_i(x) \geq 0\}$, on a $\hat{F}_i(x) \leq x_i$, $\mathcal{A}_i(\hat{F}_i(x); x) \geq 0$ et $\hat{F}_i(x) \neq x_i$ si $F_i(x) \neq x_i$.*

Définition 1.2.3 *(cf. [Mie86]) Soit $\mathcal{A}$ une M-fonction ,alors l'application $\check{F}$, de composantes $\check{F}_i$, $i = 1,...,\alpha$, est une sous-application associée à l'application de point fixe F si pour tout $i \in \{1,...,\alpha\}$ et x élément du domaine de définition de $\check{F}_i$ tel que $x_i \leq F_i(x)$, on a $\check{F}_i(x) \in \langle x_i,F_i(x)\rangle_i$ et $\check{F}_i(x) \neq x_i$ si $F_i(x) \neq x_i$.*

Définition 1.2.4 *Soit $\mathcal{A}$ une M-fonction, alors, l'application $\hat{F}$, de composantes $\hat{F}_i$, $i = 1,...,\alpha$, est une sur application associée à l'application de point fixe F si pour tout $i \in \{1,...,\alpha\}$ et x élément du domaine de définition de $\hat{F}_i$ tel que $x_i \geq F_i(x)$, on a $\hat{F}_i(x) \in \langle F_i(x),x_i\rangle_i$ et $\hat{F}_i(x) \neq x_i$ si $F_i(x) \neq x_i$.*

Les résultats suivants ont été établis dans [MES98].

Théorème 1.2.1 : *Si $\hat{F}$ est une $\mathcal{A}$-sur application associée à l'application de point fixe F, alors $\hat{F}$ est une sur application associée à l'application de point fixe F.*

Théorème 1.2.2 : *Si $\hat{F}$ est une $\mathcal{A}$-sous-application associée à l'application de point fixe F, alors $\hat{F}$ est une sous-application associée à l'application de point fixe F.*

Remarque : La réciproque des théorèmes 1.2.1 et 1.2.2 est en général fausse.

Les définitions suivantes sont relatives aux propriétés de continuité.

Définition 1.2.5 : *Une $\mathcal{A}$-sous-application ou une sous-application $\check{F}$ est continue pour l'ordre si*

$$x(j) \uparrow x^\star, j \to \infty \Rightarrow \check{F}_i(x(j)) \uparrow \check{F}_i(x^\star), j \to \infty, \text{ pour tout } i = 1,...,\alpha, \tag{1.5}$$

où la notation $x(j) \uparrow x^\star, j \to \infty$, signifie que $x(0) \leq x(1) \leq ... \leq x(j) \leq x(j+1) \leq ... \leq x^\star$ et $\lim_{j\to\infty} x(j) = x^\star$.

De manière analogue on dira qu'une $\mathcal{A}$-sur application ou une sur application $\hat{F}$ est continue pour l'ordre si $x(j) \downarrow x^\star, j \to \infty \Rightarrow \hat{F}_i(x(j)) \downarrow \hat{F}_i(x^\star), j \to \infty$, pour tout $i = 1,...,\alpha$, où la notation $x(j) \downarrow x^\star, j \to \infty$, signifie que $x(0) \geq x(1) \geq ... \geq x(j) \geq x(j+1) \geq ... \geq x^\star$ et $\lim_{j\to\infty} x(j) = x^\star$.

Définition 1.2.6 : *Une $\mathcal{A}$-sur application $\hat{F}$ est dite du premier type si pour tout $i = 1,...,\alpha$, et $x \in R^n$ tel que $\mathcal{A}_i(x) \geq 0$, on a : $\hat{F}'_i(x) \in \langle \hat{F}_i(x), x_i \rangle_i$, où $\hat{F}'$ est une $\mathcal{A}$-sur application continue pour l'ordre.*

Définition 1.2.7 : *Une sur application $\hat{F}$ est dite du premier type si pour tout $i = 1,...,\alpha$, et $x \in R^n$ tel que $x_i \geq F_i(x)$, on a : $\hat{F}'_i(x) \in \langle \hat{F}_i(x), x_i \rangle_i$, où $\hat{F}'$ est une sur application continue pour l'ordre.*

Définition 1.2.8 : *Une $\mathcal{A}$-sous-application $\check{F}$ est dite du premier type si pour tout $i = 1,...,\alpha$, et $x \in R^n$ tel que $\mathcal{A}_i(x) \leq 0$, on a : $\check{F}'_i(x) \in \langle x_i, \check{F}_i(x) \rangle_i$, où $\check{F}'$ est une $\mathcal{A}$-sous-application continue pour l'ordre.*

Définition 1.2.9 : *Une sous-application $\check{F}$ est dite du premier type si pour tout $i = 1,...,\alpha$, et $x \in R^n$ tel que $x_i \leq F_i(x)$, on a : $\check{F}'_i(x) \in \langle x_i, \check{F}_i(x) \rangle_i$, où $\check{F}'$ est une sous-application continue pour l'ordre.*

Définition 1.2.10 : *Une $\mathcal{A}$-sur application $\hat{F}$ est dite du second type si pour tout $i = 1,...,\alpha$, et $x \in R^n$ tel que $\mathcal{A}_i(y) \geq 0$, il existe $\delta > 0$ tel que*

$$\|x_i - \hat{F}_i(x)\|_i \geq \delta \|x_i - F_i(x)\|_i,$$

où $\|.\|_i$ est une norme definie sur R^{n_i}.

Définition 1.2.11 : *Une sur application $\hat{F}$ est dite du second type si pour tout $i = 1,...,\alpha$, et $x \in R^n$ tel que $x_i \geq F_i(x)$, il existe $\delta > 0$ tel que*

$$\|x_i - \hat{F}_i(x)\|_i \geq \delta \|x_i - F_i(x)\|_i.$$

Définition 1.2.12 : *Une $\mathcal{A}$-sous-application $\check{F}$ est dite du second type si pour tout $i = 1,...,\alpha$, et $x \in R^n$ tel que $\mathcal{A}_i(x) \leq 0$, il existe $\delta > 0$ tel que*

$$\|x_i - \hat{F}_i(x)\|_i \geq \delta \|x_i - F_i(x)\|_i.$$

Définition 1.2.13 : *Une sous-application $\check{F}$ est dite du second type si pour tout $i = 1,...,\alpha$, et $x \in R^n$ tel que $x_i \leq F_i(x)$, il existe $\delta > 0$ tel que*

$$\|x_i - \hat{F}_i(x)\|_i \geq \delta \|x_i - F_i(x)\|_i.$$

1.2.2 Un premier modèle d'itérations asynchrones avec communication flexible

Dans ce sous-paragraphe nous présentons un premier modèle d'itérations asynchrones avec communication flexible. Par la suite nous présentons un second modèle qui est différent et un peu plus général (cf. [MES98] et voir aussi [Elb98]).

Définition 1.2.14 *([MES98] et [Elb98]) Soit $\mathcal{A}$ une M-fonction continue surjective et soit F l'application de point fixe associée. Une itération asynchrone avec communication flexible relative à une sous-application $\check{F}$ du premier ou du second type associée à l'application de point fixe F sur $\Pi_{i=1}^{\alpha}\mathbb{R}^{n_i}$ et au point initial $x(0)$ satisfaisant $x(0) \leq F(x(0))$ est une suite $\{x(k)\}$ de vecteurs telle que pour $i = 1,...,\alpha$:*

$$\begin{cases} x_i(k) = \check{F}_i(\dot{x}(k)), \text{ si } i \in s(k), \\ x_i(k) = x_i(k-1), \text{ si } i \notin s(k), \end{cases} \tag{1.6}$$

Le vecteur $\dot{x}(k)$ est implicitement défini par :

$$\dot{x}_l(k) \in \langle \max\{x_l(\tau_l(k)),\dot{x}_l(q)\},x_l(k-1)\rangle_l,\ l = 1,...,\alpha, \tag{1.7}$$

$\langle,\rangle_l$ étant un segment d'ordre dans E_l et $q = \max_{p\in K_{s(k)}(k)} p$, $\{K_i(k)\}$ satisfaisant pour $i \in \{1,...,\alpha\}$ et $k = 1,2,...$, $K_i(k) = \{p \in \mathbb{N} \mid i \in s(p), 0 \leq p < k\}$. On pose $\dot{x}(0) = x(0)$ et $q = 0$ si $K_{s(k)}(k) = \emptyset$ et $\{(\tau_1(k),...,\tau_\alpha(k))|k = 1,2,...\}$ est une suite d'éléments de $\mathbb{N}^\alpha$ satisfaisant pour $i = 1,...,\alpha$:
a) i apparaît un nombre infini de fois dans $s = \{s(k)|k = 1,2,...\}$ qui est une suite de singletons de $\{1,...,\alpha\}$,
b) $0 \leq \tau_i(k) \leq k-1, k = 1,2,...$
c) $\tau_i(k) = k-1$, si $i \in s(k)$;
d) $lim_{k\to\infty}\tau_i(k) = +\infty$.

De manière analogue, une itération asynchrone avec communication flexible relative à une sur application $\hat{F}$ et au point initial $x(0)$ satisfaisant $x(0) \geq F(x(0))$ peut être définie en renversant les inégalités. On définit de manière analogue les itérations asynchrones avec communication flexible associés à une $\mathcal{A}$-sous-application ou une $\mathcal{A}$-sur application (cf. [MES98] et [Elb98]).
Selon le modèle (1.6-1.7), le sous-vecteur utilisé pour calculer $x_i(k)$ prend sa valeur dans le segment conique $\langle \max\{x_l(\tau_l(k)),\dot{x}_l(q)\},x_l(k-1)\rangle$, où $\dot{x}_l(q)$ représente la valeur du sous-vecteur x_l utilisée lors de la dernière réactualisation de x_i, le terme $x_l(\tau_l(k))$ traduit le comportement non déterministe de l'algorithme itératif asynchrone et $x_l(k-1)$ est la valeur de x_l à l'itération $(k-1)$. L'introduction des intervalles d'ordre nous permet de prendre en compte le cas où $\dot{x}_l(k)$ n'est pas nécessairement issu d'une nouvelle réactualisation du sous-vecteur x_l produite par l'application $\check{F}$ et peut correspondre à la valeur courante de x_l comme par exemple celle qui pourrait être produite par quelques pas du processus itératif

qui engendre $\check{F}$. La valeur $\dot{x}_l(k)$ peut donc correspondre à un nouvel itéré ou à un itéré partiel issu d'une réactualisation en cours et qui n'est donc pas étiqueté par un numéro d'itération particulier (cf. [Elb98]).

1.2.3 Un second modèle d'itérations asynchrones avec communication flexible

Le modèle suivant a été introduit dans [ESMG96] et permet aussi de définir les itérations asynchrones avec communication flexible.

Définition 1.2.15 *Soit $\mathcal{A}$ une M-fonction continue surjective et soit F l'application de point fixe associée. Une itération asynchrone avec communication flexible relative à une sous-application $\check{F}$ du premier ou du second type associée à F sur $\Pi_{i=1}^{\alpha}\mathbb{R}^{n_i}$ et au point initial $x(0)$ satisfaisant $x(0) \leq F(x(0))$ est une suite $\{x(k)\}$ de vecteurs telle que pour $i = 1,...,\alpha$:*

$$\begin{cases} x_i(k) \in \langle \check{F}_i'(x_1(\tau_1(k)),...,x_\alpha(\tau_\alpha(k))), \check{F}_i(x_1(\tau_1(k)),...,x_\alpha(\tau_\alpha(k)))\rangle \; si \; i \in s(k), \\ x_i(k) = x_i(k-1), \; si \; i \notin s(k) \end{cases} \quad (1.8)$$

où $\check{F}'$ est une sous-application continue pour l'ordre telle que pour tout $i \in \{1,..,\alpha\}$ et $x \in \Pi_{i=1}^{\alpha}\mathbb{R}^{n_i}$ tel que $x_i \leq F_i(x)$ on a : $\check{F}_i'(x) \in \langle x_i, \check{F}_i(x)\rangle$. Nous avons toujours les conditions a), b), c) et d) du premier modèle.

De même, une itération asynchrone avec communication flexible peut être définie relativement à une sur application $\hat{F}$ associée à F sur $\Pi_{i=1}^{\alpha}\mathbb{R}^{n_i}$ et au point initial $x(0)$ satisfaisant $x(0) \geq F(x(0))$ et vérifiant:

$$\begin{cases} x_i(k) \in \langle \hat{F}_i(x_1(\tau_1(k)),...,x_\alpha(\tau_\alpha(k))), \hat{F}_i'(x_1(\tau_1(k)),...,x_\alpha(\tau_\alpha(k)))\rangle \text{ si } i \in s(k) \\ x_i(k) = x_i(k-1), \text{ si } i \notin s(k) \end{cases} \quad (1.9)$$

1.3 Quelques résultats de convergence

L'étude de la convergence des itérations asynchrones a été effectuée dans des contextes variés. Les premiers résultats, datent de 1969, ont été donnés par Chazan et Miranker (cf. [CM69]) pour la résolution d'un système d'équations linéaires du type $\mathcal{A}x^* = z$, où $\mathcal{A}$ est une H-matrice (cf. Définition A.1.6 de l'annexe). On notera que la condition de convergence est alors nécessaire et suffisante. D'autres résultats de convergence des itérations asynchrones ont été établis par Miellou et son équipe au Laboratoire de Calcul Scientifique de Besanon, à savoir : Bahi, Boulbrachêne, Comte, Cortey-Dumont, El Tarazi, Jacquemard, Laouar et Spiteri pour la résolution de problèmes écrits sous la forme d'opérateurs non linéaire de point fixe (cf. [Mie75a], [Mie75b], [Mie86], [Spi83], [ET81], [ET82] et [Jac77]) ainsi que d'opérateurs H-accrétifs (cf. [MiS85b] et [Spi84]). Bertsekas a établi un résultat

de convergence très général (cf. [Ber83] et [BT89]) qui donne un ensemble de conditions suffisantes assurant la convergence des itérations asynchrones pour des problèmes de point fixe. Ce théorème a été appliquée notamment dans [BeE87], et [Elb96] afin de démontrer respectivement la convergence d'algorithmes de relaxation asynchrones et de gradient asynchrones pour des problèmes d'optimisation non linéaire de type flot dans les réseaux. En ce qui concerne la résolution des systèmes d'équations non linéaires avec ordre partiel on peut citer les travaux d'El Baz (cf. [Elb90], [Elb94a]). Dans ce même contexte El Baz, Miellou et Spiteri ont établi des résultats de convergence des itérations asynchrones avec communication flexible (cf. [MES98] et [ESMG96]). Par la suite certains de ces résultats on été étendus à la résolution de problèmes aux limites en combinaison avec des méthodes de multidécomposition (cf. [ESM97]). Dans le cas des systèmes singuliers, nous pouvons citer les résultats de convergence des itérations asynchrones établis par Bahi (cf. [Bah98a], [Bah98b] [Bah99a] et [Bah99b])), par Lubachewski et Mitra (cf. [LuM86] et par El Baz (cf. [Elb94b]). D'autres résultats de convergences ont été établis par Bahi dans le cas des systèmes différentiels algébriques (cf. [BGM96], [Bah96] et [BRM97]). Les résultats de convergence établis par Frommer, Szyld et Schwandt (cf. [FS94], [Fro95], [FS97], [FSc97], [FSS97], [FS98], [FS00] et [Szy97]), par Bru (cf. [BMPS95]) et par Bai Zhong-Zhi (cf. [ZDE95]) concernent des méthodes de multisplitting. On se rapportera aussi à [Elb98] et [FS99] pour une étude synthétique sur ce sujet.
Dans ce paragraphe nous présentons quelques résultats utilisés dans les chapitre II.1, II.2 et II.3 concernant des applications de point fixe contractantes et monotones, les systèmes non linéaires avec ordre partiel et les systèmes linéaires singuliers.

1.3.1 Problèmes de point fixe

Soit F une application de $D(F) \subset \mathbb{R}^n = \sum_{i=1}^{\alpha} \mathbb{R}^{n_i}$ à valeurs dans $D(F)$, telle que $D(F) \neq \emptyset$. On considère le problème de point fixe suivant:

$$x^* = F(x^*) \tag{1.10}$$

1.3.1.1 Contraction

Soit ψ une norme vectorielle sur $\mathbb{R}^n$ dans $\mathbb{R}^{\alpha}_{+}$ qui fait correspondre à tout $x \in \mathbb{R}^n$, $\psi(x) = (||x_1||_1,...,||x_\alpha||_\alpha)$, avec $||.||_i$ est une norme de $\mathbb{R}^{n_i}$.

Théorème 1.3.1 : *(cf. [Mie75a]) Si $D(F) \neq \emptyset$, F admet un point fixe $x^* \in D(F)$, et F est contractante en x^* pour la norme vectorielle ψ (cf. Définition A.4.4 de l'annexe) alors l'algorithme itératif asynchrone défini par (1.1) appartient à $D(F)$ et converge vers x^*.*

Remarque : Ce résultat peut être étendu au cas général où $D(F) \subset E$ et E est un espace de Banach reflexif.

Théorème 1.3.2 : *(cf. [ET81] et [ET82]) Si $D(F) \neq \emptyset$, F admet un point fixe $x^* \in D(F)$, et F est contactante sur $D(F)$ pour la norme pondérée $\|,\|_T^w$. alors l'algorithme itératif asynchrone défini par (1.1) appartient à $D(F)$ et converge vers x^*.*

1.3.1.2 Ordre partiel

Théorème 1.3.3 : *(cf. [Mie75b]) Si chaque espace $\mathbb{R}^{n_i}$ est muni d'un ordre partiel (cf. Définition A.1.2 de l'annexe), $\mathbb{R}^n$ étant muni de l'ordre partiel noté $\leq$, soient y^0 et x^0 deux éléments de $\mathbb{R}^n$ tels que $x^0 \leq y^0$, définissant le segment d'ordre $< x^0, y^0 >$ (cf. Définition A.1.3 de l'annexe) si l'application de point fixe F de $\mathbb{R}^n$ dans E est croissante, continue, et y^0 et x^0 sont respectivement sur solution et sous-solution, alors l'algorithme itératif asynchrone défini par (1.1) et initialisé par y^0 ou x^0 converge de manière monotone vers un point fixe de F.*

Le résultat suivant combine les deux aspects contraction et ordre partiel:

Théorème 1.3.4 : *(cf. [Jac77]) Sous les hypothèses du Théorème 1.3.3 et si de plus T une matrice de type (α,α), à éléments non négatifs est une majorante de F pour l'ordre, c'est à dire si :*

$$x \leq y \Rightarrow F(y) - F(x) \leq T(y-x), \forall x,y \in E, \tag{1.11}$$

alors F est continue et l'algorithme itératif asynchrone défini par (1.1) et initialisé par y^0 ou x^0 converge de manière monotone vers un point fixe de F. Si de plus F est contractante (cf. Définition A.4.4 de l'annexe), alors le point fixe de F est unique.

1.3.2 Systèmes d'équations non linéaires avec ordre partiel

Soit le système d'équations non linéaire suivant :

$$\mathcal{A}x^* = \mathcal{B}, \tag{1.12}$$

où x^* et $\mathcal{B}$ sont deux vecteurs de $\mathbb{R}^n$. Le sous espace $\mathbb{R}^{n_i}$ est muni de l'ordre partiel naturel, ou encore ordre composante par composante, associé au cne $K_i = \mathbb{R}_+^{n_i}$ des vecteurs de composantes non négatives de $\mathbb{R}^{n_i}$.
Soit le vecteur $x = \{x_1, \ldots, x_i, \ldots, x_\alpha\} \in \prod_{i=1}^{\alpha} R^{n_i}$ et la décomposition de $\mathcal{A}$:

$$\mathcal{A}(x) = \{\mathcal{A}_1(x), \ldots, \mathcal{A}_i(x), \ldots, \mathcal{A}_\alpha(x)\} \in \prod_{i=1}^{\alpha} R^{n_i}.$$

On note par $(\hat{x}_i; x)$ le vecteur $\{x_1, \ldots, x_{i-1}, \hat{x}_i, x_{i+1}, \ldots, x_\alpha\}$ et on considère pour tout $i \in \{1,...,\alpha\}$, le problème : $\mathcal{A}_i(\hat{x}_i; x) = 0$, et l'application F de R^n dans R^n qui satisfait : $F(x) = \hat{x} = \{\hat{x}_1, \ldots, \hat{x}_i, \ldots, \hat{x}_\alpha\}$, où $\hat{x}_i$ est une solution de ce problème.

Théorème 1.3.5 : *(cf. [Elb94a]) Si $\mathcal{A}$ est continue, diagonale monotone croissante et hors diagonale monotone décroissante (cf. Définition A.2.1 de l'annexe) et s'il existe* *une $\mathcal{A}$-sur solution $y(0)$ et une $\mathcal{A}$-sous-solution $x(0)$ (cf. Définition A.2.10 et A.2.7 de l'annexe), alors l'application de point fixe par point F définie ci-dessus (pour laquelle on a : $\alpha = n$) est multivoque et pour tout $i \in \{1,...,n\}$ et $x \in D = \{x \in R^n \mid x^0 \leq x \leq y^0\}$, $F_i(x)$ est un intervalle non vide et compact, de plus les applications de point fixe $\overline{F}$ et $\underline{F}$ telles que*

$$\overline{F}_i(x) = \ max_{x \in F_i(x)} x \ \ et\ \underline{F}_i(x) = \ min_{x \in F_i(x)} x, \forall i \in \{1,...,n\}, \forall x \in D, \tag{1.13}$$

respectivement, sont bien définies et monotones croissantes sur D. Si de plus $\overline{F}$ et $\underline{F}$ sont continues, alors toute itération asynchrone par point définie par le modèle (1.1) et associée à $\overline{F}$ ou $\underline{F}$ converge de manière monotone à partir de y^0 ou x^0,respectivement.

Théorème 1.3.6 : *(cf. [Elb90]) Si $\mathcal{A}$ est continue, diagonale monotone strictement croissante et hors diagonale monotone décroissante (cf. Définition A.2.2 de l'annexe) et s'il existe une $\mathcal{A}$-sur solution $y(0)$ et une $\mathcal{A}$-sous-solution $x(0)$, alors l'application de point fixe par point F définie ci-dessus est univoque et tout algorithme asynchrone par point défini par le modèle (1.1) et associé à l'application F converge de manière monotone à partir d'une $\mathcal{A}$-sur solution ou d'une $\mathcal{A}$-sous-solution.*

Remarque: L'application F de point fixe associée à $\mathcal{A}$ est continue et monotone croissante sur un segment d'ordre délimité par la $\mathcal{A}$-sous-solution et la $\mathcal{A}$-sur solution (cf. [Elb90] et [Elb98]).

Théorème 1.3.7 : *(cf. [Elb90]) Si $\mathcal{A}$ est une M-fonction continue surjective (cf. Définition A.2.5 de l'annexe), alors toute itération asynchrone définie par le modèle (1.1) et associée à l'application de point fixe F définie ci-dessus converge vers l'unique solution du problème quel que soit le point initial.*

Théorème 1.3.8 : *(cf. [Elb94a]) Si $\mathcal{A}$ est continue au sens de Lipschitz, diagonale monotone croissante, hors diagonale monotone décroissante, et s'il existe une $\mathcal{A}$-sur solution $y(0)$ et une $\mathcal{A}$-sous-solution $x(0)$, alors toute itération asynchrone par point définie par le modèle (1.1) et associée à l'application de point fixe $F : R^n \to R^n$ telle que :*

$$F(x) = x - \frac{1}{\alpha}\mathcal{A}x, \forall x \in R^n, \tag{1.14}$$

où α est la constante de Lipschitz relative à $\mathcal{A}$, converge de manière monotone à partir de y^0 et x^0.

Théorème 1.3.9 : *(cf. [Mie86]) Si $\mathcal{A}$ est une M-fonction continue surjective (cf. Définition A.2.5 de l'annexe), alors toute itération asynchrone définie par le modèle (1.1) et relative à une $\mathcal{A}$-sur application ou une $\mathcal{A}$-sous-application associée à l'application de point fixe F converge de manière monotone vers la solution du problème à partir d'une $\mathcal{A}$-sur solution ou d'une $\mathcal{A}$-sous-solution, respectivement.*

Nous donnons maintenant un résultat de convergence pour les algorithmes itératifs asynchrones avec communication flexible définis au paragraphe 1.2.2.

Théorème 1.3.10 : *(cf. [MES98]) Si $\mathcal{A}$ est une M-fonction continue surjective, alors toute itération asynchrone avec communication flexible définie par le modèle (1.6) et relative à une $\mathcal{A}$-sur application ou une $\mathcal{A}$-sous-application associée à l'application de point fixe F converge de manière monotone vers la solution du problème à partir d'une $\mathcal{A}$-sur solution ou d'une $\mathcal{A}$-sous-solution, respectivement.*

Le résultat suivant donne des conditions plus faibles de convergence pour les algorithmes itératifs asynchrones avec communication flexible. Ce résultat a été appliqué en particulier à des problèmes d'optimisation non linéaire de type flot dans les réseaux. (cf. chapitre 3)

Théorème 1.3.11 : *(cf. [ESMG96]) Si $\mathcal{A}$ est continue, diagonale monotone croissante, hors diagonale monotone décroissante, et s'il existe une sur solution $y(0)$ et une sous-solution $x(0)$, alors toute itération asynchrone par point avec communication flexible définie par le modèle (1.8) et relative à une sur application ou une sous-application associée à l'application de point fixe $\overline{F}$ ou $\underline{F}$, respectivement, converge de manière monotone à partir de $y(0)$ ou de $x(0)$, respectivement.*

1.3.3 Systèmes singuliers

Nous présentons dans ce sous-paragraphe trois résultats de convergence des itérations asynchrones relatifs à un système singulier. Le premier est celui de Bahi (cf. [Bah98a], [Bah98b] et [Bah99a]) qui donne un résultat général de convergence pour les systèmes singuliers. Le deuxième est établis par Lubachevsky et Mitra (cf. [LuM86]) qui est relatif à un système markovien. Le dernier est aussi relatif à un système markovien (cf. [Elb94b] qui utilise une autre approche différente faisant intervenir le concept d'ordre partiel.

Nous considérons un système singulier de la forme :

$$Ax = b, \tag{1.15}$$

où $A \in \mathbb{R}^{n\times n}$ est une matrice singulière et b est un vecteur de $\mathbb{R}^n$.
Considérons une décomposition de A sous la forme $A = M - N$ où $M = diag(A_{11},...,A_{\alpha\alpha})$.
Outre les hypothèses de la définition des itérations asynchrones (1.1) nous effectuons les hypothèses suivantes:

Hypothèse 1.3.1 *Il existe une sous-suite $\{k_l\}_{l\in\mathbb{N}}$ telle que $\forall i \in \{1,...,\alpha\}$, $i \in s(k_l)$ et $\tau_i(k_l) = k_l$,*

Hypothèse 1.3.2 *Il existe $K \in \mathbb{N}_+^*$ tel que $\forall i \in \{1,...,\alpha\}$, $\forall k \in \mathbb{N}$, $k - K + 1 \leq \tau_i(k) \leq k$,*

Hypothèse 1.3.3 *On suppose que la matrice M est non singulière.*

Proposition 1.3.1 *(cf. [Bah98a], [Bah98b] et [Bah99a]) Sous les hypothèses 1.3.1, 1.3.2 et 1.3.3 et les hypothèses suivantes :*
a) il existe $\gamma \in \mathbb{R}^n$, $\gamma > 0$ (c'est-à-dire $\forall i \in \{1,...,n\}, \gamma_i > 0$), tel que $A\gamma \geq 0$,
b) la décomposition de A en $M - N$ est faiblement régulière,
alors la matrice $M^{-1}N$ est non expansive par rapport à la norme $||.||_{\infty,\gamma}$, où $\gamma = \{\gamma_1,...,\gamma_\alpha\}$. De plus si $M^{-1}N$ est paracontractante par rapport à la norme $||.||_{\infty,\gamma}$, alors tout algorithme asynchrone associé au problème de point fixe $x^* = M^{-1}Nx^* + M^{-1}b$ converge vers une solution du problème 1.15.

Ces résultats ont été appliqués à un problème de chaîne de Markov (cf. [Bah98a], [Bah98b] [Bah99a]).

Lubachevsky et Mitra (cf. [LuM86]) ont établi un résultat de convergence pour les itérations asynchrones appliquées à la résolution d'un système markovien de type $x^* = x^*P$, où P est une matrice stochastique nonnégative, irréductible et de rayon spectral unité et x^* est le vecteur des probabilités stationnaires des états du système. Le résultat a été établi en supposant les hypothèses suivantes :

Hypothèse 1.3.4 *il existe au moins un indice $\bar{i}$ tel que $0 \leq \bar{i} \leq \alpha - 1$,*

$$P_{\bar{i}\bar{i}} > 0,$$
$$\tau_{\bar{i},\bar{i}}(k) = 0, k \geq 0,$$
$$x_i(0) > 0,$$

Hypothèse 1.3.5 *il existe une valeur finie d qui borne le terme de retard, c'est à dire*

$$0 \leq \tau_{i,j}(k) \leq d < +\infty \, ; \, k \geq 0, \, 0 \leq i \leq \alpha - 1, \, 0 \leq j \leq \alpha - 1,$$

Hypothèse 1.3.6 *il existe un entier s, $s < +\infty$, tel que chaque composante est réactualisée au moins une fois toutes les s réactualisations.*

Théorème 1.3.12 *Supposons que les hypothèses 1.3.4, 1.3.5 et 1.3.6 sont vérifiées, alors (i) il existe une constante C qui dépend de $x(0)$ et de x^* telle que pour tout $m = 0,1,...$ et $mr \leq k \leq (m+1)r$, où $r = 1 + d + (\alpha - 2)(s + d)$, on a:*

$$max_{0 \leq \tau \leq d}(max_{0 \leq j \leq \alpha-1}\{\frac{x_j(k-\tau)}{x_j^*}\}) - min_{0 \leq \tau \leq d}(min_{0 \leq j \leq \alpha-1}\{\frac{x_j(k-\tau)}{x_j^*}\}) \leq C(1-\sigma^r)^m,$$

où $\sigma = min^+_{ij}(\frac{P_{ij}x^*_{ij}}{x^*_{ij}})$, min^+ représente le minimum des éléments $\sigma \leq 1$.
(ii) Il existe une constante positive et finie c telle que l'itération asynchrone $\{x(k)\}$ satisfait :

$$\lim_{k \to +\infty} x(k) = cx^*$$

de plus

$$\sigma^r ||x(0)||_I \leq c \leq ||x(0)||.$$

La constante c dépend de $x(0)$, de la matrice P, de la séquence de réactualisation $\{s(k)\}$ et du terme de retard $\{\tau_{i,j}(k)\}$, $I = \{i | P_{ii} > 0$ et $\tau_{i,i}(k) = 0, \forall k\}$.
$x(k)$ converge proportionnellement vers x^* avec une constante de proportionnalité qui dépend du vecteur probabilité initial. Dans notre cas, $\sum_{l=0}^{n-1} x_l^* = 1$, donc la constante c peut être éliminée par une procédure de normalisation.

Les résultats de Lubachevsky et Mitra sont à rapprocher de ceux de Bertsekas et Tsitsiklis (cf. [BT89], p. 515) obtenus pour des algorithmes asynchrones avec retards bornés appelés aussi partiellement asynchrones par les auteurs. De manière plus générale, le résultat de convergence important de Bertsekas et Tsitsiklis (cf. [BT89], p. 488) donné pour les itérations partiellement asynchrones est un outil particulièrement intéressant pour prouver la convergence de ce type d'algorithmes avec des applications notamment à la recherche d'un point fixe d'une application non expansive.

Le résultat établi dans [Elb94b] suppose que l'une des composantes du vecteur probabilité x est fixée par exemple x_n et on élimine la dernière équation du système $Qx = 0$. On normalisera le vecteur probabilité par la suite.
Soit $x(0)$ le vecteur probabilité ayant pour composantes

$$\{ x_i(0) = 0, \text{ pour tout } i \in \{1,...,n-1\}, \quad x_n(0) = 1,$$

et l'ensemble $P = \{x \in \mathbb{R}^n | x(0) \leq x \leq x^*\}$.
Considérons la solution du système $Qx = 0$ par des méthodes itératives et plus particulièrement par la méthode de relaxation et la méthode de Richardson. L'application de point fixe associée à la méthode de relaxation est $F : P \subset \mathbb{R}^{n-1} \to \mathbb{R}^{n-1}$, ayant pour composantes $F_i(x) = \dot{x}_i$ telles que

$$\sum_{j=1, j\neq i}^{n-1} q_{ij}x_j + q_{ii}\dot{x}_i + q_{in} = 0, \; i \in \{1,...,n-1\}.$$

L'application de point fixe associée à la méthode de Richardson est $F' : P \in \mathbb{R}^n \to \mathbb{R}^n$ ayant pour composantes

$$F'_i(x) = x_i - \alpha(\sum_{j=1}^{n-1} q_{ij}x_j + q_{in}), \; i = 1,...,n-1,$$

où α est une constante positive.

Théorème 1.3.13 *(cf. [Elb94b]) Les itérations asynchrones relatives à la méthode de relaxation et à la méthode de Richardson convergent d'une manière monotone vers x^* à partir* *de* $x(0)$.

Remarque : Ce résultat est le seul pour lequel la convergence des itérations asynchrones appliquées à la résolution des systèmes markoviens est abordé sous l'aspect ordre partiel. De plus ce résultat est le seul pour lequel on ne suppose pas que les retards sont bornés.

1.4 Conclusion

Nous avons présenté dans ce chapitre les différents modèles mathématiques et informatiques des algorithmes itératifs synchrones, asynchrones et asynchrones avec communication flexible. Ces modèles permettent de définir différents schémas itératifs parallèles. Dans le cas des itérations asynchrones, chaque processeur itère à son propre rythme et les itérations s'enchaînent sans aucune attente. Les itérations asynchrones avec communication flexible permettent de communiquer des itérés partiels en cours de réactualisation qui ne sont pas forcément étiquetés par un numéro d'itération. nous avons présenté aussi quelques résultats de convergence des algorithmes itératifs asynchrones et asynchrones avec communication flexible. La mise en œuvre des algorithmes itératifs synchrones, asynchrones et asynchrones avec communication flexible sur des machines parallèles dépend de son architecture et de l'environnement de programmation parallèle utilisés, ceci fera l'objet du second chapitre.

Chapitre 2

Mise en œuvre parallèle

2.1 Introduction

La diversité des architectures parallèles nous a conduit à trouver des techniques de mise en œuvre parallèle adéquates aux itérations parallèles synchrones, asynchrones et asynchrones avec communication flexible. De nombreuses mises en œuvre d'algorithmes itératifs parallèles synchrones et asynchrones ont été effectuées sur des architectures parallèles. Parmi les toutes premières mises en œuvre d'itérations asynchrones, on peut citer celles effectuées par Baudet (cf. [Bau78]) sur la machine à mémoire partagée CMM, par El Baz et al (cf. [Elb84], [GBEB82], [BACEB84], [Elb89], [Elb93] et [ESMG96]) sur un réseau de 6 micro-ordinateurs à base de processeurs M6809 de Motorola et sur la machine à base de transputers Tnode de Telmat, par Chajakis et Zenios (cf. [CZ91]), par Giraud et Spiteri(cf. [Gir91] et [GS92]) et par Bru, Migallón, Penadés et Szyld (cf. [BMPS95] sur la machine à mémoire partagée Allient FX/8 et sur la machine à mémoire partagée IBM 3090-400, par Guivarch [Gui97] sur la machine à mémoire distribuée IBM SP2, par Frommer, Schwandt et Szyld sur le CRAY T3D et le CRAY J90 (cf. [FSS97]) et par Rhofir, El Kyal et Bahi sur une station SGI Power Challenge R10000 à 8 processeurs (cf. [REB98]). Dans ce chapitre, nous présentons des techniques de mise en œuvre des algorithmes itératifs parallèles synchrones, asynchrones et asynchrones avec communication flexible sur une architecture à mémoire partagée telle qu'une machine multiprocesseur symétrique SMP, sur un supercalculateur à mémoire distribuée tel que le CRAY T3E et sur un réseau de machines tel qu'un réseau de SMP. Dans un premier temps nous classifions ces architectures par rapport à l'ensemble des architectures parallèles qui existent actuellement, en suite nous présentons pour chaque type d'architecture les bibliothèques de programmation parallèle et les techniques utilisées pour mettre en œuvre les différents types d'algorithmes itératifs parallèles.

2.2 Classification des architectures parallèles utilisées

Dans la littérature, plusieurs classifications d'architectures parallèles ont été proposées. La plus connue est celle de Flynn(cf. [Fly66]) qui est basée sur le type d'organisation des flots de données et des flots d'instructions. Cependant, d'un avis général, cette classification ne permet pas de tenir compte de beaucoup de facteurs tels que le mode de fonctionnement des processeurs, l'organisation de la mémoire, etc... (cf. [CT93]). Avec cette classification le processus essentiel dans un ordinateur est l'exécution d'une suite d'instructions sur un ensemble de données.

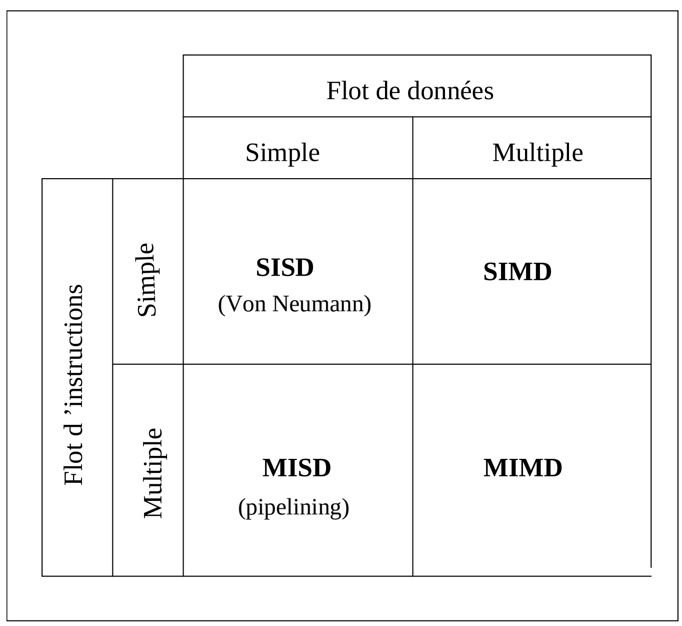

FIG. 2.1 – *Classification de Flynn des architectures parallèles*

Nous y trouvons en particulier l'architecture initiale des calculateurs qui est celle de Von Neumann. Celle-ci, constituée d'un seul processeur, ne permet que le traitement séquentiel des instructions et des données; on parle alors d'une architecture de type SISD "Single Instruction stream Single Data stream" qui est inadaptée au parallélisme. L 'architecture MISD "Multiple Instruction streams Single data stream", nommée aussi "pipline" est basée sur le principe de travail à la chaîne et d'une préparation au préalable des instructions avant leur exécution. L'architecture SIMD "Single Instruction stream Multiple Data streams" souvent nommée "array processors". Cette architecture permet à l'ensemble des processeurs une exécution synchrone de la même instruction sur d'importants flots de données différents. Cette synchronisation est effectuée par n séquenceurs (contrleur central) qui dirigent les données. La dernière architecture est l'architecture MIMD "Multiple Instruction streams Multiple Data streams". Cette architecture est basée sur un concept commun avec l'architecture SIMD, c'est-à-dire l'exploitation de plusieurs processeurs simul-

tanément, mais chaque processeur élémentaire exécute un ensemble distinct d'instructions sur ses propres données. Il existe dans ce cas un système d'interconnexion reliant les processeurs et les mémoires.
Les architectures parallèles que nous avons utilisées appartiennent à la classe des architectures MIMD. Dans un premier temps nous avons utilisé une architecture MIMD à mémoire partagée, ensuite une architecture MIMD à mémoire distribuée et finalement nous avons combiné ces deux types d'architecture sur un réseau de machines multiprocesseurs.

2.2.1 Architecture parallèle à mémoire partagée

Dans une architecture parallèle à mémoire partagée "Shared Memory MIMD", les processeurs P_1,P_2,...,P_p se partagent entre eux une mémoire unique $M = M_1 \cup M_2 ... \cup M_m$. Le système d'interconnexion assure tous les accès à la mémoire, aux unités d'entrée-sortie et les échanges de données entre les processeurs. Les processeurs ont chacun une unité de contrle (UC) et accèdent tous à une même zone mémoire (cf. Figure 2.2). La machine possédant ce type de fonctionnement est nommée machine multiprocesseur symétrique ou SMP. Une architecture à mémoire partagée est complexe dans sa réalisation, car tous les

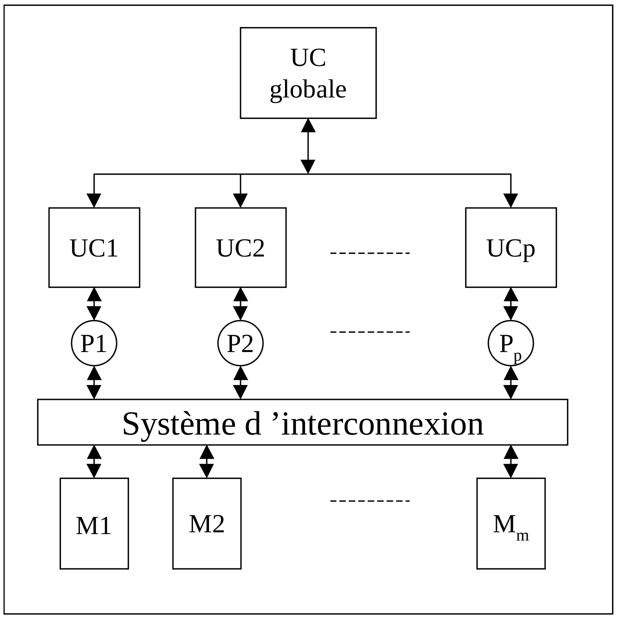

FIG. 2.2 – *architecture MIMD à mémoire partagée*

processeurs accèdent à une mémoire unique. En effet, il faut prendre en compte la simultanéité des accès mémoire ou des communications effectuées par les processeurs sans pour cela engendrer des conflits (mise à jour, duplication, cohérence des données). Généralement les architectures à mémoire partagée ne possèdent pas un grand nombre de processeurs, car au delà de quelques dizaines de processeurs, les performances du réseau d'interconnexion se dégradent (cf. [CT93]).

2.2.2 Architecture parallèle à mémoire distribuée

L'architecture multiprocesseur à mémoire distribuée offre plus de simplicité que les machines MIMD à mémoire partagée. Le système d'interconnexion n'assure que les échanges de messages entre les processeurs. Les topologies de connexion les plus connues sont les anneaux, les grilles toriques et les hypercubes. Il existe deux classifications possibles des machines MIMD à mémoire distribuée :
- Les machines MIMD à mémoire distribuée qui ne partagent rien, c'est-à-dire que chaque processeur possède sa propre mémoire, sa propre unité de stockage ainsi qu'une copie locale du système d'exploitation. Généralement ces machines sont massivement parallèles.
- Les machines MIMD à mémoire physiquement distribuée, mais logiquement partagée. Ce type d'architecture utilise des techniques qui permettent à chaque processeur de lire ou d'écrire dans la mémoire de ses voisins. En fait, le fonctionnement se déroule comme si les processeurs se partagent entre eux une mémoire commune virtuelle.

Le supercalculateur CRAY T3E que nous avons utilisé entre dans la première catégorie des machines MIMD à mémoire distribuée, mais avec une possibilité d'avoir un fonctionnement similaire à la deuxième catégorie et ceci en utilisant la bibliothèque d'échange des données SHMEM.

2.2.3 Réseau de machines

Les réseaux de machines ont été conus initialement pour partager des ressources matérielles (disques, etc...), ensuite ils ont évolué vers des systèmes et des applications répartis. En effet, avec l'apparition d'utilitaires orientés réseaux, il est possible de partager des ressources logicielles et d'autoriser la coopération entre des processus distants. Ce concept est appelé "client serveur", le client et le serveur communiquent via le réseau en échangeant des informations ou messages.

Les évolutions technologiques de ces dernières années font que les stations de travail ou les PC sont devenus de plus en plus puissants (processeurs plus rapides, multiprocesseur, etc...). Toutefois, leur puissance n'est jamais utilisée à pleine charge. Afin d'utiliser au maximum leur potentiel, nous pouvons les associer pour des calculs parallèles. Dans ce cas un réseau de machines pourra être vu comme une machine parallèle virtuelle (cf. Figure 2.3).

Un réseau peut regrouper plusieurs types d'architecture (MIMD à mémoire partagée, MIMD à mémoire distribuée, etc...). Nous avons utilisé un réseau de SMP qui peut être vu comme étant une machine parallèle virtuelle MIMD à mémoire distribuée dont chaque nœud est une SMP à mémoire partagée.

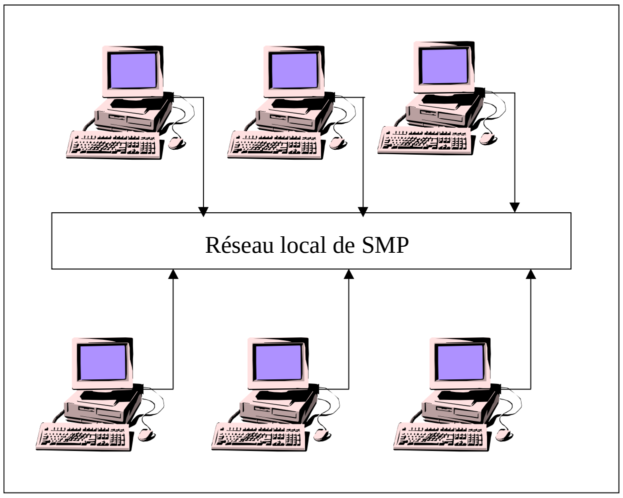

FIG. 2.3 – *machine parallèle virtuelle*

2.3 Mise en œuvre sur CRAY T3E

2.3.1 Description matérielle

La machine CRAY T3E est un supercalculateur parallèle à mémoire distribuée qui peut atteindre une puissance de crête de 1 Tera flops/s (10^{12} opérations en virgule flottante par seconde). Le nombre de processeurs élémentaires varie entre 32 et 2048 processeurs. Nous avons utilisé le supercalculateur CRAY T3E de l'Institut du Développement et des Ressources en Informatique Scientifique (IDRIS-CNRS). Cette machine est constituée de 256 processeurs Digital α EV5 ayant chacun une puissance de crête théorique de 600 Méga flops/s. Chaque processeur est constitué d'un microprocesseur, d'une mémoire locale et d'un support pour la circuiterie qui étend les fonctions d'adressage et de contrôle du microprocesseur. Il existe deux types de processeurs : les processeurs systèmes qui assurent la gestion globale du système via le système d'exploitation, et les processeurs de commande qui assurent l'exécution des programmes, l'édition de fichier, la compilation, etc...
Le CRAY T3E contient aussi une horloge centrale de fréquence 300MHz qui est distribuée sur tous les processeurs de calcul et les processeurs d'entrées/sorties. La topologie du réseau est celle d'un tore 3D qui constitue une grille tridimensionnelle connectant les nœuds dans les dimensions X,Y, Z (cf. Figure 2.4).
Chaque lien de communication connecte deux nœuds dans une dimension donnée. Ils sont constitués de deux canaux unidirectionnels par lesquels transitent les données, les informations de contrôle et les signaux d'accusé de réception. Le réseau d'interconnexion est à très faible latence (1μs) et à forte bande passante (480 Méga octet/s). L'échange de données ou messages entre les processeurs du CRAY T3E se fait par le biais des bibliothèques de communication ou bibliothèques de passage de message telles que PVM, MPI, SHMEM, etc.... Les fonctionnalités offertes par PVM sont toutes incluses dans MPI, c'est la raison

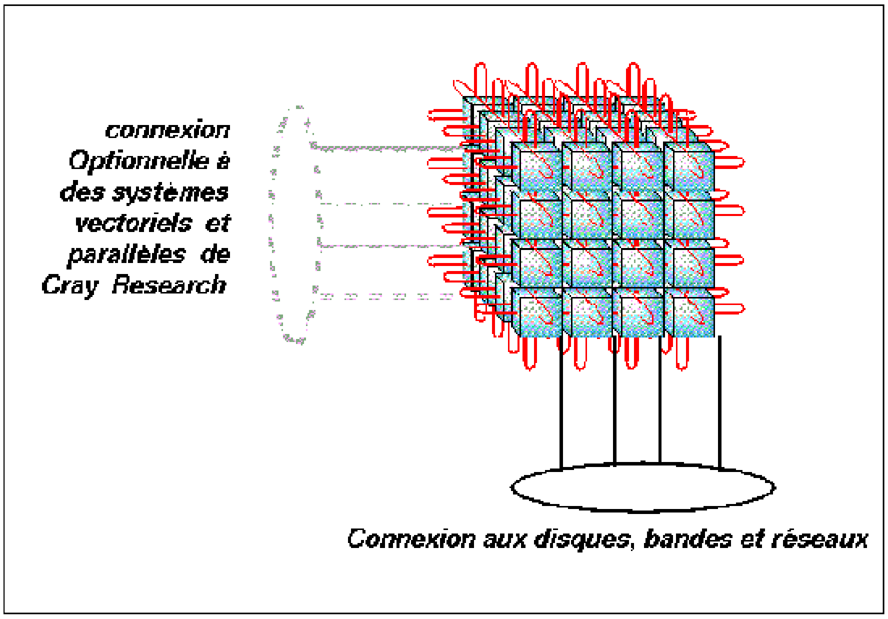

FIG. 2.4 – *architecture de la machine CRAY T3E*

pour laquelle nous avons choisi la bibliothèque MPI qui représente actuellement un standard pour les applications parallèles. Nous avons utilisé aussi la bibliothèque SHMEM de CRAY qui offre des fonctionnalités très intéressantes pour les mises en œuvre asynchrones et asynchrones avec communication flexible (cf. [EGJSM98], [JEG98], [JE99], [JE00a] et [JE00b]).

2.3.2 Mise en œuvre utilisant la bibliothèque MPI

MPI est une bibliothèque de passage de message sur architecture parallèle. Elle représente un standard pour les applications parallèles vu son degré de portabilité entre les différentes machines et architectures parallèles. Une application MPI est un ensemble de processus autonomes exécutant chacun leur propre code et communiquant via des appels à des sous-programmes de la bibliothèque MPI. La communication peut se faire, soit point à point entre deux processeurs, soit d'une manière collective en faisant plusieurs communications à partir d'une seule requête. Un message est constitué de paquets de données transitant du processeur émetteur au(x) processeur(s) récepteur(s). Un des points forts de cette bibliothèque est de pouvoir envoyer des données qui ne sont pas contigus en mémoire et ceci en définissant un nouveau type dérivé des types standards et des types qui sont définis par le concepteur. Ceci représente un intérêt particulier, car le concepteur pourra optimiser le nombre de communications entre les processeurs et par suite augmenter les performances de la mise en œuvre parallèle. MPI offre à l'utilisateur un ensemble de fonctions pour la communication qui peuvent être soit bloquantes, soit non bloquantes. Cette bibliothèque est particulièrement riche au niveau des communications de groupe par échange collectif d'un ensemble de données ou sélectif parmi un ensemble de données. MPI intègre le concept de communicateurs en associant un ensemble de tches à un contexte précis et qui donne à l'utilisateur une certaine souplesse de programmation en définissant les tches qui communiquent entre elles au sein d'un communicateur. En outre, un utilisateur MPI peut associer une certaine topologie graphique à l'ensemble des communicateurs afin d'affecter au mieux les tches aux processeurs. Nous avons utilisé la norme MPI-1 de MPI sur le CRAY T3E afin de mettre en œuvre en parallèle les itérations synchrones, asynchrones et asynchrones avec communication flexible.

a) Mise en œuvre synchrone

Dans le cas d'une mise en œuvre synchrone, le calcul de la valeur d'une composante à l'itération $(k+1)$ ne peut se faire que si tous les autres processeurs ont terminé le calcul des valeurs de ses composantes à l'itération (k) ; tout comme dans le cas séquentiel. La synchronisation peut se faire dans ce cas de deux manières : soit en échangeant les données en utilisant les fonctions non bloquantes MPI_Isend() et MPI_Irecv() pour envoyer et recevoir les données d'un processeur source à un processeur destination et la synchronisation se fait en appelant la primitive MPI_Barrier(), soit en synchronisant les processeurs au moment de la réception des données en utilisant la fonction bloquante MPI_Recv(). Cependant, en faisant une synchronisation par appel à une barrière présente un inconvénient, car sur une architecture à mémoire distribuée telle que le CRAY T3E, la réception non bloquante ne garantit pas l'arrivée des données avant la barrière de synchronisation et le schéma itératif peut commencer l'itération qui suit avec les anciennes valeurs du vecteur itéré. La mise en œuvre dans ce cas a un comportement synchrone, mais la synchronisation ne se fait pas comme définit au chapitre I.1. Nous indiquons dans ce qui suit la deuxième stratégie de mise en œuvre synchrone sur une architecture à mémoire distribuée. On note par calcul_des_données() la fonction qui réactualise les composantes (ou blocs de composantes) du vecteur itéré et par test_de_terminaison() la fonction qui teste la terminaison de l'algorithme itératif.

tant que non terminé **faire**
  calcul_des_données()
  MPI_Isend(données,...)
  MPI_Recv(données,...)
  test_de_terminaison()
fin tant que

b) Mise en œuvre asynchrone

Dans le cas d'une mise en œuvre asynchrone, chaque processeur itère à son propre rythme sans attente de réception de données provenant des autres processeurs. L'envoi et la réception entre les processeurs se font à l'aide des primitives non bloquantes d'envoi et de réception. Ces primitives peuvent être utilisées selon trois modes:
- un mode standard formé par les deux applications MPI_Isend() et MPI_Irecv qui utilisent un tampon pour stocker temporairement les données dans la mémoire cache du processeurs émetteur. Les messages sont stockés selon l'ordre FIFO (First In First Out).
- Un mode bufférisé qui consiste à définir la taille du tampon dans le cas où le message dépasse la taille par défaut du tampon. L'envoi des données se fait dans ce cas par le biais de la fonction MPI_Ibsend().
- Un mode sans tampon qui consiste à envoyer directement les données sans les stocker dans le tampon. L'envoi direct est assuré dans ce cas par la fonction MPI_Issend().
En plus des modes d'envoi et de réception, il existe deux modes de mise en œuvre asynchrone. Le premier consiste à envoyer à la fin d'une itération toutes les valeurs des composantes (ou blocs de composantes) frontières au(x) processeur(s) destination(s). Ce type de mise en œuvre est nommé asynchrone par sous-domaine. La bibliothèque MPI permet alors le cas où les adresses des composantes ne sont pas contigus en mémoire de définir un type dérivé construit à partir du type standard des composantes et leurs emplacement en mémoire. Le deuxième mode appelé par point consiste à envoyer la valeur d'une composante au(x) processeur(s) destination(s) dès qu'elle est réactualisée. Dans le cas de la mise en œuvre asynchrone par sous-domaine, nous utilisons généralement la fonction MPI_Isend() pour une taille qui ne dépasse pas l'espace par défaut du tampon. En ce qui concerne la mise en œuvre asynchrone par point, la fonction MPI_Issend() est la plus efficace. A chaque appel d'une primitive de communication (send() ou recv()), il y un temps additionnel au temps de communication appelé temps de latence. Dans le cas où le tampon de réception contient plus qu'une occurrence d'un message, il est coteux d'appeler plusieurs fois la fonction MPI_Irecv() pour récupérer la dernière occurrence. Afin de minimiser ce temps de latence additionnel nous avons utilisé une réception persistante qui consiste à exécuter une seule fois le processus de réception en utilisant la fonction MPI_Init_Recv() et la lecture du tampon se fait à l'aide de la fonction MPI_Start(). La détection d'un message dans le tampon se fait en appelant la fonction non bloquante MPI_Iprobe() qui donne des informations sur l'état du tampon. On note par $front$ l'ensemble des composantes (ou blocs de composantes) frontières et par P le numéro du processeur cible. L'algorithme suivant montre le mécanisme d'une réception persistante.

bool=FAUX
MPI_Init_Recv($front$,P)
MPI_Iprobe(bool,P)
tant que bool **faire**
  MPI_Start()
  MPI_Iprobe(bool,P)

fin tant que

c) Mise en œuvre asynchrone avec communication flexible

Suivant les modèles présentés au paragraphe 1.3.4, la communication flexible consiste à faire usage des itérés partiels des valeurs des composantes du vecteur itéré réactualisés au cours du calcul. L'envoi et la réception ne se font pas uniquement à la fin d'une itération comme dans le cas asynchrone standard, mais peuvent aussi être effectués par exemple après quelques pas fixés par une politique d'envoi et de réception des itérés partiels. Dans ce cas, nous noterons par S le sous-ensemble des numéros des pas intermédiaires auxquelles les itérés partiels sont communiqués. L'envoi et la réception des données se font alors à l'aide des fonctions non bloquantes MPI_Isend() et MPI_recv(). On note par Nproc le nombre de processeurs cibles et par l le numéro du processeur considéré. Afin de simplifier la présentation des itérations asynchrones avec communication flexible, les seuls arguments représentés pour les fonctions MPI_Isend() et MPI_recv() sont l'adresse des données à envoyer et le numéro du processeur cible. L'algorithme suivant met en œuvre les itérations asynchrones avec communication flexible à l'itération j en réactualisant la composante (ou le bloc de composantes) x_i:

$q = 0$
$x_i^q = x_i(j-1)$
pour $l = 1,Nproc$ **faire**
  reception_persistante($front$,l)
fin pour
tant que (recherche unidirectionnelle non terminée) **faire**
  $q = q + 1$
  si $q \in S$ **alors**
    pour $l = 1,Nproc$ **faire**
      reception_persistante($front$,l)
    fin pour
  fin si
  calcul de x_i^q
  si $q \in S$ **alors**
    pour $l = 1,Nproc$ **faire**
      MPI_Issend(x_i^q,l)
    fin pour
  fin si
fin tant que
$x_i(j) = x_i^q$
pour $l = 1,Nproc$ **faire**
  MPI_Issend($x_i(j)$,l)
fin pour

2.3.3 Mise en œuvre utilisant la bibliothèque SHMEM

La bibliothèque SHMEM est propre à l'architecture CRAY. Elle permet, comme MPI, de communiquer les données entre les processeurs. Cette bibliothèque gère les communications comme si toutes les mémoires locales étaient partagées entre tous les processeurs. En effet, l'envoi et la réception des données sont assimilés respectivement à une écriture et une lecture sur une mémoire locale d'un autre processeur. Les fonctions d'envoi et de réception font une copie mémoire à mémoire en faisant soit une copie dans la mémoire d'un processeur distant (shmem_put()), soit le chargement dans la mémoire locale d'un processeur à partir du processeur distant (shmem_get()). Le transfert d'une donnée de mémoire à mémoire se fait sans que le processeur récepteur en soit averti, contrairement aux fonction MPI_Isend() et MPI_Irecv() de MPI qui établissent un protocole de communications entre les deux processeurs afin d'échanger les données.

a) Mise en œuvre synchrone

La mise en œuvre synchrone se fait en échangeant les données entre les processeurs et en synchronisant ces derniers en appelant la primitive shmem_all_barrier(). Toutefois, si on utilise la fonction shmem_put()pour envoyer les données qui peuvent arriver aux processeurs destinations après la barrière de synchronisation et la mise en œuvre n'est plus synchrone en respectant la définition présentée au chapitre 1. La fonction shmem_get() est de nature bloquante, mais les données reues peuvent ne pas être réactualisé à la fin d'une itération et correspondent donc à des itérés partiels en cours de réactualisation. A fin d'être conforme aux itérations synchrones définies au chapitre II.1, nous nous retenons pas une mise en œuvre synchrone en utilisant la bibliothèque SHMEM.

b) Mise en œuvre asynchrone

Dans le cas d'une mise en œuvre asynchrone, deux stratégies ont été utilisées afin d'échanger les données entre les processeurs. La première consiste à envoyer après chaque réactualisation la valeur des composantes (ou blocs de composantes) frontières aux autres processeurs en utilisant la fonction shmem_put(). La deuxième consiste à recevoir, avant la réactualisation, les valeurs des composantes (ou blocs de composantes) frontières réactualisées par les autres processeurs en appelant la fonction shmem_get(). Ces deux stratégies de mise en œuvre asynchrone semblent plus naturelles que celle utilisant MPI_Isend() et MPI_Irecv(), car il n'y a pas de tampon à gérer. Cependant, dans le cas de l'utilisation de la fonction shmem_get(), l'accès direct en mémoire à la valeur courante du vecteur itéré peut ne pas correspondre forcément à la valeur produite à la fin d'une réactualisation. Dans ce cas on se ramène à une mise en œuvre asynchrone avec communication flexible.

c) Mise en œuvre asynchrone avec communication flexible

Les itérations asynchrones avec communication flexible peuvent être mises en œuvre, soit en utilisant la fonction shmem_put() pour envoyer les itérés partiels réactualisés au cours d'une recherche unidirectionnelle, soit en utilisant la fonction shmem_get() pour recevoir les itérés partiels réactualisés par les autres processeurs. L'envoi peut se faire selon une politique fixée (cf. paragraphe 2.3.2). Toutefois, on peu combiner les deux mises en œuvre asynchrone avec communication flexible en utilisant les deux fonctions shmem_put() et shmem_get() ensemble ; nous avons donc quatre cas possibles de mise en œuvre asynchrone avec communication flexible.

Algorithme 1 Utilisation de shmem_get sans politique fixée

$q = 0$
$x_i^q = x_i(j-1)$
pour $l = 1, Nproc$ **faire**
  shmem_get($front$,l)
fin pour
tant que (recherche unidirectionnelle non terminée) **faire**
  $q = q + 1$
  calcul de x_i^q
fin tant que
$x_i(j) = x_i^q$

2.4 Mise en œuvre sur une SMP

La mise en œuvre sur une SMP constitue une suite du travail de thèse de Didier Gazen (cf [Gaz98], [EGJSM98] et [JEG98]) qui a utilisé les threads POSIX pour mettre en œuvre les itérations synchrones, asynchrones et asynchrones avec communication flexible pour résoudre un problème de flot convexe dans les réseaux. Nous complétons cette étude et nous généralisons par la suite l'utilisation de cette technique sur un réseau de SMP. Nous présentons au début une description matérielle de l'architecture utilisée et nous rappelons par la suite les techniques utilisées (cf. [Gaz98], [EGJSM98], [JEG98], [JE00a] et [JE00b]) pour mettre en œuvre les itérations synchrones, asynchrones et asynchrones avec communication flexible.

2.4.1 Description matérielle

Nous avons utilisé un PC quadriprocesseur qui est une machine SMP Pentium Pro à 4x200MHZ disposant d'une mémoire cache primaire de 16Koctets et une mémoire cache secondaire de 512Koctets. Les 4 processeurs de la machine partagent une mémoire de 64Moctets.

Algorithme 2 Utilisation de shmem_get() avec politique fixée

$q = 0$
$x_i^q = x_i(j-1)$
pour $l = 1, Nproc$ **faire**
  shmem_get($front$,l)
fin pour
tant que (recherche unidirectionnelle non terminée) **faire**
  $q = q + 1$
  si $q \in S$ **alors**
    pour $l = 1, Nproc$ **faire**
      shmem_get($front$,l)
    fin pour
  fin si
  calcul de x_i^q
fin tant que
$x_i(j) = x_i^q$

Algorithme 3 Utilisation de shmem_put() avec politique fixée

$q = 0$
$x_i^q = x_i(j-1)$
tant que (recherche unidirectionnelle non terminée) **faire**
  $q = q + 1$
  calcul de x_i^q
  si $q \in S$ **alors**
    pour $l = 1, Nproc$ **faire**
      shmem_put(x_i^q,l)
    fin pour
  fin si
fin tant que
$x_i(j) = x_i^q$
pour $l = 1, Nproc$ **faire**
  shmem_put($x_i(j)$,l)
fin pour

Algorithme 4 Utilisation de shmem_put() et de shmem_get() avec une politique fixée

```
q = 0
x_i^q = x_i(j − 1)
pour l = 1,Nproc faire
  shmem_get(front,l)
fin pour
tant que (recherche unidirectionnelle non terminée) faire
  si q ∈ S alors
    pour l = 1,Nproc faire
      shmem_get(front,l)
    fin pour
  fin si
  q = q + 1
  calcul de x_i^q
  si q ∈ S alors
    pour l = 1,Nproc faire
      shmem_put(x_i^q,l)
    fin pour
  fin si
fin tant que
x_i(j) = x_i^q
pour l = 1,Nproc faire
  shmem_put(x_i(j),l)
fin pour
```

2.4.2 Mise en oeuvre en utilisant les processus legers

La mise en œuvre parallèle a été effectuée en utilisant les threads POSIX sous le système Linux. Elle repose sur la création des threads qui exécutent sur plusieurs processeurs des portions du code. Les threads sont crées au sein d'un processus Unix et utilisent les mêmes ressources que celui-ci. Ils sont appelés des processus légers, car ils ne nécessitent pas la création d'un contexte lourd comme pour un processus Unix. Ils consomment peu de ressources (registres, compteur ordinal et une petite pile d'exécution). Le temps mis par le système pour commuter des threads sur les processeurs est beaucoup plus faible que celui mis pour commuter des processus Unix (processus lourds). Les threads sont bien adaptés à une architecture à mémoire partagée vu qu'ils partagent toutes les ressources du processus Unix y compris la mémoire.

a) Mise en œuvre synchrone

Dans le cas d'une mise en œuvre synchrone, tous les threads doivent se synchroniser à la fin d'une itération. Une barrière de synchronisation a été développée en utilisant un verrou d'exclusion mutuelle et une variable condition. Cette barrière de synchronisation utilise un compteur qui est initialement égal à zéro. L'accès à ce compteur est protégé par le verrou d'exclusion mutuelle. Un thread qui a franchi la barrière appèle la fonction lock pour verrouiller l'accès au compteur, incrémente ce dernier et se met en attente en appelant la fonction pthread_cond_wait(). Cette fonction libère automatiquement l'accès aux autres threads à cette zone critique. Le dernier thread qui a franchi la barrière met à zéro le compteur et envoie par le biais de la fonction pthread_cond_broadcast() un signal à tous les threads qui sont en attente afin de les débloquer.

b) Mise en œuvre asynchrone

La mise en œuvre asynchrone est plus naturelle que celle synchrone vu qu'il n'y a pas de barrière de synchronisation. Chaque thread itère à son propre rythme et l'accès au vecteur itéré se fait d'une manière directe par une simple lecture de la mémoire commune. Cependant, dans le cas où la lecture d'un processeur se fait en plus d'un cycle d'horloge, les données communes (données frontières) doivent être protégées par un verrou d'exclusion mutuelle afin de conserver la cohérence des données. L'accès d'un thread à la valeur courante du vecteur itéré peut ne pas correspondre à la valeur réactualisée à la fin d'une itération (étiquetée par un numéro d'itération) et correspond en fait à un itéré partiel réactualisé au cours du calcul. Dans ce cas là, nous sommes dans un contexte de mise en œuvre asynchrone avec communication flexible. Afin de forcer la mise en œuvre asynchrone, les valeurs produites au cours du calcul doivent être transparentes vis à vis des autres threads ; ce qui revient à remplacer temporairement, au cours d'une réactualisation, la composante par une variable locale et la mise à jour du vecteur itéré ne se fait alors qu'à la fin de réactualisation. Nous ne considérerons dorénavant que la mise en œuvre asynchrone avec communication flexible vue qu'elle reproduit un schéma naturel. La figure

2.5 montre un modèle d'itérations synchrones et un modèle d'itérations asynchrones avec communication flexible sur une SMP.

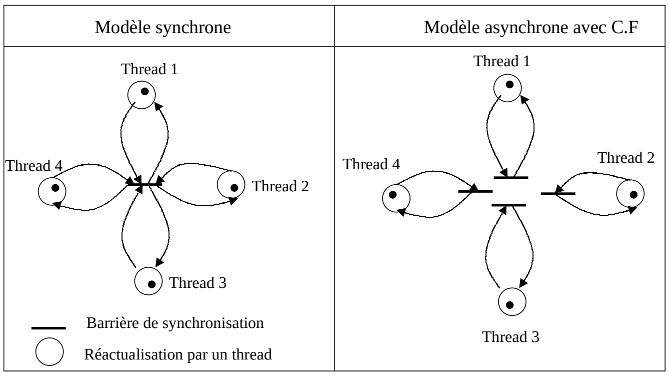

Fig. 2.5 – *modèles d'itérations synchrones et asynchrones avec communication flexible sur une SMP*

2.5 Mise en œuvre sur un réseau de SMP

2.5.1 Description matérielle

Nous avons utilisé deux types de réseau : un réseau Ethernet à 100Moctets/s et un réseau Myrinet à 1Goctets/s. Nous avons considéré une plate-forme homogène de 4 PC bi-processeurs symétriques Pentium II ayant une fréquence d'horloge de 233MHz et une taille mémoire de 128Moctets. Nous avons considéré aussi une plate-forme hétérogène formée de 2 PC bi-processeurs symétriques ayant une fréquence d'horloge de 333MHz et un PC Pentium Pro ayant une fréquence d'horloge de 200MHz. Le but de cette variété de réseaux et de plate-forme de machines est d'étudier l'influence des caractéristiques matérielles du réseau et des machines connectées à ce réseau sur la mise en œuvre des algorithmes itératifs parallèles synchrones, asynchrones et asynchrones avec communication flexible en fonction des bibliothèques utilisées.

2.5.2 Mise en œuvre avec passage de message et multithreading

2.5.2.1 Conception générale

Nous avons combiné les deux aspects "passage de message et multithread" pour mettre en œuvre efficacement les algorithmes itératifs parallèles sur un réseau de SMP. Le premier niveau de parallélisme utilise une bibliothèque de passage de message en considérant que chaque SMP est un nœud d'une architecture MIMD à mémoire distribuée. Le deuxième niveau de parallélisme utilise une bibliothèque de threads pour mettre en œuvre l'algorithme itératif sur l'ensemble des processeurs d'une architecture MIMD à mémoire partagée.

a) Mise en œuvre synchrone

Dans le cas d'une mise en œuvre synchrone, deux niveaux de synchronisation doivent être respectés. Le premier est entre les SMP connectées à travers le réseau et le deuxième est au sein des SMP en synchronisant tous les threads. La synchronisation des threads au sein d'une SMP se fait d'une manière identique à celle présentée au sous-paragraphe précédent, mais la synchronisation entre les SMP doit tout d'abord respecter la synchronisation des threads d'une SMP et la synchronisation sur l'envoi et la réception des données entre les SMP. Nous avons affecté toutes les communications à un seul thread, car le driver de communication refuse un accès concurrent par plusieurs threads au sein d'une même SMP. La synchronisation entre des threads situés sur des SMP différentes est effectuée en utilisant un envoi non bloquant et une réception bloquante. La figure 2.6 montre un modèle de mise en œuvre synchrone sur un réseau de SMP.

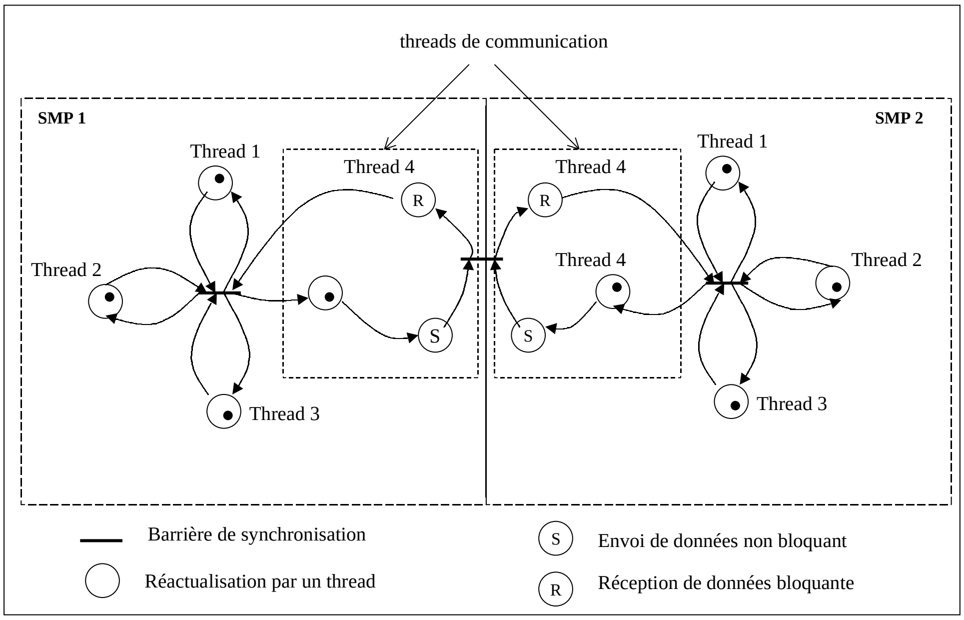

FIG. 2.6 – *modèle des itérations synchrones sur un réseau de SMP*

b) Mise en œuvre asynchrone

La mise en œuvre asynchrone est effectuée sur deux niveaux: le premier correspond à la mise en œuvre asynchrone sur une SMP et le deuxième entre les SMP du réseau.

Au sein d'une SMP chaque thread peut accéder à la valeur courante du vecteur itéré réactualisé par les autres threads. Cet accès est effectué avant chaque réactualisation par une simple lecture de la mémoire commune. On rappelle que dans le cas où la lecture ou l'écriture se fait en plus d'un cycle d'horloge de la SMP, un verrou d'exclusion mutuelle doit être mis en place afin de protéger cette zone mémoire contre une éventuelle opération d'écriture par un autre thread. On rejoint le cas d'une mise en œuvre asynchrone avec communication flexible sur une SMP. Dans le cas du second niveau qui correspond au réseau de SMP, les données frontières doivent être échangées par un thread communicant de chaque SMP. Chaque thread communicant envoie et reoit des données des autres threads communicant situés sur les autres SMP. Une donnée frontière peut être aussi réactualisée par un thread non communicant. L'accès à cette donnée par un thread communiquant pour une opération d'envoi peut ne pas correspondre au résultat final d'une réactualisation. Dans ce cas là, le thread communicant envoi un itéré partiel réactualisé au cours d'une recherche unidirectionnelle. On se ramène implicitement à un cas de mise en œuvre asynchrone avec communication flexible qui est naturel pour ce type d'architecture. Une politique d'envoi des itérés partiels par les threads communicant peut être aussi fixée, comme dans le cas d'une mise en œuvre asynchrone avec communication flexible sur une architecture MIMD à mémoire distribuée telle que le CRAY T3E. L'algorithme suivant met en œuvre sur un réseau de SMP les itérations asynchrones avec communication flexible.

$q = 0$
$x_i^q = x_i(j-1)$
si (thread communicant) **alors**
  pour $l = 1, Nproc$ **faire**
    reception_non_bloquante($front$,l)
  fin pour
fin si
tant que (recherche unidirectionnelle non terminée) **faire**
  $q = q + 1$
  si ($q \in S$ et thread communicant) **alors**
    pour $l = 1, Nproc$ **faire**
      reception_non_bloquante($front$,l)
    fin pour
  fin si
  calcul de x_i^q
  si ($q \in S$ et thread communicant) **alors**
    pour $l = 1, Nproc$ **faire**
      envoi_non_bloquant(x_i^q,l)
    fin pour
  fin si
fin tant que

$x_i(j) = x_i^q$
si (thread communicant) **alors**
  pour $l = 1,Nproc$ **faire**
    envoi_non_bloquant($x_i(j)$,l)
  fin pour
fin si

2.5.2.2 Cas de l'utilisation de MPICH

Nous avons utilisé la bibliothèque MPICH sur le réseau Ethernet à 100Moctets/s et MPICH-gmm sur le réseau Myrinet à 1Goctes/s. MPICH est une version de la norme MPI-1 sur un réseau de SMP. Nous avons toutes les fonctionnalités de MPI-1 avec la même syntaxe. Les machines sur lesquelles la mise en œuvre parallèle est effectuée doivent être spécifiées dans un fichier lu par MPICH au moment de l'exécution. Nous avons mesuré avec un programme test le temps de latence des primitives de communication de MPICH et de MPICH-gmm ainsi que la bande passante respectivement du réseau Ethernet et du réseau Myrinet. Avec MPICH sur Ethernet, le temps de latence est de $250\mu s$ et la bande passante réelle est de 64Mocts/s. Avec MPICH-gmm sur Myrinet, le temps de latence est de $27\mu s$ et la bande passante réelle est de 512Moctets/s.

2.5.2.3 Cas de l'utilisation de LANDA

LANDA (cf. [Mon95]) est un système multiserveur qui permet à des applications réparties de communiquer tout en surveillant le bon déroulement de celles-ci. Cet environnement est constitué de trois types de serveurs : les serveurs de tche qui sont situés sur les différentes machines composant la machine virtuelle et qui lancent et surveillent les tches, les serveurs d'application qui s'occupent de la gestion d'une application (événements, communication avec l'interface graphique, cohérence des tches,...) et le serveur de message qui assure la gestion des boîtes aux lettres des différentes tches de l'application. LANDA offre aux tches la possibilité de créer de nouvelles tches au cours de l'exécution d'une application. La tche créée de cette faon devient une tche à part entière de l'application et sera considérée de la même faon que celles lancées dès le début de l'application. Les fonctions de communication de LANDA ne manipulent que des tableaux. L'échange de données se fait à l'aide d'un mécanisme de boîte aux lettres géré par un serveur de message qui crée au moment de l'envoi d'un message, une boîte aux lettres au niveau de la machine cible pour stocker le message soit avant le message qui l'a précédé (Lsend : gestion par pile), soit après (Lsend():gestion par file), soit en le recouvrant (Lsende():gestion par écrasement et mise à jour). L'environnement LANDA a été développé au LAAS-CNRS et utilisé actuellement sur un réseau de PC de type Myrinet à 1Goctets/s.

2.5.2.4 Cas de l'utilisation d'ATHAPASCAN

L'environnement ATHAPASCAN est dédié à une mise en œuvre sur un réseau de SMP. ATHAPASCAN est développé dans le cadre du projet APACHE (cf. [BGP98]) au sein du laboratoire d' Informatique et Distribution (ID) de l'Institut d'Informatique et de Mathématiques Appliquées de Grenoble (IMAG). Nous avons utilisé la version 0 d'ATHAPASCAN qui permet de gérer deux niveaux de parallélisme : passage de message et multithreading. Chaque instance créée sur une SMP est un thread qui peut créer d'autres threads concurrents sur les processeurs de la SMP. La communication se fait à travers des ports qui donnent une abstraction des deux niveaux de parallélisme de telle sorte que l'architecture est vue comme étant homogène. Le gestion des threads se fait d'une manière transparente et la communication est soit locale (lecture d'une mémoire partagée) soit distante en communicant des données. ATHAPASCAN-0 est développé sur la couche MPI et les threads POSIX et offre des fonctionnalités de même sémantique mais de syntaxe légèrement différente afin d'offrir à l'utilisateur une certaine souplesse de programmation et une transparence au niveau du mécanisme de création et de gestion des threads. La mise en œuvre a été effectuée en utilisant ATHAPASCAN-0 sur le réseau Ethernet à 100Moctets/s.

2.6 Mesures de performance et paramètres importants

Sur chaque architecture nous mesurons le temps mis par la mise en œuvre séquentielle t_s et le temps mis par la mise en œuvre parallèle t_p sur p processeurs.
L'accélération $a(p)$ est définie par :

$$a(p) = \frac{t_s}{t_p}, \tag{2.1}$$

L'efficacité $e(p)$ est définie de la manière suivante :

$$e(p) = \frac{t_s}{t_p . p}, \tag{2.2}$$

Les performances des algorithmes itératifs parallèles dépendent grandement de nombreux facteurs liés aux applications et méthodes considérées, ainsi qu'aux machines et bibliothèques de communication utilisées. Parmi les facteurs les plus importants on citera : le temps de latence des communications, la bande passante et la granularité des tâches qui notée par g et définie de la manière suivante :

$$g = \frac{t_{cal}}{t_{com}}, \tag{2.3}$$

où t_{cal} est la durée moyenne d'une phase de calcul entre deux communications et t_{com} est la durée moyenne d'une communication. Un autre facteur important est l'équilibrage des tâches de calcul.

Dans les chapitres suivants nous détaillons l'influence de ces facteurs sur les performances des algorithmes itératifs parallèles.

2.7 Conclusion

Dans ce chapitre nous avons classifié les différentes architectures utilisées. Nous avons présenté pour chaque type d'architecture les techniques utilisées pour mettre en œuvre efficacement les itérations synchrones, asynchrones et asynchrones avec communication flexible en utilisant des bibliothèques et des environnements de programmation parallèle adéquats et disponibles sur cette architecture. Le but de cette étude est de définir une conception générale pour les différentes mises en œuvre qui utilisent les modèles présentés au chapitre I.1 et qui sera utilisée par la suite afin de résoudre certains problèmes numériques.

Deuxième partie

Applications et étude de performance

Chapitre 1

Résolution d'un problème de flot convexe dans les réseaux

1.1 Introduction

Nous étudions dans ce chapitre les performances des algorithmes itératifs synchrones, asynchrones et asynchrones avec communication flexible pour résoudre un problème de flot convexe dans les réseaux. Ce type de problème s'insère dans le cadre de l'optimisation convexe. Les problèmes de flot convexe dans les réseaux ont de nombreuses application; on peut citer par exemple les télécommunications et réseaux routiers, la finance et la distribution d'eau ou de gaz. La résolution de ce type de problème requiert des calculs intensifs (cf. [ZM86]) aussi sa mise en œuvre en parallèle semble-t-elle particulière intéressante (cf. [BCEZ95], [TB86] et [ZL88]).
Dans un premier temps, nous introduisons une formulation mathématique du problème sous sa forme primale; la théorie de la dualité permet de construire un problème dual convexe. Dans un second temps, nous proposons quelques méthodes de résolution séquentielles du dual. Ces méthodes se prêtent bien à une mise en œuvre parallèle dont nous étudions les performances dans le cas synchrone, asynchrone et asynchrone avec communication flexible. Nous comparons les différentes mises en œuvre sur diverses architectures parallèles en utilisant différents environnements et bibliothèques de programmation parallèle. Nous étudions les facteurs et les paramètres qui ont une influence sur les performances des mises en œuvre parallèles.

1.2 Les problèmes de flot à critère convexe

1.2.1 Formulation du problème

Soit G un graphe connexe orienté défini par un ensemble de sommets N et un ensemble d'arcs A, noté par G=(N,A). Les sommets de G sont numérotés de 1 à n où $n = |N|$ est le nombre total de sommets et $m = |A|$ est le nombre d'arcs du graphe G. Soit $c_{ij} : R \rightarrow$

$(-\infty, +\infty]$ la fonction de coût associée à l'arc $(i,j) \in A$, fonction du flux f_{ij} dans l'arc (i,j). Soit b_i l'entrée ou la sortie de produit au sommet $i \in N$, avec $\sum_{i \in N} b_i = 0$.

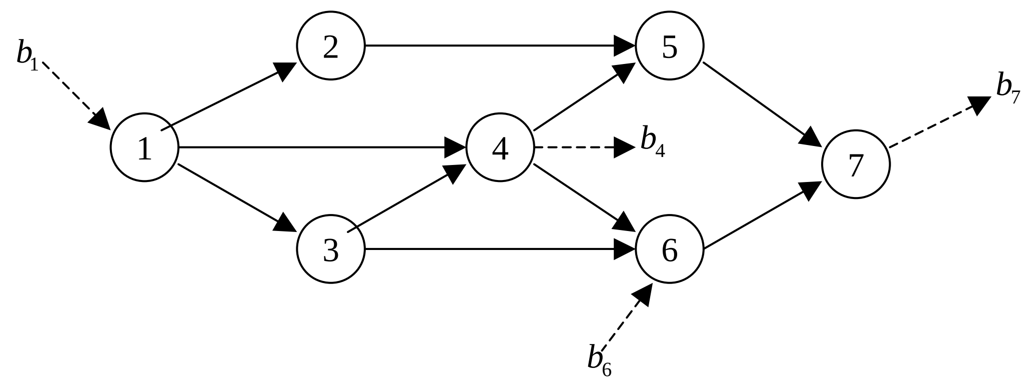

FIG. 1.1 – *Un graphe connexe orienté*

Le probleme consiste à déterminer le vecteur flot $f = \{f_{ij} | (i,j) \in A\}$ qui minimise le coût total sous la contrainte de conservation de flot en chaque sommet du graphe, soit :

$$\begin{cases} min \displaystyle\sum_{(i,j) \in A} c_{ij}(f_{ij}), \\ \text{sous} \displaystyle\sum_{(i,j) \in A} f_{ij} - \sum_{(m,i) \in A} f_{mi} = b_i, \forall i \in N. \end{cases} \tag{1.1}$$

Un vecteur flot qui satisfait la contrainte de conservation de flux est une solution admissible. Un vecteur flot admissible qui minimise le critère est une solution optimale du problème. Nous supposons que le problème (1.1) possède une solution admissible et nous considérons les hypothèses suivantes sur les fonctions de coût c_{ij}.

Hypothèse 1.2.1 *c_{ij} est semi-continue inférieure;*

Hypothèse 1.2.2 *c_{ij} est une fonction strictement convexe;*

Hypothèse 1.2.3 *la fonction convexe conjuguée de c_{ij} définie par*

$$c^*_{ij}(t_{ij}) = \sup_{f_{ij}} \{f_{ij}.t_{ij} - c_{ij}(f_{ij})\}, \tag{1.2}$$

est à valeur réelle, c'est à dire que $-\infty < c^*_{ij}(t_{ij}) < +\infty$ pour tout réel t_{ij}.

Les hypothèses 1.2.1 à 1.2.3 correspondent aux hypothèses générales effectuées dans [BeE87]. L'hypothèse 1.2.3 implique que $\lim_{|f_{ij}| \to \infty} c_{ij}(f_{ij}) = +\infty$. Par conséquent les ensembles circonscrits par les courbes de niveau de la fonction critère du problème (1.1) sont tous bornés (cf. [Roc70], chapitre 8). Il s'en suit que le problème (1.1) possède une solution optimale qui est unique en raison de l'hypothèse 1.2.2. Il résulte aussi de l'hypothèse 1.2.2 que les fonctions convexes conjuguées c_{ij}^* sont continûment différentiables et que leur gradient noté $\nabla c_{ij}^*(t_{ij})$, est l'unique f_{ij} qui atteint le supremum dans l'équation (1.2) (cf. [Roc70] p. 218 et p. 253 et [BeE87]). Notons enfin que ∇c_{ij}^*, qui est le gradient d'une fonction convexe différentiable est monotone non décroissant.

1.2.2 Une formulation duale

La formulation duale du problème (1.1) est obtenue en associant un multiplicateur de Lagrange à chaque contrainte de conservation de flux. Notons par f et p les vecteurs formés respectivement par les éléments f_{ij}, $(i,j) \in A$, et p_i, $i \in N$, où le vecteur p est appelé vecteur prix ou potentiel. Le Lagrangien s'écrit de la façon suivante :

$$\begin{aligned} L(f,p) &= \sum_{(i,j)\in A} c_{ij}(f_{ij}) + \sum_{i\in N} p_i \left(\sum_{k|(k,i)\in A} f_{ki} - \sum_{j|(i,j)\in A} f_{ij} + b_i \right) \\ &= \sum_{(i,j)\in A} (c_{ij}(f_{ij}) - (p_i - p_j) f_{ij}) + \sum_{i\in N} b_i p_i. \end{aligned} \tag{1.3}$$

La fonction duale q est obtenue en maximisant l'opposé du Lagrangien $-L(f,p)$ par rapport à f_{ij}.

$$\begin{aligned} q(p) &= min_f L(f,p) \\ &= -max_{(f_{ij}|(i,j)\in A)} \{ \sum_{(i,j)\in A} ((p_i - p_j) f_{ij} - c_{ij}(f_{ij})) \} + \sum_{i\in N} b_i p_i \\ &= - \sum_{(i,j)\in A} c_{ij}^*(p_i - p_j) + \sum_{i\in N} b_i p_i. \end{aligned} \tag{1.4}$$

Le dual du problème (1.1) est

$$max_{p\in\mathbb{R}^n} q(p) = -\min_{p\in\mathbb{R}^n} -q(p), \text{ sans contrainte sur le vecteur } p = \{p_i | i \in N\}, \text{ avec } -q(p) = \sum_{(i,j)\in A} c_{ij}^*(p_i - p_j) - \sum_{i\in N} b_i p_i.$$

La condition nécessaire et suffisante d'optimalité d'une paire (f,p) est donnée dans [Roc70] : un vecteur de flot admissible $f = \{f_{ij} | (i,j) \in A\}$ est optimal pour (1.1) et un vecteur prix $p = \{p_i | i \in N\}$ est optimal pour (1.5) si et seulement si, pour tout $(i,j) \in A, (p_i - p_j)$ est un sous-gradient de C_{ij} en f_{ij}. Une condition équivalente est : $f_{ij} = \nabla c_{ij}^*(p_i - p_j)$.

1.2.3 L'ensemble des solutions duales optimales.

Le problème dual sans contrainte consiste en la recherche d'un vecteur $p = (p_1, p_2, ..., p_n)$ où p_i représente le prix ou le potentiel d'un nœud i. L'existence d'une solution optimale pour le problème dual peut être garantie sous une hypothèse supplémentaire d'admissibilité régulière (cf. [Roc84] p.360 et p.329 et [BeE87]) :

Hypothèse 1.2.4 *Il existe un vecteur $x = \{x_{ij} | (i,j) \in A\}$ admissible vérifiant $c'_{ij-} < +\infty$ et $c'_{ij+}(x_{ij}) > -\infty$ pour tout $(i,j) \in A$, où c'_{ij-} et c'_{ij+} sont respectivement les dérivées à gauche et à droite de c_{ij}. Cette hypothèse supplémentaire implique que les composantes du vecteur optimal dual sont toutes bornées.*

Soit $g(p)$ le gradient de la fonction duale, donc les composantes $g_i(p)$ sont données par :

$$g_i(p) = \frac{\partial q(p)}{\partial p_i} = \sum_{(i,j)\in A} \nabla c^*_{ij}(p_i - p_j) - \sum_{(m,i)\in A} \nabla c^*_{mi}(p_m - p_i) - b_i, i \in N. \qquad (1.6)$$

La solution optimale du problème dual n'est pas unique puisque la fonction $g_i(p)$ reste inchangée si on ajoute la même constante à toutes les composantes du vecteur p. Ce degré de liberté peut être supprimé en imposant à une des composantes du vecteur p d'être égale à une constante. Par exemple, on prend $p_d = 0$, où d est un sommet destination. Considérons l'ensemble $P = \{p \in \mathbb{R}^n | p_d = 0\}$, et le problème dual réduit : $min_{p \in P} q(p)$. L'ensemble des solutions duales optimales réduit P^* est défini par : $P^* = \{p' \in P | min_{p\in P} q(p)\}$.

Hypothèse 1.2.5 *L'ensemble des solutions duales optimales réduit P^* est non vide et compact.*

Remarque : Dans [BeE87] les auteurs ont montré que l'hypothèse 1.2.5 n'est pas très restrictive. En particulier, elle est vérifiée dans le cas d'un problème où le flot de chaque arc n'est pas contraint en capacité. Dans le cas de flot contraint, le problème peut être converti en problème de flot non contraint en remplaant les contraintes de capacité par des fonctions de pénalité non différentiables (cf. [Ber82])

Il résulte de l'hypothèse 1.2.5 que le problème dual réduit admet une solution minimale et une solution maximale, c'est-à-dire qu'il existe $\underline{p}, \overline{p} \in P^*$ tel que $\underline{p} \leq p \leq \overline{p}$ pour tout $p \in P^*$, où $\underline{p} \leq p$ représente l'ordre partiel dans $\overline{R}^n$ (cf. [BeE87]). La fonction $g_i(p)$ est continue et monotone non décroissante puisque $g_i(p)$ est la dérivée partielle d'une fonction convexe continûment différentiable (cf. [TsB87]).
Dans ce qui suit nous envisageons quelques méthodes itératives pour la résolution du système d'équations non linaires $g(p) = 0$ avec $p \in P$.

1.3 Méthodes itératives de résolution.

Le problème dual ainsi que le problème dual réduit sont deux problèmes de minimisation sans contrainte et différentiables, donc on peut envisager leur résolution par des méthodes itératives de descente (cf. [Lue73]); parmi ces méthodes spécifiques à l'optimisation, nous avons utilisé la méthode de relaxation. L'interêt de cette méthode a été établi par Bertsekas (cf. [BCEZ95], [Ber95] et[BT89]).

1.3.1 Principe de la méthode de relaxation

Etant donné un vecteur prix p, un sommet $i \in N - \{d\}$ est sélectionné et son prix prend une valeur $\hat{p}_i$ qui minimise le coût dual par rapport au i-ème prix, les autres prix restent inchangés. Cette opération s'appelle une recherche unidirectionnelle suivant la i-ème direction. La méthode procède de manière cyclique en relaxant les prix de tous les sommets élément de $N - \{d\}$. La méthode de relaxation présente l'avantage de bien se prêter à une mise en œuvre parallèle ou distribuée pour laquelle chaque processeur réactualise un sous-ensemble de composantes du vecteur itéré. La propriété de décroissance du critère est conservée lorsqu'on réactualise simultanément un sous ensemble de composantes relatives à des sommets non adjacents.
Considérons maintenant l'application multivoque F_i, $i \in N - \{d\}$, qui associe à tout vecteur prix $p \in P$, l'ensemble des prix $\hat{p}_i$ qui minimisent le coût dual par rapport au i-ème prix, soit :

$$\begin{aligned} F_i(p) &= \{\hat{p}_i \in R | g_i(\hat{p}_i; p) = 0\} \\ &= \{\hat{p}_i \in R | \sum_{(i,j) \in A} \nabla c^*_{ij}(\hat{p}_i - p_j) - \sum_{(m,i) \in A} \nabla c^*_{mi}(p_m - \hat{p}_i) - b_i = 0\}. \end{aligned} \tag{1.7}$$

Une fonction convexe à valeur réelle pour laquelle un ensemble circonscrit par une courbe de niveau est compact est telle que tous les ensembles circonscrits par des courbes de niveau sont compacts (cf. [Roc70] p. 70); par conséquent, il résulte de l'hypothèse 1.2.5 que les ensembles $F_i(p)$, $p \in P$, $i \in N - \{d\}$, sont non vides et compact. Il s'en suit que les applications minimales et maximales de relaxation $\underline{F}$ et $\overline{F}$ de composantes respectives $\underline{F}_i(p) = min_{\hat{p}_i \in F_i(p)} \hat{p}_i$ et $\overline{F}_i(p) = max_{\hat{p}_i \in F_i(p)} \hat{p}_i$ $i \in N - \{d\}$ sont bien définies sur l'ensemble P (cf. [BeE87]).

Proposition 1.3.1 *(cf. [BeE87])sous les hypothèses 1.2.1 à 1.2.5, les applications* $\underline{F}$ *et* $\overline{F}$ *sont continues et monotones croissantes sur P.*

c'est-à-dire pour tout p,$p' \in P$, $i \in N - \{d\}$ on a:

$$\begin{aligned} \underline{F}_i(p) &\leq \underline{F}_i(p') \text{ si } p \leq p' \\ \overline{F}_i(p) &\leq \overline{F}_i(p') \text{ si } p \leq p' \end{aligned} \tag{1.8}$$

Proposition 1.3.2 *(cf. [BeE87]) Sous les hypothèses 1.2.1 à 1.2.5, les algorithmes de relaxation asynchrones associés respectivement à l'application minimale de relaxation* $\underline{F}$ *et l'application maximale de relaxation* $\overline{F}$ *convergent d'une façon monotone respectivement vers* $\underline{p}$ *et* $\overline{p}$ *à partir de points initiaux satisfaisant respectivement* $p(0) \leq \underline{p}$ *et* $p(0) \geq \overline{p}$.

1.3.2 Méthode de type gradient

Nous présentons maintenant une méthode de descente construite à partir de l'application du gradient. Les composantes F_i' de l'application de gradient F' sont définies par : $F_i'(p) = p_i - \frac{1}{\alpha} g_i(p)$ pour tout $i \in N - \{d\}$ et $p \in P$, où α est une constante positive. Nous introduisons maintenant l'hypothèse suivante :

Hypothèse 1.3.1 *c_{ij} est fortement convexe de module $\frac{1}{\beta}$.*

C'est à dire qu'il existe $\beta > 0$ tel que pour tout couple $(i,j) \in A$ et pour tout $(\eta,\xi),(\eta',\xi') \in \Gamma_{ij}^{-1}$ satisfaisant $\eta \leq \eta'$ on a $\xi - \xi' \leq \beta(\eta - \eta')$ où $\Gamma_{ij}^{-1} = \{(\eta,\xi) \in \mathbb{R}^2 | \xi = \nabla C_{ij}^*(\eta)\}$ est la courbe caractéristique réciproque associée à la fonction convexe conjuguée du coût, ainsi appelée courbe de coût marginal de c_{ij}^* (cf. [Roc84]).

Il a été démontré dans [Elb96] que sous les hypothèses 1.2.1 à 1.2.3 et 1.3.1, il existe une constante $\alpha = \beta max_{i \in N} a_i$ (où a_i est le degré du sommet $i \in N$) telle que pour tout $p,p' \in P$ satisfaisant $p' \leq p$, on a $g(p) - g(p') \leq \alpha(p - p')$. Il en résulte que l'application de gradient F' est monotone croissante sur P pour $\alpha = \beta max_{i \in N} a_i$ (l'application F' est également monotone croissante sur un sous-ensemble de P si c_{ij} est fortement convexe sur un domaine associé (cf. [Elb96])).

La méthode de type gradient consiste à itérer sur la i-ème composante du vecteur prix de la manière suivante : $p_i^q = p_i^{q-1} - \frac{1}{\alpha} g_i(p_i^{q-1},p)$, $i = 1,...,n$, $q \in \{1,2,...,q'\}$, où $p_i^0 = p_i$ et q' est le nombre d'itérations tel que $|g_i(p_i^{q'},p)| \leq \epsilon$, ϵ est la précision fixée pour la recherche unidirectionnelle.

Remarque : La méthode de gradient consiste à prendre systématiquement $q' = 1$.

Remarque : Nous rappelons qu'il a été démontré dans [ESMG96] que l'application de type gradient $\check{F}$ définie par $\check{F}_i(p) = p_i^{q'}$ pour tout $i \in N - \{d\}$ est une sous-application du premier type sur $P' = \{p \in P | p \leq \underline{p}\}$ et que $\hat{F}$ est une sur application du premier type sur $P'' = \{p \in P | p \geq \overline{p}\}$. En utilisant ces propriétés, la convergence monotone des itérations asynchrones ave communication flexible associées à l'application de type gradient a été démontrée dans [ESMG96] à partir de sous-solutions et de sur solutions.

1.3.3 Test de terminaison

Le test de terminaison est basé uniquement sur la valeur de $|g_d(p_1(\tau_1^d(k)),...,p_n(\tau_n^d(k)))|$. Il a été démontré dans [ESMG96] que ce test de terminaison peut détecter la terminaison globale des itérations asynchrones avec communication flexible.

Proposition 1.3.3 *(cf. [ESMG96]) On suppose que les hypothèses 1.2.1 à 1.2.5 sont verifiées, et que $\{p(k)\}$ est une séquence engendrée par un algorithme itératif asynchrone avec communication flexible associé à une sous-application m-continue sur P' et commenant par $p(0) \in P'$ tel que $p(0) \leq \underline{F}(p(0))$, où $P' = \{p \in P | p \leq \underline{p}\}$, alors $g_d(p(k)) \geq 0$ pour tout k.*

Proposition 1.3.4 *(cf. [ESMG96]) On suppose que les hypothèses 1.2.1 à 1.2.5 sont vérifiées, ϵ étant une constante positive et qu'il existe k tel que $g_d(p(k)) \leq \epsilon$, alors $g_d(p(k')) \leq \epsilon$ pour tout $k' \geq k$.*

Proposition 1.3.5 *(cf. [ESMG96]) On suppose que les hypothèses 1.2.1 à 1.2.5 sont vérifié, ϵ étant une constante positive et qu'il existe k tel que $|g_d(p_1(\tau_1^d(k)),\ldots,p_n(\tau_n^d(k)))| \leq \epsilon$, alors $\sum_{i \in N-d} |g_i(p(k'))| \leq \epsilon$ pour tout $k' \geq k$.*

Un résultat similaire (cf. [ESMG96]) a été démontré pour un algorithme asynchrone avec communication flexible associé à une sur application m-continue sur P'' et commençant par $p(0) \in P''$ tel que $p(0) \geq \overline{F}(p(0))$ où $P'' = \{p \in P | \overline{p} \leq p\}$ Ces résultats restent valables pour les itérations asynchrones.

1.4 Mise en œuvre séquentielle

1.4.1 Présentation de l'algorithme séquentiel

Nous présentons dans ce sous-paragraphe l'algorithme séquentiel associé à la méthode de type gradient. Cette méthode procède en relaxant de manière cyclique à chaque itération les prix des nœuds du réseau à l'exception du nœud destination noté d. L'algorithme ne s'arrête que lorsque le déficit (le gradient) du nœud destination est inférieur à une certaine valeur. L'algorithme séquentiel se présente de la manière suivante :

Algorithme 5 Séquentiel

tant que $g_d(p,p_d) \geq \epsilon_1$ **faire**
  pour $i = 1,\ i = d-1$ **faire**
    tant que $g_i(p,p_i) \geq \epsilon_2$ **faire**
      $p_i = p_i - \frac{1}{\alpha} g_i(p,p_i);$
    fin tant que
    $p_i = p_i - \frac{1}{\alpha} g_i(p,p_i)$
  fin pour
fin tant que

1.4.2 Problèmes traités

Nous avons traité trois types de problèmes de flot dans les réseaux : les problèmes de flot à coût quadratique, les problèmes de flot turbulent dans les réseaux hydrauliques et les problèmes de flot de communication. Les fonctions de coût ainsi que leur gradient conjugué sont notés respectivement par $c_{ij}(f_{ij})$ et $\nabla c^*_{ij}(p_i - p_j)$ et qui sont explicités dans le tableau suivant :

	flots quadratiques	flots turbulents	flots de communication
c_{ij}	f_{ij}^2	$\|f_{ij}\|^{\frac{(1+b)}{b}}$	$(\frac{1}{a_{ij}-f_{ij}} + b_{ij}).f_{ij}$ si $0 \leq f_{ij} < a_{ij}$ $+\infty$ si $f_{ij} < 0$ ou $f_{ij} \geq a_{ij}$
$\nabla c^*_{ij}(p_i - p_j)$	$\frac{1}{2}(p_i - p_j)$	$sig(p_i - p_j)\|p_i - p_j\|^{1.85}$ (cas où $b = 1.85$)	$1 - (\frac{1}{p_i - p_j - \frac{1}{2}})^{\frac{1}{2}}$ si $p_i - p_j \geq \frac{3}{2}$ 0 si $p_i - p_j \leq \frac{3}{2}$ (cas où $a_{ij} = 1$ et $b_{ij} = 0.5$)

Les fonctions de coût des trois problèmes présentés ci-dessus vérifient les hypothèses 1.2.1 à 1.2.3 et sur un P. L'hypothèse 1.3.1 étant vérifiée éventuellement sur un sous-ensemble de P. β est déterminé en calculant la plus grande pente que peut prendre la courbe caractéristique réciproque sur un certain domaine. Nous avons pris $\beta = 0.5$ pour les problèmes de flot quadratique et de flot de communication et $\beta = 0.74$ pour les problèmes de flot turbulent.

1.4.3 Comparaison entre la méthode de type gradient et la méthode du gradient conjugué

Afin de montrer l'interêt des algorithmes de type gradient pour les problèmes de flot dans les réseaux, nous allons comparer l'algorithme de type gradient avec l'algorithme du gradient conjugué décrit en Annexe (cf. A.5). Le tableau 1.1 montre les temps de calcul en seconde de la méthode de type gradient et de la méthode du gradient conjugué pour les différents problèmes présentés ci-dessus. Les expérimentations ont été effectuées sur des réseaux de dimension 16x3, 24x3, 32x3 et 48x3, où la notation lxc signifie que le réseau est une grille bidimensionnelle formée de l lignes et de c colonnes.

D'après le tableau 1.1, la méthode de type gradient est beaucoup plus rapide que la méthode de gradient conjugué dans le cas d'un flot quadratique ou d'un flot turbulent et légèrement plus lente dans le cas d'un flot de communication présentant une forte non linéarité par rapport aux deux premiers cas. Ces résultats montrent que la mise en œuvre de la méthode de type gradient est très intéressante pour la résolution des problèmes de flot convexes dans les réseaux ; une raison étant la propriété de séparabilité du critère. Nous verrons par la suite que cette méthode se prête bien à une mise en œuvre parallèle.

1.5 Etude de performance

1.5.1 Décomposition du problème

Nous décomposons le graphe $G = (N,A)$ en α sous-graphes, où α correspond au nombre de processeurs. Chaque sous-graphe est attribué de manière statique à un processeur qui est

alors chargé de mettre à jour le sous-ensemble des composantes du vecteur prix associées aux sommets du sous-graphe. L'ensemble des sommets est réparti le plus équitablement possible sur les différents processeurs de calcul. Cette répartition se fait en divisant le nombre de sommets par le nombre de processeurs, le reste de la division sera aussi réparti sur les différents processeurs. Soit N_k le sous-ensemble des sommets affectés au processeur P_k et Ω_i l'ensemble des sommets adjacents au sommet i. On appelle sommet interne tout sommet $i \in N_k$ tel que $\forall j \in \Omega_i$, $j \in N_k$ et sommet frontière tout $i \in N_k$ tel que $\exists j \in \Omega_i$ qui satisfait $j \notin N_k$. Tout sommet frontière affecté à un processeur est relié à au moins un sommet frontière d'un autre processeur. Les processeurs vont donc devoir échanger ou lire directement en mémoire les prix des sommets frontières. En respectant cette répartition, il existe plusieurs manières d'affecter les sommets aux différents processeurs. La figure (1.2) montre deux affectations possibles d'un réseau maillé formé de 32 sommets (8 lignes x 4 colonnes) sur 4 processeurs. Selon le choix de l'affectation, le nombre de sommets internes et frontières varie. Nous retenons dans cette étude de performance la première décomposition pour ce type de réseau.
Dans le cas où les processeurs communiquent entre eux toutes les valeurs des composantes frontières à la fin d'une itération, la mise en œuvre parallèle est appelée par sous-domaines et dans le cas où les processeurs communiquent uniquement la valeur d'une composante réactualisée, la mise en œuvre est appelé par point.

1.5.2 Etude de performance sur CRAYT3E

Nous comparons les différentes mises en œuvre parallèles synchrones, asynchrones par sous-domaines, asynchrones par point et asynchrone avec communication flexible en utilisant les bibliothèques de passage de message MPI et SHMEM. Nous comparons au début la mise en œuvre synchrone et asynchrone par sous-domaines pour les différents problèmes évoqués au sous-paragraphe 1.4.2, ensuite nous comparons la mise en œuvre asynchrone par sous-domaines et la mise en œuvre asynchrone par point et finalement nous comparons

dimension du vecteur prix	48	72	96	144
type de problème	flots quadratiques			
gradient conjugué (sec)	0.084	0.299	0.720	2.499
type gradient (sec)	0.015	0.050	0.120	0.417
type de problème	flots turbulents			
gradient conjugué (sec)	2.800	9.387	22.227	74.882
type gradient (sec)	0.732	2.224	4.865	14.558
type de problème	flots de communication			
gradient conjugué (sec)	0.319	0.991	2.279	7.390
type gradient (sec)	0.446	1.281	2.734	8.057

TAB. 1.1 – *Problèmes de flot : comparaison de la méthode de type gradient et de la méthode du gradient conjugué*

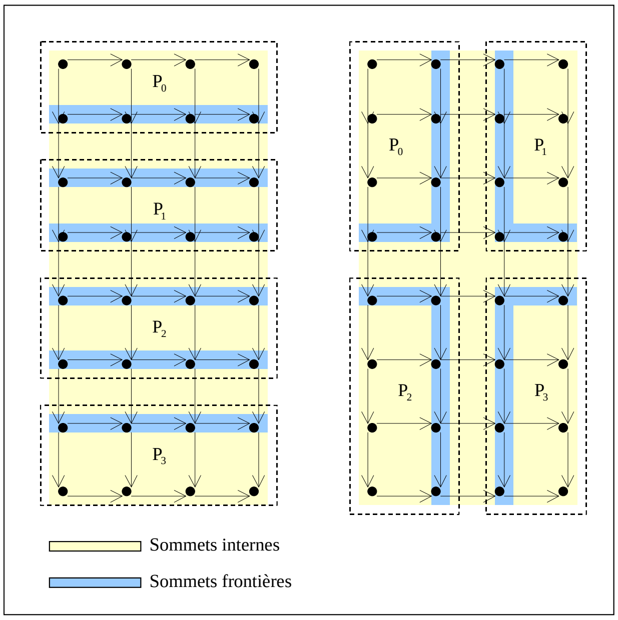

FIG. 1.2 – *décomposition du réseau sur les processeurs*

la mise en œuvre asynchrone par point et la mise en œuvre asynchrone avec communication flexible.

Nous présentons sous forme de courbes l'efficacité en fonction du nombre de processeurs pour la mise en œuvre synchrone et la mise en œuvre asynchrone sur un des exemples traités qui correspond à une grille bidimensionnelle ayant 64 lignes et 3 colonnes. Notons par (s_x) la mise en œuvre parallèle synchrone, par (asd_x) la mise en œuvre parallèle asynchrone par sous-domaines, par (as_x) la mise en œuvre asynchrone par point et par (asf_x) la mise en œuvre asynchrone avec communication flexible, où x est le nom de la bibliothèque de passage de message utilisée afin de communiquer les données entre les processeurs.

a) Comparaison entre la mise en œuvre synchrone et la mise en œuvre asynchrone par sous-domaines

Les figures (1.3, 1.4 et 1.5) montrent que la mise en œuvre asynchrone par sous-domaines de la méthode de type gradient est toujours plus efficace que la mise en œuvre synchrone. Avec MPI, la mise en œuvre asynchrone devient moins performante. Les communications ne se font pas au même rythme sur tous les processeurs; les données sont stockées dans une file du type FIFO (First in First out). Etant donné que la séquence du vecteur itéré est monotone, nous avons intérêt dans ce cas à lire la dernière valeur disponible sur le tampon. La procédure de scrutation de la file augmente le temps de latence quand le nombre de messages est important. Nous notons que l'utilisation de la bibliothèque SHMEM est plus naturelle pour une mise en œuvre asynchrone, car la prise en compte des données transmises se fait comme sur une architecture à mémoire partagée.

Remarque: Autre limitation de la mise en œure asynchrone avec MPI : utilisation d'un nombre limité de processeurs toujours pour les mêmes raisons.

b)Comparaison entre la mise en œuvre asynchrone par sous-domaines et asynchrone par point

Remarque : La figure (1.6) montre que la méthode de type gradient asynchrone par sous-domaines est plus performante dans le cas d'un problème de flot que pour les autres problèmes. Nous présentons par la suite les résultats des mises en œuvre asynchrones de l'algorithme de type gradient dans le cas de résolution d'un problème de flot turbulent.

La figure (1.7) montre que dans le cas d'un problème de flot turbulent, la mise en œuvre asynchrone par point est plus performante que celle en asynchrone par sous-domaines. En effet, d'après la monotonie de la convergence, la communication de la valeur d'une composante à la fin de chaque réactualisation permet de converger plus rapidement vers la solution du problème.

Remarque: Théoriquement l'efficacité de l'algorithme parallèle est toujours strictement inférieure à un. Nous avons toutefois obtenu des efficacités pour la mise en œuvre parallèle asynchrone par point qui sont supérieures à un. Ce phénomène est connu sous le nom d'accélération super linéaire. Le principe de ce phénomène est le suivant : en augmentant le nombre de processeurs sans augmenter en proportion la taille du problème, le domaine à traiter par processeur sera plus petit. Avec un domaine très réduit, le processeur retrouvera souvent des éléments déjà stockés dans la mémoire cache, d'où un temps d'accès mémoire plus court et bien sûr des performances meilleures. Mais au delà d'un certain nombre de processeurs, le temps de communication devient relativement important comparé à la charge de travail et compense l'accélération (effet de la granularité). Cette remarque ne concerne que les machines parallèles à processeurs RISC telles que le CRAY T3E.
Nous comparons au paragraphe suivant les performances de la mise en œuvre asynchrone par point avec celle de la mise en œuvre asynchrone avec communication flexible.

c)Comparaison entre la mise en œuvre asynchrone par point et asynchrone avec communication flexible

La communication des itérés partiels permet théoriquement de converger plus rapidement vers la solution du problème dans le cas d'une convergence monotone. Cependant, la figure (1.8) montre pour un faible degré (degré maximum ≤ 4) que la mise en œuvre asynchrone par point reste toujours plus efficace. Dans ce cas en effet, l'envoi des itérés partiels ne permet pas d'avoir un gain significatif en raison de la faible granularité et de l'excès du nombre de communication. Toutefois dans le cas où la granularité est plus importante on constate que les itérations asynchrones avec communication flexible sont plus

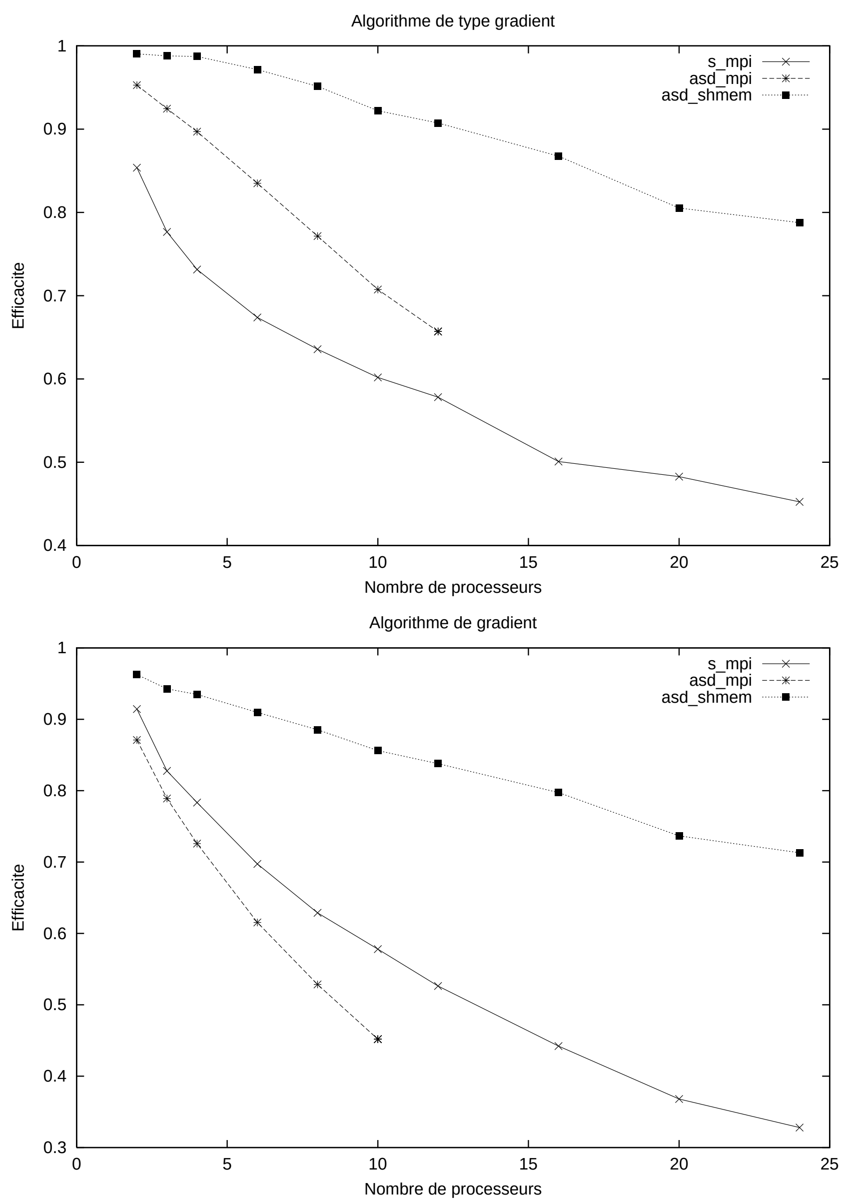

FIG. 1.3 – *Problème de flot turbulent*

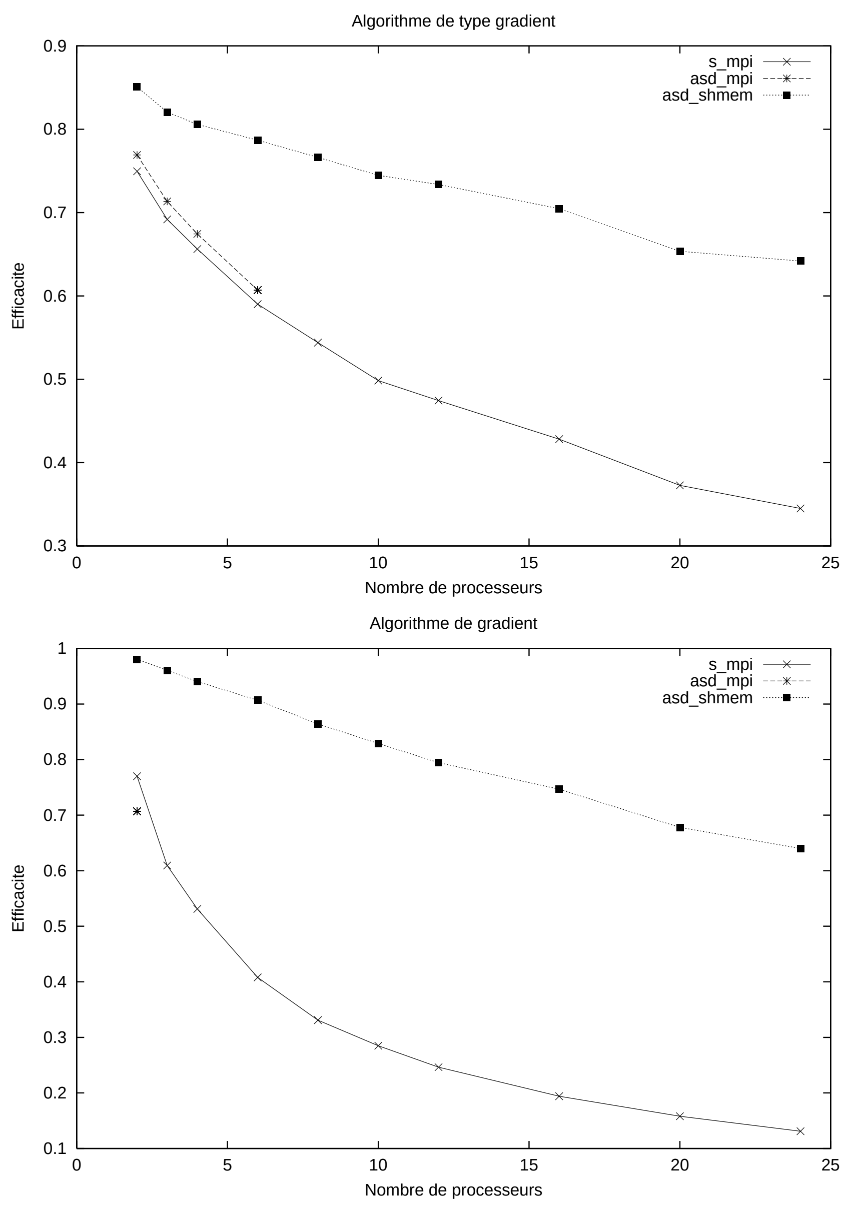

FIG. 1.4 – *Problème de flot de communication*

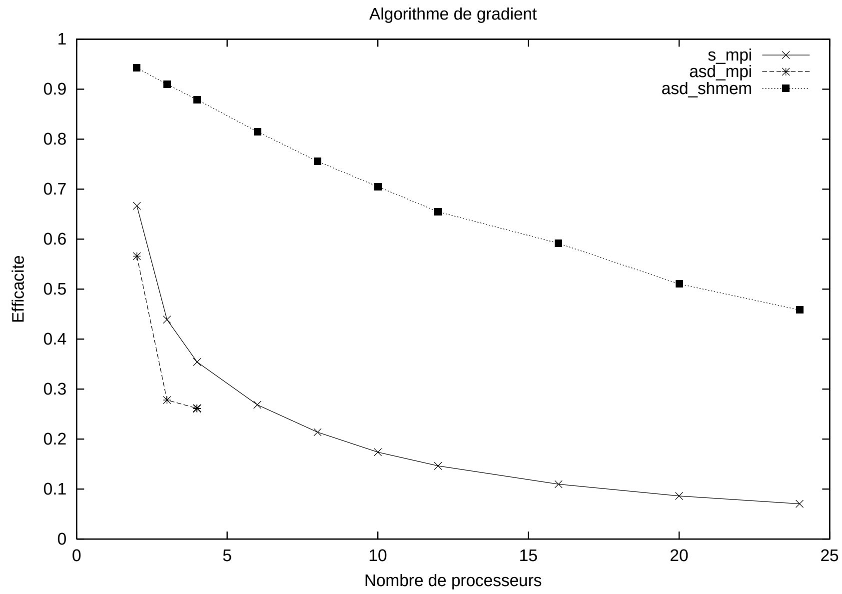

Fig. 1.5 – *Problème de flot quadratique*

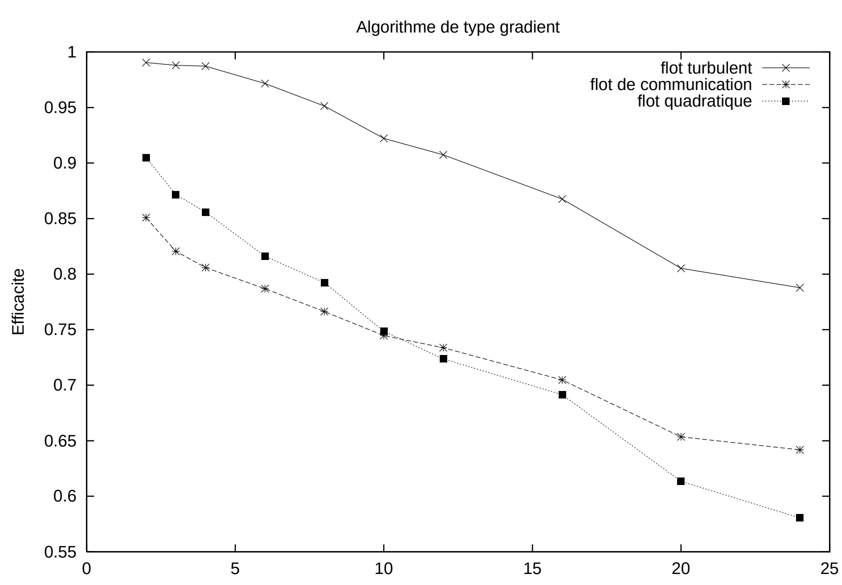

Fig. 1.6 – *Comparaison des performances des itérations asynchrones par sous-domaines pour les différents problèmes*

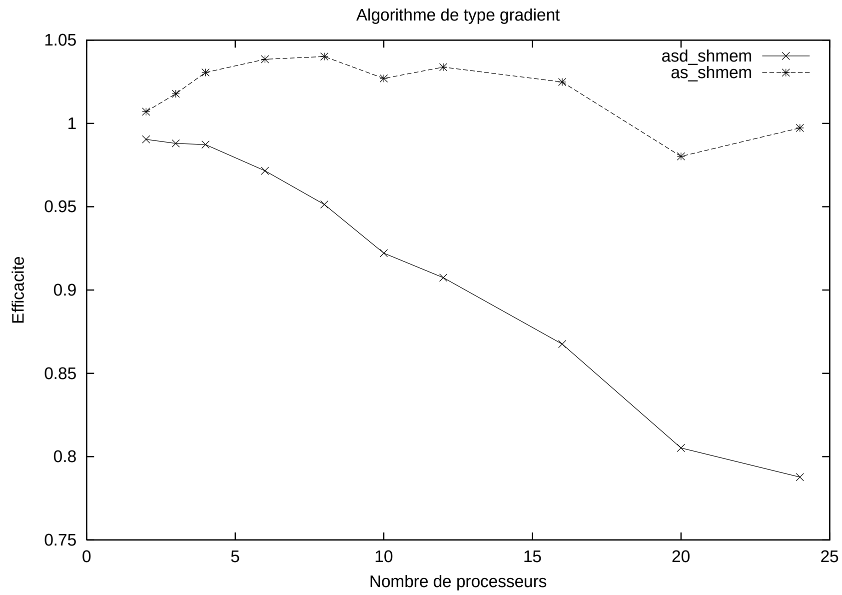

Fig. 1.7 – *Comparaison des performances des itérations asynchrones par sous-domaines et asynchrone par point*

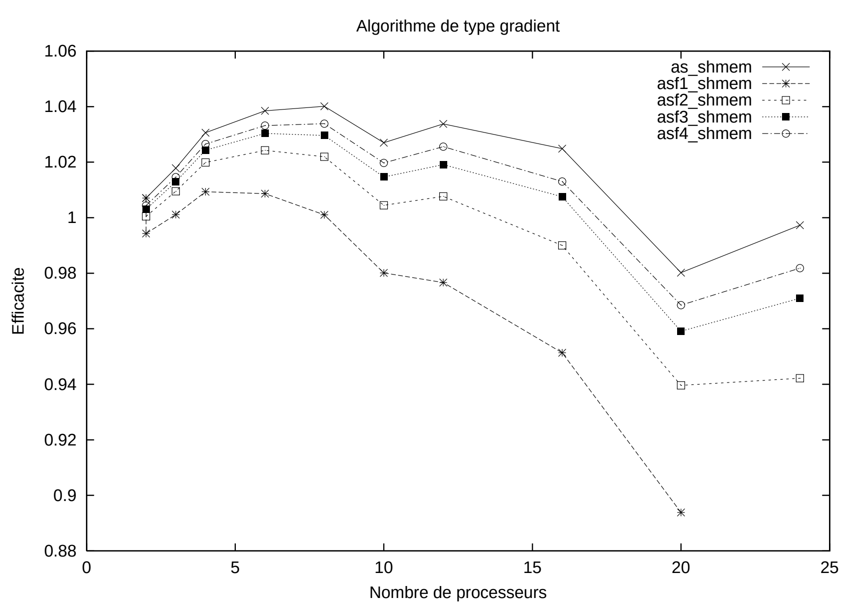

Fig. 1.8 – *Comparaison des performances des itérations asynchrones par point et asynchrone avec communication flexible pour le cas d'un degré maximum égal à 4*

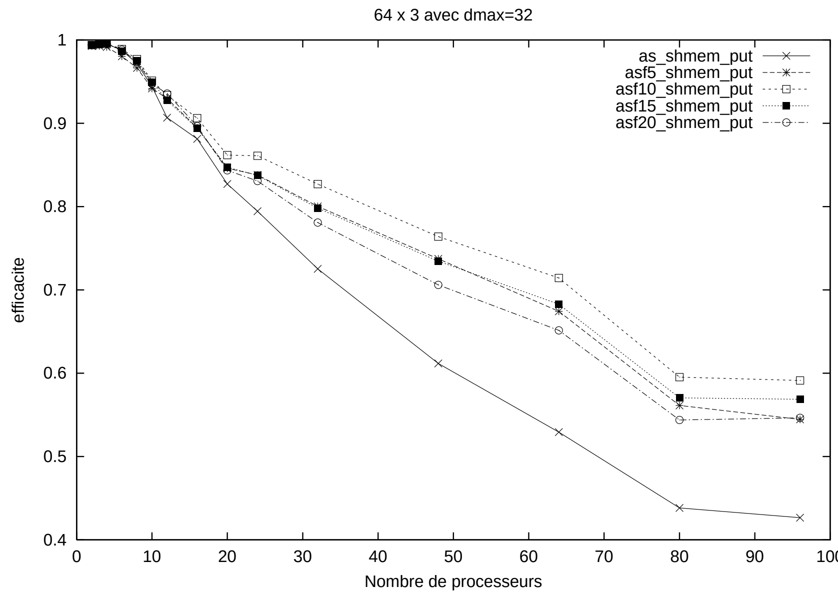

FIG. 1.9 – *Comparaison des performances des itérations asynchrones par point et asynchrone avec communication flexible pour le cas d'un degré maximum égal à 32*

performantes que les itérations asynchrones par point (cf. figure (1.9)) qui correspond à des réseaux de degré maximum égal à 32.

1.5.3 Etude de performance sur SMP

Sur une architecture a mémoire partagée, les processeurs peuvent accéder directement aux données communes. Dans ce cas on note que la lecture de la valeur courante des composantes du vecteur itéré s'effectue très naturellement et qu'elle peut donc correspondre à un itéré partiel réactualisé par un autre processeur. Le tableau 1.2 montre les performances des mises en œuvre synchrones et asynchrones avec communication flexible sur une grille de dimension 64x3 avec un degré maximum égal à 32. Nous ne donnons pas les résultats pour les itérations asynchrones au sens classique, car la mise en œuvre avec communication flexible est plus naturelle.

	Synchrones		Asynchrone avec c.f	
Threads	Temps(sec)	Efficacité	Temps (sec)	Efficacité
1	205.61	1	205.61	1
2	116.92	0.879	103.66	0.991
3	81.25	0.843	69.22	0.990
4	62.45	0.823	52.16	0.998

TAB. 1.2 – *Performance sur une SMP*

Le tableau 1.2 montre que la mise en œuvre asynchrone avec communication flexible est beaucoup plus performante que la mise en œuvre synchrone. L'accès à la valeur courante du vecteur itéré par une simple lecture de la mémoire commune permet d'avoir une très bonne efficacité.

1.5.4 Etude de performance sur un réseau de SMP

Dans ce sous-paragraphe nous noterons $(a_1,a_2,...,a_n)$ le n-uplet dont chaque élément correspond au nombre de threads qui sont crées sur chaque SMP.

	Synchrone		Asynchrone avec c.f	
Threads	Temps(sec)	Efficacité	Temps(sec)	Efficacité
(1,0,0,0)	199.97	1	199.97	
(2,0,0,0)	113.60	0.880	100.82	0.992
(1,1,0,0)	118.38	0.845	107.96	0.926
(2,1,0,0)	80.93	0.824	70.22	0.949
(2,2,0,0)	63.29	0.789	52.82	0.946
(1,1,1,0)	85.55	0.779	77.21	0.863
(2,1,1,0)	64.19	0.779	55.76	0.897
(2,2,1,0)	52.94	0.754	43.80	0.913
(2,2,2,0)	45.50	0.731	37.31	0.893
(1,1,1,1)	68.19	0.731	62.32	0.802
(2,1,1,1)	54.32	0.735	47.48	0.824
(2,2,1,1)	45.86	0.726	38.92	0.856
(2,2,2,1)	40.38	0.707	33.26	0.859
(2,2,2,2)	36.21	0.699	30.07	0.831

TAB. 1.3 – *Le temps d'exécution et l'efficacité de la mise en œuvre parallèle sur une réseau Ethernet en utilisant MPICH*

Les tableaux 1.3 à 1.6 montrent que les performances des itérations asynchrones avec communication flexible sur un réseau de SMP sont toujours meilleures que celles des itérations synchrones. Les performances varient selon la bibliothèque de communication et la nature de réseau. Au paragraphe suivant nous revenons sur ces résultats ainsi que sur l'ensemble des résultats présentés afin d'étudier en détail la variation de performance des méthodes parallèles en fonction de l'architecture et de l'environnement de programmation parallèle utilisés.

	Synchrone		Asynchrone avec c.f	
Threads	Temps(sec)	Efficacité	Temps(sec)	Efficacité
(1,1,0,0)	118.45	0.844	105.65	0.946
(2,1,0,0)	81.18	0.821	69.22	0.963
(2,2,0,0)	63.09	0.791	51.95	0.962
(1,1,1,0)	84.94	0.784	73.45	0.907
(2,1,1,0)	63.94	0.782	54.16	0.923
(2,2,1,0)	52.46	0.762	42.64	0.938
(2,2,2,0)	45.03	0.740	36.31	0.918
(1,1,1,1)	67.55	0.744	57.18	0.870
(2,1,1,1)	53.97	0.739	44.90	0.890
(2,2,1,1)	45.34	0.735	37.25	0.895
(2,2,2,1)	39.82	0.717	31.49	0.907
(2,2,2,2)	35.60	0.702	28.51	0.876

TAB. 1.4 – *Le temps d'exécution et l'efficacité de la mise en œuvre parallèle sur une réseau Myrinet en utilisant MPICH-gmm*

	Synchrone		Asynchrone avec c.f	
Threads	Temps(sec)	Efficacité	Temps(sec)	Efficacité
(1,1,0,0)	120.13	0.832	109.23	0.950
(2,1,0,0)	82.31	0.810	71.16	0.937
(2,2,0,0)	67.98	0.735	55.19	0.926
(1,1,1,0)	88.59	0.750	79.38	0.842
(2,1,1,0)	62.42	0.801	55.63	0.867
(2,2,1,0)	55.62	0.719	46.90	0.853
(2,2,2,0)	51.82	0.643	41.09	0.811
(1,1,1,1)	111.55	0.448	64.97	0.769
(2,1,1,1)	91.65	0.436	50.07	0.798
(2,2,1,1)	50.42	0.661	42.55	0.787
(2,2,2,1)	68.73	0.415	37.03	0.771
(2,2,2,2)	62.41	0.400	34.32	0.728

TAB. 1.5 – *Le temps d'exécution et l'efficacité de la mise en œuvre parallèle sur une réseau Myrinet en utilisant LANDA*

1.6 Les facteurs qui interviennent sur les performances de la mise en œuvre parallèle asynchrone

Dans ce paragraphe nous analysons l'effet des facteurs essentiels qui jouent sur les performances des itérations parallèles en tenant compte des architectures et environnements de communication utilisés.

1.6.1 Effet du déséquilibre des charges de calcul

Afin d'étudier l'effet du déséquilibre des charges de calcul entre les processeurs sur la mise en œuvre parallèle, on peut se rapporter aux figures relatives aux méthodes de gradient et de type gradient pour les différents types de problème sur le CRAYT 3E. La méthode de gradient permet d'avoir la même charge de calcul sur tous les processeurs quand le nombre de nœuds est équitablement réparti, alors que la méthode de type gradient ne le permet pas vu que le nombre de pas de gradient nécessaires pour faire une recherche unidirectionnelle varie d'une itération à l'autre. On a donc dans ce dernier cas un déséquilibre indéterministe des charges de calcul. Les résultats de la mise en œuvre parallèle pour les différents types de problèmes montrent que le modèle asynchrone nous permet d'avoir de bonnes performances tout particulièrement quand la charge de calcul des processeurs est déséquilibré. Le déséquilibre de charge engendre des temps d'attente dus aux synchronisations et cause une perte d'efficacité de la mise en œuvre synchrone. La mise en œuvre asynchrone est plus efficace en utilisant la bibliothèque SHMEM pour communiquer les données. Cette bibliothèque offre des primitives de communication qui sont de nature typiquement asynchrone et qui se prêtent bien à ce genre de mise en œuvre. Nous étudions par la suite les facteurs qui peuvent faire chuter les performances de la mise en œuvre asynchrone, ainsi que les limites d'applicabilité du modèle asynchrone et asynchrone avec communication flexible.

	Synchrone		Asynchrone avec c.f	
Threads	Temps(sec)	Efficacité	Temps(sec)	Efficacité
(1,1,0)	421.29	0.55	326.13	0.710
(2,1,0)	309.32	0.500	241.13	0.640
(3,1,0)	240.92	0.480	191.94	0.600
(1,1,1)	230.66	0.670	218.65	0.700
(2,1,1)	196.54	0.590	183.05	0.630
(3,1,1)	171.63	0.540	235.27	0.580

Tab. 1.6 – *Le temps d'exécution et l'efficacité de la mise en œuvre parallèle sur une réseau Myrinet en utilisant ATHAPASCAN*

1.6.2 Effet de la granularité

Nous pouvons remarquer que lorsque le nombre de processeurs augmente, le nombre de communications augmente et la charge de calcul de chaque processeur diminue car le domaine associé devient plus réduit. Dans ce cas là, le temps de communication a plus de poids et fait chuter les performances de la mise en œuvre parallèle (cf. figures (1.3), (1.4) et (1.5). Le rapport entre le temps de calcul et le temps de communication est appelé grain de la mise en œuvre parallèle. Quand la granularité est faible, La mise en œuvre asynchrone avec MPI pose un problème de saturation du tampon d'émission-réception. En ce qui concerne la résolution d'un problème de flot turbulent, la saturation se produit sur un nombre de processeurs qui dépasse 12, et pour les autres types de problèmes la saturation est produite à partir de 2 processeurs pour le problème de flot de communication et à partir de 4 processeurs pour le problème de flot quadratique. Ceci peut s'expliquer par le fait que la saturation du tampon se produit pour un nombre réduite de processeurs d'autant plus rapidement que le déséquilibre des charges de calcul est important et la granularité est faible.

Remarque : Le facteur granularité n'est pas le seul qui soit responsable de la chute de performance de la mise en œuvre parallèle : le taux de convergence du schéma itératif peut diminuer aussi quand le nombre de processeurs augmente. Le tableau 1.2 confirme cette analyse, car sur une architecture à mémoire partagée où le temps de communication entre un processeur et la mémoire commune est très faible, les performances chutent quand le nombre de processeur augmente. La granularité est un facteur très important pour la mise en œuvre parallèle d'une application numérique. Elle dépend de plusieurs paramètres matériels et logiciels de l'architecture utilisée. Nous allons étudier par la suite ces paramètres, ainsi que la variation de l'efficacité de la mise en œuvre parallèle selon le type d'architecture parallèle et la bibliothèque de communication.

1.6.2.1 Effet du temps de latence

En comparant les résultats de la mise en œuvre asynchrone en utilisant les deux bibliothèques MPI et SHMEM, les figures(1.3, 1.4 et 1.5) montrent que la mise en œuvre asynchrone en utilisant SHMEM est la plus performante. Ceci est dû en fait au temps mis par les primitives de communication pour préparer l'envoi ou la réception des données. Ce temps est appelé temps de latence. Dans le cas des fonctions MPI_(I)Send et MPI_(I)Recv, le temps de latence mis est de l'ordre de $11\mu s$ (cf. [CDG97]), alors qu'avec shmem_put() et shmem_get(), le temps de latence est de l'ordre de $1.65\mu s$ (cf. [CG96]). En utilisant la bibliothèque MPI, nous avons mis en œuvre une fonction de scrutation du tampon qui augmente bien sûr le temps de latence global. Les résultats sur un réseau de SMP nous montrent aussi que la mise en œuvre parallèle asynchrone est d'autant plus efficace que le temps de latence des primitives de communication est faible. En utilisant la bibliothèque MPICH-gmm sur un réseau Myrinet, nous avons mesuré expérimentalement un temps de latence de l'ordre de $30\mu s$, alors qu'en utilisant la bibliothèque MPICH sur un réseau Ethernet, nous avons mesuré un temps de latence de l'ordre de $250\mu s$. Le temps de latence mesuré sur un réseau Ethernet en utilisant LANDA et ATHAPASCAN est de l'ordre de $300\mu s$.

1.6.2.2 Effet du débit de communication

Le débit de communication est un facteur qui mesure le nombre de méga octets par seconde véhiculé par le réseau. Il est dépendant de la caractéristique du réseau. Pour mesurer effectivement ce facteur, il faut faire une moyenne du temps de propagation sur le réseau en faisant un "ping-pong" ou encore un envoi-réception entre deux processeurs ou deux machines distantes. Selon le type de réseau il faut faire un choix de la taille du problème afin d'avoir un grain assez satisfaisant. Sur une architecture à mémoire partagée, l'accès à la valeur courante est immédiat et ne prend que le temps accès mémoire. L'efficacité est dans ce cas maximale (cf. [JEG98] et [Gaz98]). Sur un réseau de SMP le débit de communication est plus faible que sur le CRAY T3E. En utilisant différents types de réseau ; ce qui revient à faire varier le débit de communication ainsi que le temps de latence, les performances de la mise en œuvre parallèle varient. La mise en œuvre parallèle sur le réseau Myrinet à 1Goctets/s théorique et 500 Moctets/s mesurée est plus performante que sur le réseau Ethernet a 100Moctets théorique et 65Moctets mesurée. En combinant les deux aspects de mise en œuvre sur architecture à mémoire distribuée et à mémoire partagée tel qu'on peut les trouver sur un réseau Ethernet ou Myrinet de SMP, les performances augmentent en général quand on utilise tous les processeurs de la SMP (cf. Table 1.3, 1.4 et 1.5), car les accès à la mémoire commune sont très rapides et récompensent les temps de communication entre les SMP. Ceci montre l'intérêt des threads qu'on appelle aussi processus légers, car ils consomment d'une part peut de temps de communication entre processeurs et mémoire et d'autre part ils utilisent les mêmes ressources logicielles que celui du père.

1.6.2.3 Effet de l'environnement de programmation parallèle

Les résultats expérimentaux montrent que les performances de la mise en œuvre parallèle sont pratiquement fonction de l'environnement ou de la bibliothèque de programmation parallèle utilisée. La mise en œuvre parallèle en utilisant l'environnement LANDA est moins performante que celle en utilisant MPICH, car le noyau LANDA est conu à partir de l'aspect multiserveur qui ajoute un temps supplémentaire au temps de communication nécessaire pour traiter les messages au sein du serveur de communication. Avec ATHAPASCAN, la machine n'est pas dédiée à cent pour cent à l'application vu qu'il y a toujours un threads de scrutation additionnel pour traiter les messages à envoyer aux autres machines. Ce type de mise en œuvre exige une machine qui est dédié uniquement à l'exécution de l'algorithme itératif parallèle (cf. [Car99]).

1.6.3 Effet du degré d'asynchronisme

Le degré d'asynchronisme, ou niveau d'asynchronisme est un facteur très important qui permet d'améliorer les performances de la mise en oeuvre asynchrone. Par rapport à la

mise en œuvre synchrone, on obtient une première amélioration en communiquant toutes les valeurs des composantes frontières à la fin d'une itération sans aucune synchronisation entre processeurs, d'où l'interêt d'une mise en oeuvre asynchrone. Ensuite, d'après la figure (1.7) la communication de la valeur de chaque composante frontière à la fin d'une réactualisation permet d'améliorer d'avantage les performances des méthodes asynchrones. Le dernier niveau mis en œuvre, qui est aussi le plus général et le plus flexible, consiste à communiquer ou à accéder à la valeur d'une composante au cours d'une réactualisation. Cette valeur ne correspond pas alors forcément au dernier itéré réactualisé par les autres processeurs et peut ne pas être étiquetée par un numéro d'itération.
Nous présentons maintenant une étude statistique sur la variation du nombre de pas de gradient pour chaque composante au cours des itérations.
Soit $\overline{n}_i$ le nombre moyen de pas de gradient effectués au cours d'une recherche unidirectionnelle du nœud i et $\overline{N}$ une moyenne des $\overline{n}_i$ sur tous les nœuds. La figure (1.10) montrent que l'augmentation du degré de chaque nœud du graphe entraîne une augmentation du nombre moyen $\overline{N}$. Selon les résultats de la figure (1.9), quand le nombre moyen est grand, les performances des mises en œuvre asynchrones avec communication flexible en utilisant la fonction shmem_put() sont meilleures que celles des mises en œuvre asynchrones classiques. On note par asfx_shmem_put, les courbes d'efficacités des mises en œuvre asynchrone avec communication flexible avec une politique d'envoi des itérés partiels tous les x pas de gradient. Quand le degré maximum est égal à 32, la politique optimale d'envoi des itérés partiels est tous les 10 pas. Les courbes d'efficacité des mises en œuvre asynchrones avec communications flexibles quelque soit la politique d'envoi des itérés partiels sont toutes au dessus de la courbe relative à une mise en œuvre asynchrone.

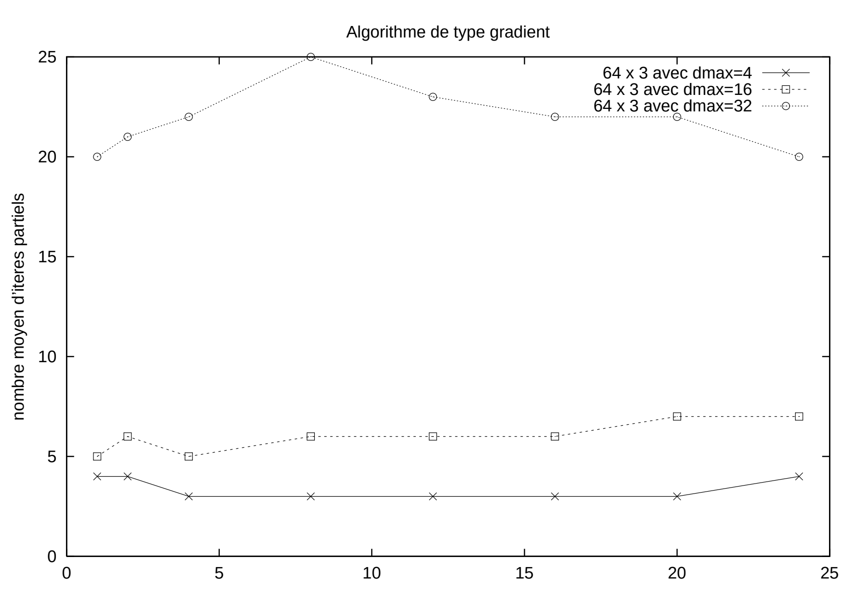

FIG. 1.10 – *Variation du nombre moyen d'itérés partiels*

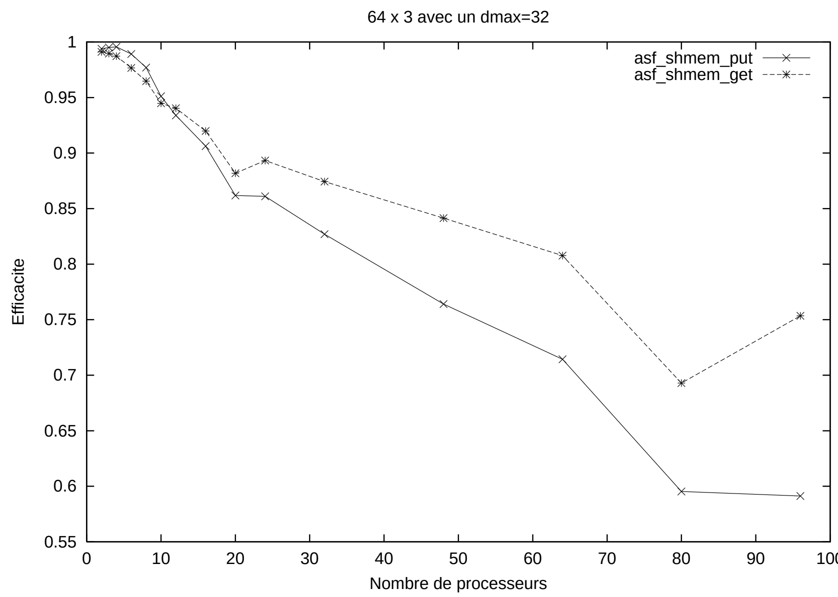

FIG. 1.11 – *Comparaison entre la meilleure politique d'envoi des itérés partiel avec shmem_put() et la mise en œuvre utilisant shmem_get()*

La mise en œuvre asynchrone avec communication flexible peut être vue comme étant l'accès à la valeur courante du vecteur itéré. La figure (1.11) montre que l'utilisation de la fonction shmem_get() de SHMEM permet d'avoir des efficacités meilleures que toutes les politiques d'envoi des itérés partiels avec la fonction shmem_put(). Cette mise en œuvre asynchrone avec communication flexible est optimale et similaire à une mise en œuvre sur architecture à mémoire partagée.

1.7 Conclusion

Les différentes mises en œuvre parallèles ont été appliquées à la résolution d'un problème de flot convexe dans les réseaux. La résolution consiste à minimiser un critère séparable sous des contraintes de conservation de flux en chaque sommet du graphe. Ce problème peut être transformé d'après la théorie de la dualité en un problème de minimisation d'une fonctionnelle convexe sans contrainte. Les inconnues représentent alors les potentiels ou prix en chaque sommet du graphe. La minimisation de la fonction duale est simple, et revient à annuler toutes les composantes de son gradient. Le problème dual peut être transformé dans ce cas en un problème dual réduit en fixant une des composantes du vecteur prix à une valeur constante. Puisque le dual réduit est sans contrainte et différentiable, il est naturel d'envisager sa résolution numérique au moyen des méthodes itératives de descente. Parmi ces méthodes, nous avons mis en œuvre la méthode de gradient et la méthode de type gradient pour résoudre trois types de problèmes : un problème de flot turbulent, un problème de flot de communication et un problème de flot quadratique. Les résultats expérimentaux ont montré que la mise en œuvre asynchrone est toujours plus performante que la mise en œuvre synchrone sur toutes les architectures décrites au chapitre I.2. Notons que ce problème induit un déséquilibre de charge non déterministe entre les différents processeurs. Les performances des itérations asynchrones par sous-domaines peuvent être améliorées, dans ce cas d'algorithmes itératifs parallèles où la séquence du vecteur itéré est monotone, en communiquant la valeur de chaque composante juste après sa réactualisation (cas d'une mise en œuvre asynchrone par point) ou en communiquant cette valeur en cours de réactualisation (cas d'une mise en œuvre asynchrone avec communication flexible). Ce gain de performance est fonction de la granularité du problème, de l'architecture et de l'environnement de programmation parallèle utilisés. Nous avons vu en particulier l'intérêt de l'utilisation de la bibliothèque de programmation parallèle telle que SHMEM sur le CRAY T3E dont certaines fonctions conviennent particulièrement à la mise en œuvre des itérations asynchrones. Sur un réseau rapide, nous avons vu aussi l'intérêt des itérations asynchrones avec communication flexible en utilisant la bibliothèque MPICH combinée avec l'emploi des threads POSIX.

Chapitre 2

Résolution d'un problème de commande optimale

2.1 Introduction

Dans ce chapitre nous nous proposons d'illustrer les performances des mises en œuvre asynchrones avec communication flexible ainsi que d'autres mises en œuvre parallèles dans le cas particulier de la méthode de l'état adjoint par relaxation appliquée à un problème de commande optimale pour laquelle nous sommes dans un contexte favorable de convergence. De nombreux travaux ont été effectués afin d'appliquer des schémas itératifs parallèles asynchrones à un problème de commande optimale pour la régulation d'un processus thermique. Parmi ces travaux nous citons ceux de Miellou, Spiteri et Lang (cf. [Mie72], [Spi84] et [LMS86]) qui ont introduit une variante asynchrone de la méthode de l'état adjoint par relaxation. Cette méthode autorise l'échange des résultats de calcul entre les processeurs dès qu'ils sont mis à jour. Ce type d'algorithme a été formalisé de faon mathématique et l'application de point fixe associée est contractante en norme scalaire dans l'espace des fonctions continues par morceaux (cf. [Spi84]). Une simulation du schéma itératif parallèle asynchrone relatif à la méthode de l'état adjoint par relaxation a été effectuée sur une machine monoprocesseur (cf. [Spi84]). D'autres travaux ont été effectués aussi sur ce même type de problème de commande optimale en appliquant un schéma itératif par bloc qui converge d'une manière monotone vers la solution (cf. [Jac76b] et [Jac77]).

Nous proposons dans ce chapitre une étude de performance sur diverses architectures parallèles de la méthode de l'état adjoint par relaxation avec échange asynchrone. Ce schéma itératif nécessite la résolution d'un système linéaire par bloc de très grande dimension. Le calcul d'un bloc de composantes peut être soit direct en inversant une matrice (solution très coteuse dans le cas où la dimension de la matrice est grande) soit indirect en mettant en œuvre un deuxième niveau de schéma itératif. Frommer et Szyld ont proposé des schémas itératifs à deux niveaux (cf. [FS94] et [FS98]) pour la résolution d'un système linéaire par bloc. La résolution du problème de commande optimale par un schéma itératif à deux niveaux s'adapte bien à une mise en œuvre asynchrone avec communication flexible dont nous proposons une analyse de convergence.

Le but principal de ce chapitre est d'évaluer, pour ce type de problème, les performances des itérations synchrones, asynchrones et asynchrones avec communication flexible sur diverse architectures parallèles et d'étudier l'influence de quelques caractéristiques matérielles et logicielles de l'architecture.

2.2 Equations canoniques d'un problème de commande optimale

Soit le problème de commande optimale suivant :

$$\begin{cases} J = \phi(y_0,y_f,t_0,t_f) + \displaystyle\int_{t_0}^{T} L(y,u,t)dt, \\ \dot{y} = f(y,u,t), \end{cases} \tag{2.1}$$

où y est le vecteur d'état du système de dimension n', u est la commande de dimension m, ϕ est fonction des conditions initiales et finales t_0, t_f, $y_0 = y(t_0)$ et $y_f = y(t_f)$, f et L sont respectivement deux fonctions de $\mathbb{R}^{n'+m+1}$ dans $\mathbb{R}^{n'}$ et de $\mathbb{R}^{n'+m+1}$ dans $\mathbb{R}$.
Le problème consiste à trouver une commande $u \in U_{ad}$ qui minimise le critère J, U_{ad} étant un ensemble convexe des commandes admissibles.
L'Hamiltonien du problème (2.1) est le suivant :

$$H = L(y,u,t) + p^T f(y,u,t), \tag{2.2}$$

où p est le vecteur d'état adjoint du problème.
Les équations canoniques du problème de commande optimale sont :

$$\begin{cases} \dfrac{\partial H}{\partial y} = -\dfrac{dp}{dt}, \\ \dfrac{\partial H}{\partial p} = \dfrac{\partial y}{\partial t}, \\ \dfrac{\partial H}{\partial u} + \partial \Psi_{U_{ad}} \ni 0, \end{cases} \tag{2.3}$$

où $\partial \Psi_{U_{ad}}$ est le sous-différentiel de la fonction indicatrice $\Psi_{U_{ad}}$ de l'ensemble convexe U_{ad}.

2.3 Etude de la régulation d'un processus thermique

2.3.1 Position du problème

Considérons un système physique composé d'un four à moufle à n zones de chauffage (on dit aussi à n tranches). L'étude consiste à amener la température z relevée en n points du barreau à une température z_d en un temps fini $(T - t_0)$, qui représente l'horizon de commande. Il faut donc déterminer la commande u qui correspond en fait à l'intensité des courants de régulation, de telle sorte que, au bout du temps T, la température z du barreau soit uniformément égale à z_d compte tenu du critère suivant :

$$J = \frac{1}{2}\int_{t_0}^{T}(|Cy(t) - z_d|_2^2 + k|u(t) - u_d|_2^2)dt, \tag{2.4}$$

où $y \in \mathbb{R}^{n'}$, avec $n' = 2n$, car l'état du système est représenté par un vecteur dont les composantes sont relatives aux températures en n points du barreau et n points du four. Le modèle suivant a été obtenu en étudiant l'évolution du système au tour d'un point de fonctionnement:

$$\begin{cases} \dfrac{dy(t)}{dt} + Ay(t) = Bu(t), \forall t \in [t_0,T], \\ y(t_0) = y_0, \\ z(t) = Cy(t), \end{cases} \tag{2.5}$$

A, B et C sont respectivement des matrices de dimension $(2n,2n)$, $(2n,m)$ et $(n,2n)$. A étant de plus une matrice ayant la propriété d'être une M-matrice (cf. Définition A.1.1 et A.1.5 de l'annexe), u est un vecteur de $\mathbb{R}^n$. L'équation (2.5) constitue l'équation d'état du modèle et l'observation (la variable z modélise l'observation du système).
L'Hamiltonien du problème de commande optimale est :

$$H = \frac{1}{2}(|Cy(t) - z_d|_2^2 + k|u(t) - u_d|_2^2) + p^T(t)[-Ay(t) + Bu(t)]. \tag{2.6}$$

En appliquant les équations canoniques (2.3) on obtient le système suivant :

$$\begin{cases} \begin{cases} \dfrac{\partial y}{dt} + Ay - Bu = 0 \text{ et} \\ y(0) = 0, \end{cases} \\ \begin{cases} -\dfrac{\partial p}{dt} + A^*p - C^*Cy = -C^*z_d \text{ et} \\ p(T) = 0, \end{cases} \\ u \in -\dfrac{1}{k}B^*p + u_d + \partial\Psi_{U_{ad}}(u). \end{cases} \tag{2.7}$$

2.3.2 Discrétisation du problème

2.3.2.1 Cas d'un problème sans contrainte

Dans la cas où l'ensemble des commandes admissibles est $\mathbb{R}^m$ alors $u = -\frac{1}{k}B^*p + u_d$. Les équations d'état, d'état adjoint et de commande peuvent s'écrire après substitution de la commande dans la première équation :

$$\begin{cases} \dfrac{\partial y}{dt} + Ay + \dfrac{1}{k}BB^*p = Bu_d, \\ -\dfrac{\partial p}{dt} + A^*p - C^*Cy = -C^*z_d, \\ u = -\dfrac{1}{k}B^*p + u_d, \\ y(0) = p(T) = 0. \end{cases} \tag{2.8}$$

La discrétisation du système différentiel (2.8) par la méthode d'Euler (approximation d'ordre 1) donne le système suivant (cf. [Jac77]) :

$$\begin{cases} \dfrac{y(t+h) - y(t)}{h} + Ay(t+h) + \dfrac{1}{k}BB^*p(t+h) = Bu_d, \\ \dfrac{p(t) - p(t+h)}{h} + A^*p(t) - C^*Cy(t) = -C^*z_d, \\ u(t+h) = -\dfrac{1}{k}B^*p(t+h) + u_d, \\ y(0) = p(T) = 0. \end{cases} \tag{2.9}$$

On pose :

$$M_{11} = \begin{pmatrix} (hA+I) & \mathbb{O} & \dots & \mathbb{O} \\ -I & (hA+I) & & \vdots \\ & \ddots & & \mathbb{O} \\ \mathbb{O} & & -I & (hA+I) \end{pmatrix}, \; M_{12} = \begin{pmatrix} \mathbb{O} & \frac{h}{k}BB^* & & \\ \vdots & & \ddots & \\ & & & \frac{h}{k}BB^* \\ \mathbb{O} & \dots & & \mathbb{O} \end{pmatrix},$$

$$M_{21} = \begin{pmatrix} \mathbb{O} & \dots & & \mathbb{O} \\ -hC^*C & & \vdots & \\ & \ddots & & \\ & & -hC^*C & \mathbb{O} \end{pmatrix}, \; M_{22} = \begin{pmatrix} (hA^*+I) & -I & & \mathbb{O} \\ \mathbb{O} & (hA^*+I) & & \\ \vdots & & \ddots & -I \\ \mathbb{O} & \dots & \mathbb{O} & (hA^*+I) \end{pmatrix}$$

$$\text{et } \mathcal{M} = \begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{pmatrix} \text{est la matrice du système discrétisé.}$$

Il s'agit de résoudre un système d'équations linéaires de dimension $4nN$, où N est le nombre de points de discrétisation, soit en notation matricielle :

$$\begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{pmatrix} \begin{pmatrix} Y \\ P \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \end{pmatrix}. \tag{2.10}$$

2.3.2.2 Cas d'un problème avec contrainte

La commande u appartient à un ensemble convexe $U_{ad} = \Pi_{l=1}^{m}[u_l^{min},u_l^{Max}]$. Soit $\Psi_{U_{ad_l}}$ la fonction indicatrice du convexe U_{ad_l}, alors $\Lambda_l^d = \partial\Psi_{U_{ad_l}}$ où Λ_l^d est défini par :

$$\Lambda_l^d(u_l(t)) = \begin{cases} 0 \text{ si } u_l(t) \in]u_l^{min},u_l^{Max}[, \\ ]-\infty,0] \text{ si } u_l(t) = u_l^{min}, \\ [0,+\infty[\text{ si } u_l(t) = u_l^{Max} \text{ et} \\ \emptyset \text{ sinon} \end{cases} \tag{2.11}$$

Dans le cas où la l-ème composante de la commande est à l'intérieur de l'intervalle des commandes admissibles, on rejoint les équations du système (2.9). Dans le cas où la commande sature les contraintes, la dernière équation du système peut être supprimée et la commande est remplacée par la valeur minimale ou maximale de l'intervalle d'admissibilité de la commande (cf. Figure 2.1).

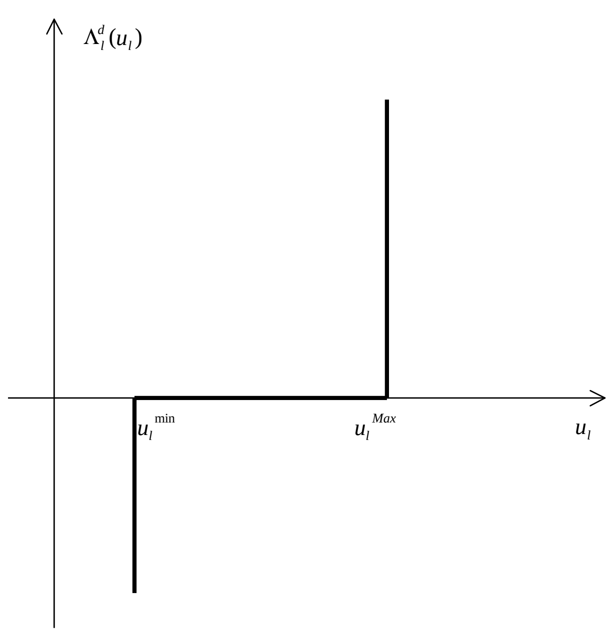

FIG. 2.1 – *Le sous-différentiel de la fonction indicatrice des commandes admissibles*

2.3.3 Détermination de la commande optimale en séquentiel

Nous présentons dans ce paragraphe la résolution en séquentiel du problème de commande optimale en appliquant un schéma itératif à deux niveaux (cf. [FS94] et [FS98]). On suppose par la suite qu'il n'y a pas de contrainte d'admissibilité de la commande .

2.3.3.1 Schémas itératifs à deux niveaux

Quand la dimension des blocs diagonaux de la matrice $\mathcal{M}$ est grande, la résolution directe du système linéaire par bloc en inversant la matrice bloc diagonale $\mathcal{M}_{ii}$ est très coteuse. O'Leary et White ont proposé dans [OLW85] le calcul du vecteur itéré par un schéma itératif à deux niveaux dont le deuxième niveau agit sur une équation bloc du système (voir aussi [FS94] et [BMR97]).
La décomposition de M_{11} et M_{22} respectivement en $M_{11}^1 - M_{11}^2$ et $M_{22}^1 - M_{22}^2$, où M_{11} et M_{22} sont deux matrices non singulières, donne le schéma itératif suivant à un niveau :

$$\begin{cases} M_{11}^1 Y^{(r+1)} = M_{11}^2 Y^{(r)} - M_{12} P^{(r)} + E_1, \\ M_{22}^1 P^{(r+1)} = M_{22}^2 P^{(r)} - M_{21} Y^{(r+1)} + E_2. \end{cases} \tag{2.12}$$

En décomposant les matrices M_{11}^1 et M_{22}^2, il est possible de mettre en œuvre une méthode itérative afin d'approcher $Y^{(r+1)}$ et $P^{(r+1)}$ au lieu de résoudre directement les équations (2.12); on parle alors de méthode à deux niveaux. Plus précisément on suppose que $M_{11}^1 = R - S$ et $M_{22}^2 = G - H$, donc le schéma itératif du second niveau s'écrit de la manière suivante :

$$\begin{cases} RY^{(r+1,s+1)} = SY^{(r+1,s)} + M_{11}^2 Y^{(r)} - M_{12} P + E_1, \\ GP^{(r+1,s+1)} = HP^{(r+1,s)} + M_{22}^2 P^{(r)} - M_{21} Y^{(r+1)} + E_2. \end{cases} \tag{2.13}$$

Le schéma itératif à deux niveaux appliqué au vecteur d'état se présente de la manière suivante :

Premier niveau

$r = 0$
répéter
  $s = 0$
  $Z^{(r,s)} = Y^{(r)}$

  Deuxième niveau
  répéter
    $RZ^{(r,s+1)} = SZ^{(r,s)} + M_{11}^2 Y^{(r)} - M_{12} P + E_1$
    $s = s + 1$
  Jusqu 'à $\left\| Z^{(r,s)} - Z^{(r,s-1)} \right\|_\infty < \varepsilon_2$

  $Y^{(r+1)} = Z^{(r,s)}$
  $r = r + 1$
Jusqu 'à $\left\| Y^{(r)} - Y^{(r-1)} \right\|_\infty < \varepsilon_1$

Schéma itératif à deux niveaux appliqué au vecteur d'état

Remarque : Nous avons considéré une décomposition où M_{11}^1 et M_{22}^1 sont deux matrices bloc diagonales telles que :

$$\begin{cases} M_{11}^1 = diag(M_{11}^{11},...,M_{11}^{1N}) = diag(hA+I,...,hA+I), \\ M_{22}^1 = diag(M_{22}^{11},...,M_{22}^{1N}) = diag(hA^*+I,...,hA^*+I), \end{cases} \tag{2.14}$$

et R et G sont deux matrices diagonales telles que :

$$\begin{cases} R = diag(diag(ha_{11}+1,...,ha_{(2n,2n)}+1),...,diag(ha_{11}+1,...,ha_{(2n,2n)}+1)), \\ G = diag(diag(ha_{11}^*+1,...,ha_{(2n,2n)}^*+1),...,diag(ha_{11}^*+1,...,ha_{(2n,2n)}^*+1)), \end{cases} \tag{2.15}$$

2.3.3.2 Mise en œuvre séquentielle

Nous appliquons la méthode de l'état adjoint par relaxation (cf. [Mie72], [Spi84], [LMS86] et [Jac77]) pour résoudre le problème de commande optimale sans contrainte. La deuxième moitié de la matrice du système (2.10) comporte une sur diagonale par bloc, donc la mise en œuvre de la méthode de Gauss Seidel sur la deuxième partie se ramène en fait à un schéma itératif de type Jacobi par bloc. Afin de rendre le schéma itératif de Gauss Seidel plus efficace, nous intégrons l'état adjoint dans le sens inverse, c'est-à-dire que sur la première partie du système on calcule le 1er bloc, puis le 2ème, etc ... jusqu'au N-ème bloc du vecteur état, puis sur la deuxième partie on calcule le dernier, l'avant dernier, etc ..., jusqu'au $(N+1)$-ème bloc du vecteur état adjoint (cf. [Jac77]). En effet cela revient à itérer suivant le schéma de Gauss Seidel sur une matrice $\mathcal{M}'$ obtenue à partir de la matrice $\mathcal{M}$ par des permutations transformant les blocs sur diagonaux de la deuxième moitié en des blocs sous-diagonaux. Ce procédé est équivalent à remonter le temps sur l'état adjoint. On retrouve donc le procédé classique de résolution d'un problème de commande optimale en intégrant l'état dans le sens direct et l'état adjoint dans le sens inverse.

2.3.3.3 Résultats expérimentaux

Nous avons considéré un four à douze tranches, donc la dimension du système est $48N$, où N est le nombre de pas de discrétisation. Les matrices A, B et C sont définies ci-dessous par:

$$A = \begin{pmatrix} D_1 & S+\theta & \epsilon S & \epsilon^2 S & \dots & \epsilon^9 S & \epsilon^{10} S \\ \theta & D & S+\theta & \epsilon S & \dots & \epsilon^8 S & \epsilon^9 S \\ 0 & \theta & D & S+\theta & & . & . \\ 0 & 0 & \theta & D & & . & . \\ . & . & & & & S+\theta & \epsilon S \\ . & . & & & \theta & D & S+\theta \\ 0 & 0 & . & . & 0 & \theta & D_{12} \end{pmatrix},$$

où les blocs, D_1, D, D_{12}, S et θ, sont des matrices définies de la faon suivante :

$$D1 = \begin{pmatrix} 0.38 & -0.196 \\ -0.0068 & 0.0629 \end{pmatrix}, \ D = \begin{pmatrix} 0.39 & -0.196 \\ -0.0068 & 0.0682 \end{pmatrix}, \ D_{12} = \begin{pmatrix} 0.44 & -0.196 \\ -0.0068 & 0.0559 \end{pmatrix},$$

$$S = \begin{pmatrix} -0.0057 & -0.012 \\ -0.0003 & -0.0008 \end{pmatrix},\ \theta = \begin{pmatrix} -0.0651 & 0 \\ 0 & -0.0031 \end{pmatrix},\ S+\theta = \begin{pmatrix} -0.0708 & -0.012 \\ -0.0003 & -0.0039 \end{pmatrix},$$

$$\begin{cases} B_{ij} = \delta \text{ si } i = 2^p - 1 \text{ et } j = i - p\,;\, p \in \mathbb{N}, \\ \quad\;\; = 0 \text{ sinon}, \\ C_{ij} = 1 \text{ si } j = 2i, \\ \quad\;\; = 0 \text{ sinon}, \end{cases}$$

avec $\epsilon = 0.67$ et $\delta = 0.00195$. Les vecteurs constants u_d et z_d sont donnés par
$u_d^T = [1075,825,840,842,845,850,858,871,893,930,958,1483]$ calories/min
et $z_d^T = [30,30,30,30,30,30,30,30,30,30,30,30]$ degré Celsius.
Nous présentons la commande optimale en fonction du nombre d'itérations et ceci en appliquant les deux schémas itératifs de Gauss Seidel : le premier en intégrant l'état et l'état adjoint dans le même sens et le deuxième en intégrant l'état dans le sens direct et l'état adjoint dans le sens inverse. L'interval de temps discritisé est fixée à 180 min avec un pas de discrétisation égale a 3 min et $k = 0.01$.

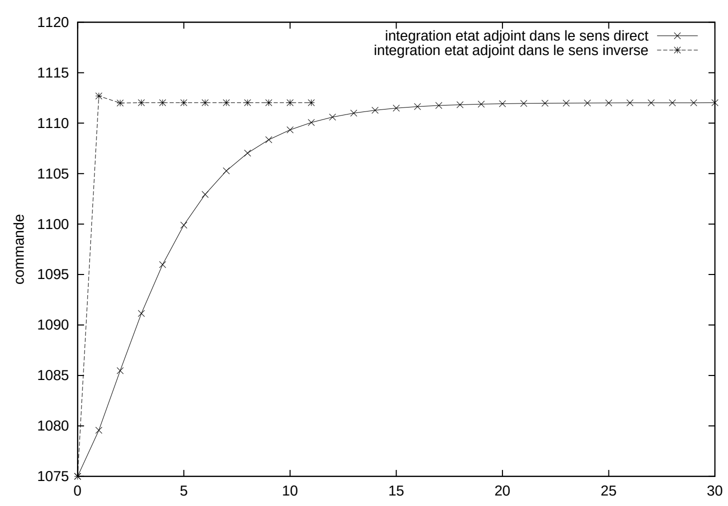

FIG. 2.2 – *Commande optimale calculé en séquentiel en fonction du nombre d'itérations*

D'après la figure 2.2, l'intégration de l'état adjoint dans le sens inverse permet logiquement d'accélérer le schéma itératif de Gauss Seidel et l'algorithme converge avec un nombre d'itérations très réduit comparé à celui obtenu en intégrant l'état adjoint dans le même sens que l'état.

2.4 Schémas itératifs parallèles asynchrones

L'étude des algorithmes itératifs asynchrones pour la résolution d'un problème de commande optimale a été proposée dans [Mie72] et [Spi84] dans le cadre d'une application de point fixe contractante et dans [Jac76b] et [Jac77] dans le cadre de l'ordre partiel. Nous présentons brièvement les principaux résultats de convergence des algorithmes asynchrones et nous donnons par la suite un nouveau résultat de convergence des algorithmes itératifs asynchrones avec communication flexible.

2.4.1 Cas d'une application de point fixe contractante

En guise d'introduction aux aspects importants relatifs à la convergence des itérations asynchrones, nous rappelons dans ce sous-paragraphe un résultat important établi par Spiteri (cf. [Spi84]) dans le cas d'une application de point fixe contractante.

2.4.1.1 Décomposition par point du problème

Les notations suivantes ont été introduites par J.C. Miellou (cf. [Mie75a]). Soit E un espace de BANACH réflexif et β un entier naturel. Soit E_l, $l \in \{1,...,\beta\}$ une famille d'espaces de BANACH réflexifs telle que $E = \Pi_{l=1}^{\beta} E_l$. Un élément X de E est un vecteur ayant pour composantes $X = \{x_1,...,x_l,...,x_\beta\}$ avec $x_l \in E_l$. On note par $|E_l|$ la norme de E_l et soit q la norme canonique de E définie comme suit : $q(X) = \{|x_1|,...,|x_\beta|\}$. On pose $\beta = 4n + m$ et on considère la matrice $\mathcal{A}'$ et le vecteur $\mathcal{B}'$ définis par bloc de la manière suivante :

$$\mathcal{A}' = \begin{pmatrix} A & 0 & -B \\ C^*C & A^* & 0 \\ 0 & \frac{1}{k}B^* & I_m \end{pmatrix}, \mathcal{B}' = \begin{pmatrix} 0 \\ -C^*z_d \\ u_d \end{pmatrix},$$

où I_m désigne la matrice identité dans $\mathbb{R}^m$. On pose $X(t) = [y(t),p(t),u(t)]^T$, $X \in E$ et on se place dans le cas où l'on a Λ un opérateur affine défini par : $\Lambda(X) = \mathcal{A}'X + \mathcal{B}'$ et Λ^d l'opérateur diagonal défini par $\Lambda^d = \{\frac{d}{dt}, -\frac{d}{dt},\partial\Psi_{U_{ad}}\}$. Avec ces notations, le problème (2.5) aura la forme :

$$0 \in \Lambda^d(X) + \Lambda(X) \tag{2.17}$$

2.4.1.2 Décomposition par bloc du problème

Soit α un entier naturel tel que $\alpha \leq \beta$ et $\{\beta_i\}$ pour $i \in \{1,...,\alpha\}$ une famille d'entiers tels que :

$$\begin{cases} \displaystyle\sum_{i=1}^{\alpha} \beta_i = \beta, \\ \beta_i \neq 0. \end{cases} \tag{2.18}$$

Soit pour $i \in \{1,...,\alpha\}$, $\alpha_i = \sum_{j=1}^{i-1} \beta_j$ et $E_i = \Pi_{l=\alpha_i+1}^{\alpha_{(i+1)}} E_l$ avec $E = \Pi_{i=1}^{\alpha} E_i$. Soit $X \in E$, on peut définir les blocs-composantes de $X = [X_1,...,X_\alpha] \in \Pi_{i=1}^{\alpha} E_i$. Soit $q(X)$ la norme vectorielle canonique définie sur E par $q(X) = [|X_1|,...,|X_\alpha|]$ où $|.|_i$ est la norme sur E_i.
L'opérateur affine Λ défini précédemment peut s'exprimer en fonction des blocs-composantes X_i de la manière suivante : $\Lambda(X) = [\Lambda_1(X),...,\Lambda_\alpha(X)]^T$. On définit de même l'opérateur diagonal $\Lambda^d(X) = [\Lambda_1^d(X),...,\Lambda_\alpha^d]^T$. Le problème aura toujours la forme de l'équation (2.17).

2.4.1.3 Résultats de convergence de la méthode de l'état adjoint avec échange asynchrone

Les résultats de convergence des itérations asynchrones pour la résolution du problème de commande optimale par la méthode de l'état adjoint par relaxation ont été établis dans [Spi84] pour une décomposition par bloc avec $\alpha = 3$, soit pour $t \in [t_0,T]$:

$$\begin{cases} X_1 = y(t) \\ X_2 = p(t) \\ X_3 = u(t) \end{cases} \tag{2.19}$$

Compte tenu de la décomposition en trois blocs et de manière analogue à ce qui a été écrit au paragraphe précédent, les opérateurs Λ_i, pour $i \in \{1,2,3\}$, peuvent s'écrire de la manière suivante :

$$\begin{cases} \Lambda_1(X(t)) = AX_1(t) - BX_3(t) \\ \Lambda_2(X(t)) = -C^*CX_1(t) + A^*X_2(t) - C^*z_d \\ \Lambda_3(X(t)) = B^*X_2(t) + k(X_3(t) - u_d) \end{cases} \tag{2.20}$$

On considère la matrice Z de coefficients z_{ij} tels que, pour tous $i,j \in \{1,...,\beta\}$:

$$\begin{cases} z_{ij} = -|\mathcal{A}'_{ij}| \text{ si } i \neq j \\ z_{ij} = \mathcal{A}'_{ij} \text{ sinon} \end{cases}$$

Proposition 2.4.1 *(cf. [Spi84], Proposition 3.3) Z est une minorante Z-accrétive (cf. Définition A.3.1 de l'annexe) de l'opérateur Λ sur E.*

Proposition 2.4.2 *(cf. [Spi84], Proposition 3.4) Si A est une M-matrice, alors il existe une valeur $k_0 > 0$ telle que pour $k > k_0$, Z est une M-matrice.*

Proposition 2.4.3 *(cf. [Spi84], Proposition 3.6) Pour $k > k_0$, les itérations asynchrones pour une décomposition par point ou par bloc convergent vers la solution X^* quelle que soit la donnée initiale $X^0 \in E$.*

Ces résultats ont été établis en utilisant la propriété de H-accrétivité de l'opérateur Λ (cf. [Spi84] et Définitions de la section A.3 de l'annexe).

On présente par la suite un autre type de schéma itératif qui peut être mis en œuvre sur un grand nombre de processeurs et dont on donne un résultat de convergence pour une mise en œuvre asynchrone avec communication flexible.

2.4.2 Cas d'une application de point fixe monotone

Le système (2.10) peut se transformer en équation de point fixe Linéaire (cf. [Jac77]):

$$\begin{pmatrix} Y \\ P \end{pmatrix} = \mathcal{A} \begin{pmatrix} Y \\ P \end{pmatrix} + \mathcal{B},$$

$\mathcal{A}$ peut être décomposée en deux matrices non négatives $\mathcal{A}_1$ et $\mathcal{A}_2$ telles que : $\mathcal{A} = \mathcal{A}_1 - \mathcal{A}_2$ et $\rho(\mathcal{A}_1 + \mathcal{A}_2) < 1$ (cf. [Jac76b] et [Jac77]).
Considérons les deux suites $\{U^{(r)} = (\check{Y}^{(r)}, \check{P}^{(r)})^t\}$ et $\{V^{(r)} = (\hat{Y}^{(r)}, \hat{P}^{(r)})^t\}$ qui sont respectivement définies par :

$$\begin{cases} U^{(r+1)} = \mathcal{A}_1 U^{(r)} - \mathcal{A}_2 V^{(r)} + \mathcal{B}, \\ V^{(r+1)} = \mathcal{A}_1 V^{(r)} - \mathcal{A}_2 U^{(r)} + \mathcal{B}. \end{cases} \tag{2.22}$$

Il a été démontré dans [Jac77] que s'il existe une sous-solution et une sur solution de départ $U^{(0)}$ et $V^{(0)}$ vérifiant les inégalités suivantes:

$$\begin{cases} U^{(0)} \leq V^{(0)}, \\ U^{(0)} \leq \mathcal{A}_1 U^{(0)} - \mathcal{A}_2 V^{(0)} + \mathcal{B}, \\ V^{(0)} \geq \mathcal{A}_1 V^{(0)} - \mathcal{A}_2 U^{(0)} + \mathcal{B}, \end{cases}$$

alors : $U^{(0)} \leq ... \leq U^{(m)} \leq ... \leq U^* \leq V^* \leq ... \leq V^{(m)} \leq ... \leq V^{(0)}$.

2.4.2.1 Sous-solution et sur solution de départ

Le choix d'un vecteur constant dans le temps comme sous-solution ou sur solution transforme le système (2.9) en un autre système plus réduit et ceci en annulant le terme qui exprime la dérivée partielle de l'état et de l'état adjoint :

$$\begin{cases} Ay(t) + \dfrac{1}{k} BB^* p(t) = Bu_d \\ A^* p(t) - C^* C y = -C^* z_d \end{cases} \tag{2.24}$$

Soit H la matrice du système (2.24):

$$H = \begin{pmatrix} A & \frac{1}{k} BB^* \\ -C^* C & A^* \end{pmatrix}.$$

Une méthode de détermination d'une sous-solution et d'une sur solution de départ a été proposée dans [KVZ72]. Considérons la matrice de Jacobi J_H de H et soit z_0 un point intérieur du cne K défini sur J_H, alors il existe $q \in \mathbb{N}$ tel que $J_H^q z_0 \leq \alpha z_0$, avec $\alpha < 1$. Le vecteur z_1 calculé à partir de z_0 de la manière suivante :

$$z_1 = \alpha^{1-\frac{1}{q}} z_0 + \alpha^{1-\frac{2}{q}} z_0 + ... + \alpha^{q-1} z_0,$$

satisfait l'inégalité suivante : $J_H z_1 \leq \alpha^{1/q} z_1$.
Le point z_1 est intérieur du cne K, donc il existe $b > 0$ (cf. [KVZ72]) tel que:

$$-b(1 - \alpha^{\frac{1}{q}} z_1) \leq f \leq b(1 - \alpha^{\frac{1}{q}} z_1),$$

où $f^T = (Bu_d, - C^* z_d)$. On pose $u_0 = -bz_1$ et $v_0 = bz_1$, alors $J_H u_0 + f \geq -bz_1 = u_0$ et $J_H v_0 + f \leq bz_1 = v_0$.

Algorithme 6 : Détermination d'une sous-solution et d'une sur solution de départ

$q = 0$
rpter
  $\alpha = Max_i \frac{(J_H^q z_0)_i}{(z_0)_i}$
  $q = q + 1$
jusqu' $(\alpha < 1)$
$z_1 = \alpha^{(1-1/q)} z_0$
$i = 1$
rpter
  $i = i + 1$
  $z_1 = z_1 + \alpha^{-1/q} z_1$
jusqu' $(i = q)$
$b = Max_i \frac{|f_i|}{(1-\alpha^{1/q})|(z_1)_i|}$
$u_0 = -bz_1$
$v_0 = bz_1$

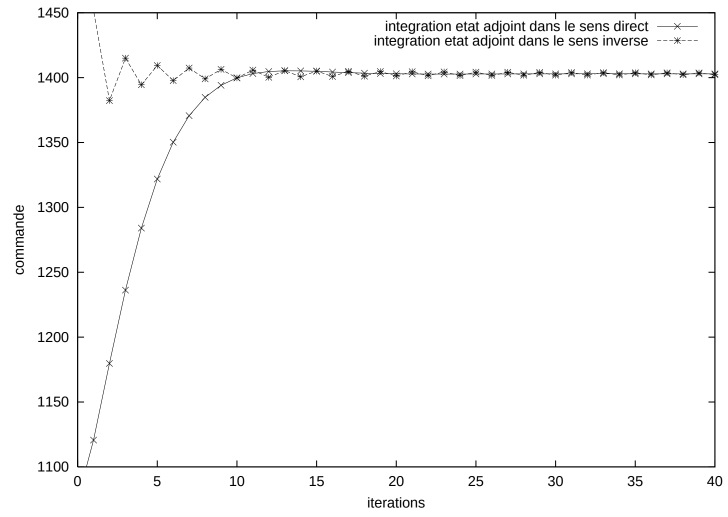

Fig. 2.3 – *Cas d'une application de point fixe contractante*

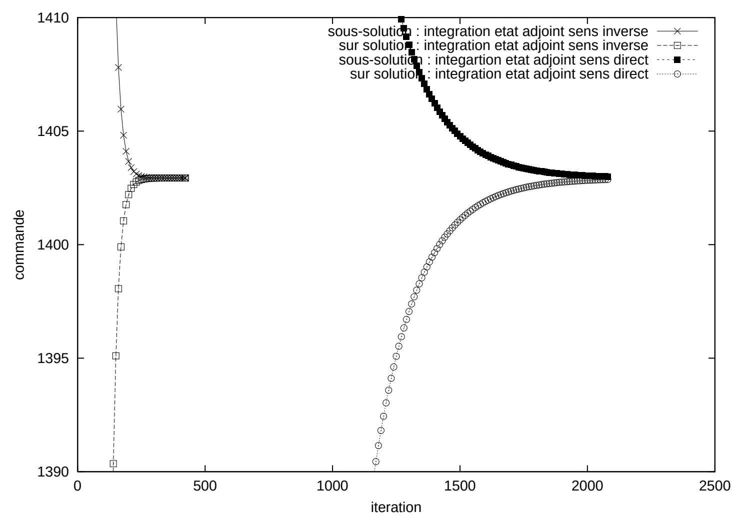

FIG. 2.4 – *Cas d'une application de point fixe qui engendre deux séquences monotones de sous-solutions et de sur solutions*

L'algorithme 6 permet de déterminer une sous-solution et une sur solution de départ.

Les figures 2.3 et 2.4 montrent la valeur de la première composante de la commande en fonction du nombre d'itérations respectivement pour le cas d'une application de point fixe contractante et le cas d'une application de point fixe monotone. Dans le cas monotone, les résultats montrent que l'intégration dans le sens inverse conduit à une résolution plus rapide.

2.4.2.2 Un exemple d'application de point fixe monotone

Soit $i \in \{1,...N\}$, $t_i = t_{i-1} + h$, $y(t_i) = y_i$ et $p(t_i) = p_i$. Le système (2.9) peut s'écrire sous la forme :

$$\begin{cases} (hA + I)y_i = y_{i-1} - \dfrac{h}{k} BB^* p_i + hBu_d, \\ (hA^* + I)p_i = p_{i+1} + C^*Cy_i - C^*z_d, \end{cases} \tag{2.27}$$

où y_i et p_i sont respectivement le i-ème bloc du vecteur état et du vecteur état adjoint. On note aussi par $y_{i,j}$ et $p_{i,j}$ respectivement la j-ème composante du i-ème bloc du vecteur état et du vecteur état adjoint. La j-ème composante du i-ème bloc du vecteur état et du vecteur état adjoint s'expriment alors par:

$$\begin{cases} (ha_{jj} + 1)y_{i,j} + \displaystyle\sum_{l=1, l\neq j}^{2n} ha_{jl} y_{i,l} = y_{i-1,j} - \frac{h}{k} \sum_{l=1}^{2n} (BB^*)_{jl} p_{i,l} + h(Bu_d)_j, \\ (ha^*_{jj} + 1)p_{i,j} + \displaystyle\sum_{l=1, l\neq j}^{2n} ha^*_{jl} p_{i,l} = p_{i+1,j} + h \sum_{l=1}^{2n} (C^*C)_{jl} y_{i,l} - h(C^*z_d)_j, \end{cases} \tag{2.28}$$

On peut introduire une application de point fixe F définie par : pour $q \in \{1,...,2nN\}$, avec $q = 2n(i-1)+j$,

$$F_q\begin{pmatrix} Y \\ P \end{pmatrix} = \frac{1}{ha_{jj}+1}[-\sum_{l=1,l\neq j}^{2n} ha_{jl}y_{i,l} + y_{i-1,j} - \sum_{l=1}^{2n}\frac{h}{k}(BB^*)_{jl}p_{i,l} + h(Bu_d)_j],$$

et pour $q \in \{2nN+1,...,4nN\}$, avec $q = 2n(i-1)+j+2nN$,

$$F_q\begin{pmatrix} Y \\ P \end{pmatrix} = \frac{1}{ha^*_{jj}+1}[-\sum_{l=1,l\neq j}^{2n} ha^*_{jl}p_{i,l} + p_{i+1,j} + \sum_{l=1}^{2n} h(C^*C)_{jl}y_{i,l} - h(C^*z_d)_j].$$

Soit $U^{(r)} = (\check{Y}^{(r)},\check{P}^{(r)})$ une sous-solution et $V^{(r)} = (\hat{Y}^{(r)},\hat{P}^{(r)})$ une sur solution à l'itération (r). Le schéma itératif de Gauss Seidel engendré par l'application de point fixe F à l'itération (s) du second niveau est le suivant :

$$\begin{aligned}
\check{y}^{(r,s)}_{i,j} &= \frac{1}{ha_{jj}+1}[-\sum_{l=1}^{j-1} ha_{jl}\check{y}^{(r,s)}_{i,l} - \sum_{l=j+1}^{2n} ha_{jl}\check{y}^{(r,s-1)}_{i,l} + \check{y}^{(r)}_{i-1,j} - \sum_{l=1}^{2n}\frac{h}{k}(BB^*)_{jl}\hat{p}^{(r-1)}_{i,l} + h(Bu_d)_j],\\
\check{p}^{(r,s)}_{i,j} &= \frac{1}{ha^*_{jj}+1}[-\sum_{l=1}^{j-1} ha^*_{jl}\check{p}^{(r,s)}_{i,l} - \sum_{l=j+1}^{2n} ha^*_{jl}\check{p}^{(r,s-1)}_{i,l} + \check{p}^{(r)}_{i+1,j} + \sum_{l=1}^{2n} h(C^*C)_{jl}\check{y}^{(r)}_{i,l} - h(C^*z_d)_j],\\
\hat{y}^{(r,s)}_{i,j} &= \frac{1}{ha_{jj}+1}[-\sum_{l=1}^{j-1} ha_{jl}\hat{y}^{(r,s)}_{i,l} - \sum_{l=j+1}^{2n} ha_{jl}\hat{y}^{(r,s-1)}_{i,l} + \hat{y}^{(r)}_{i-1,j} - \sum_{l=1}^{2n}\frac{h}{k}(BB^*)_{jl}\check{p}^{(r-1)}_{i,l} + h(Bu_d)_j],\\
\hat{p}^{(r,s)}_{i,j} &= \frac{1}{ha^*_{jj}+1}[-\sum_{l=1}^{j-1} ha^*_{jl}\hat{p}^{(r,s)}_{i,l} - \sum_{l=j+1}^{2n} ha^*_{jl}\hat{p}^{(r,s-1)}_{i,l} + \hat{p}^{(r)}_{i+1,j} + \sum_{l=1}^{2n} h(C^*C)_{jl}\hat{y}^{(r)}_{i,l} - h(C^*z_d)_j].
\end{aligned} \tag{2.29}$$

2.4.2.3 Un résultat de convergence des itérations asynchrones avec communication flexible

Nous proposons dans ce paragraphe un résultat de convergence des itérations parallèles asynchrones avec communication flexible pour le schéma à un niveau. La définition des itérations asynchrones avec communication flexible a été présentée au chapitre I.1.
Soit $\mathcal{B}$ un vecteur de $\mathbb{R}^{4nN}$ et $\mathcal{A}$ une matrice de dimension $(4nN,4nN)$ telle que $\mathcal{A} = \mathcal{A}_1 - \mathcal{A}_2$ où $\mathcal{A}_1$ et $\mathcal{A}_2$ sont deux matrices non négatives telles que $\rho(\mathcal{A}_1+\mathcal{A}_2) < 1$. Alors le système $X = \mathcal{A}_1X - \mathcal{A}_2X + \mathcal{B}$ admet une solution unique (cf. [OR70]) et il existe un couple de sous-solution et sur solution notées respectivement $U^{(0)}$ et $V^{(0)}$ qui satisfont les inégalités suivantes :

$$\begin{cases} U^{(0)} \leq V^{(0)} \\ U^{(0)} \leq \mathcal{A}_1U^{(0)} - \mathcal{A}_2V^{(0)} + \mathcal{B} \\ V^{(0)} \geq \mathcal{A}_1V^{(0)} - \mathcal{A}_2U^{(0)} + \mathcal{B} \end{cases} \tag{2.30}$$

Proposition 2.4.4 *Sous les hypothèses de ce sous-paragraphe, le schéma itératif parallèle asynchrone avec communication flexible converge d'une manière monotone vers la solution* $U^* = V^*$ *vérifiant :*
$U^{(0)} \leq U^{(1)} \leq ... \leq U^{(k)} \leq ... \leq U^* = V^* \leq ... \leq V^{(k)} \leq ... \leq V^{(1)} \leq V^{(0)}$

Démonstration :
Considérons l'application de point fixe G de $\mathbb{R}^{8nN}$ dans $\mathbb{R}^{4nN}$ définie par :

$$G(U,V) = \mathcal{A}_1 U - \mathcal{A}_2 V + \mathcal{B}.$$

On a clairement d'après (2.30) que :

$$\begin{cases} U^{(0)} \leq V^{(0)}, \\ U^{(0)} \leq G(U^{(0)},V^{(0)}), \\ V^{(0)} \geq G(V^{(0)},U^{(0)}). \end{cases} \tag{2.31}$$

Nous définissons une application G' de $\mathbb{R}^{8nN}$ dans $\mathbb{R}^{8nN}$ à partir de G par :

$$G'(U,V) = \begin{pmatrix} G'_1(U,V) \\ G'_2(U,V) \end{pmatrix} = \begin{pmatrix} G(U,V) \\ G(V,U) \end{pmatrix}$$

Nous allons appliquer les itérations asynchrones avec communication flexible à la recherche d'un point fixe de G'.

Prenons $r = 0$, d'après la définition des itérations asynchrones avec communication flexible il résulte clairement de (2.31) que :

si $i \in s(0)$ avec $i \in \{1,...,2N\}$, alors
$U_i^{(1)} = G_i(U^{(0)},V^{(0)}) \geq U_i^{(0)}$,

si $i \in s(0)$ avec $i \in \{2N+1,...,4N\}$, alors
$V_{i-2N}^{(1)} = G_i(V^{(0)},U^{(0)}) \leq V_{i-2N}^{(0)}$,

si $i \notin s(0)$ avec $i \in \{1,...,2N\}$ alors
$U_i^{(1)} = U_i^{(0)}$,

si $i \notin s(0)$ avec $i \in \{2N+1,...,4N\}$ alors
$V_{i-2N}^{(1)} = V_{i-2N}^{(0)}$.

On a donc : $U^{(1)} \geq U^{(0)}$ et $V^{(1)} \leq V^{(0)}$.

Comme $\mathcal{A}_1$ et $\mathcal{A}_2$ sont deux matrices non négatives, il résulte de (2.30) et de la définition des itérations asynchrones avec communication flexible que :

si $i \in s(0)$, avec $i \in \{1,...,2N\}$, alors
$U_i^{(1)} = [\mathcal{A}_1 U^{(0)} - \mathcal{A}_2 V^{(0)} + \mathcal{B}]_i \leq [\mathcal{A}_1 V^{(0)} - \mathcal{A}_2 U^{(0)} + \mathcal{B}]_i \leq V_i^{(0)} = V_i^{(1)}$,

si $i \in s(0)$, avec $i \in \{2N+1,...,4N\}$, alors
$V_{i-2N}^{(1)} = [\mathcal{A}_1 V^{(0)} - \mathcal{A}_2 U^{(0)} + \mathcal{B}]_{i-2N} \geq [\mathcal{A}_1 U^{(0)} - \mathcal{A}_2 V^{(0)} + \mathcal{B}]_{i-2N} \geq U_{i-2N}^{(0)} = U_{i-2N}^{(1)}$,

Nous avons donc $U^{(1)} \leq V^{(1)}$, d'où $U^{(0)} \leq U^{(1)} \leq V^{(1)} \leq V^{(0)}$.

Supposons qu'au rang r, on a:

$$U^{(0)} \leq U^{(1)} \leq ... \leq U^{(r)} \leq V^{(r)} \leq ... \leq V^{(1)} \leq V^{(0)}. \tag{2.32}$$

Nous démontrons maintenant que ces inégalités sont vraies au rang $(r+1)$.
Il résulte de la définition des itérations asynchrones avec communication flexible que :

si $i \in s(r)$ avec $i \in \{1,...,2N\}$, alors
$U_i^{(r+1)} = G_i(\dot{U}^{(r)},\dot{V}^{(r)})$,

si $i \in s(r)$ avec $i \in \{2N+1,...,4N\}$, alors
$V_{i-2N}^{(r+1)} = G_i(\dot{V}^{(r)},\dot{U}^{(r)})$,

si $i \notin s(r)$ avec $i \in \{1,...,2N\}$ alors
$U_i^{(r+1)} = U_i^{(r)}$,

si $i \notin s(r)$ avec $i \in \{2N+1,...,4N\}$ alors
$V_{i-2N}^{(r+1)} = V_{i-2N}^{(r)}$,
avec

$$\dot{U}_l^{(r+1)} \in < max\{U_l^{(\tau_l(r))},\dot{U}_l^{(q)}\},U_l^{(r)} >_l ,l \in \{1,...,2N\},\dot{V}_l^{(r+1)} \in < V_l^{(r)},max\{V_l^{(\tau_l(r))},\dot{V}_l^{(q')}\} >_l ,l \in \{1,...,2N\}. \tag{2.33}$$

Il résulte de (2.32), (2.33), du fait que $\mathcal{A}_1$ et $\mathcal{A}_2$ sont deux matrices non négatives et de la définition des itérations asynchrones avec communication fleixble que

si $i \in \{1,...,2N\}$, alors
$U_i^{(r+1)} \geq U_i^{(r)}$,

si $i \in \{2N+1,...,4N\}$, alors
$V_{i-2N}^{(r+1)} \leq V_{i-2N}^{(r)}$,

donc $U^{(r+1)} \geq U^{(r)}$ et $V^{(r+1)} \geq V^{(r)}$. D'autre part,

si $i \in \{1,...,2N\}$, alors en utilisant (2.32)
$U_i^{(r+1)} = [\mathcal{A}_1 \dot{U}^{(r)} - \mathcal{A}_2 \dot{V}^{(r)} + \mathcal{B}]_i \leq [\mathcal{A}_1 \dot{V}^{(q')} - \mathcal{A}_2 \dot{U}^{(q')} + \mathcal{B}]_i = V_i^{(r)} = V_i^{(r+1)}$,

si $i \in \{2N+1,...,4N\}$, alors

$$V_{i-2N}^{(r+1)} = [\mathcal{A}_1\dot{V}^{(r)} - \mathcal{A}_2\dot{U}^{(r)} + \mathcal{B}]_{i-2N} \geq [\mathcal{A}_1\dot{U}^{(q)} - \mathcal{A}_2\dot{V}^{(q)} + \mathcal{B}]_{i-2N} = U_{i-2N}^{(r)} = U_{i-2N}^{(r+1)},$$

donc $U^{(r+1)} \leq V^{(r+1)}$.

Toutes les inégalités sont démontrées au rang $r+1$, par conséquent les suites monotones $\{U^{(r)}\}$ et $\{V^{(r)}\}$ ont respectivement pour limite U^* et V^* telles que $U^* \leq V^*$.
D'autre part d'après la continuité de l'opérateur G, on a :

$$G(U^*,V^*) = \mathcal{A}_1U^* - \mathcal{A}_2V^* + \mathcal{B} = U^*, G(V^*,U^*) = \mathcal{A}_1V^* - \mathcal{A}_2U^* + \mathcal{B} = V^*. \tag{2.34}$$

En faisant la soustraction membre à membre des égalités de (2.34), nous obtenons $U^*-V^* = (\mathcal{A}_1 + \mathcal{A}_2)(U^* - V^*)$. $\mathcal{A}_1$ et $\mathcal{A}_2$ étant deux matrice non négatives avec $\rho(\mathcal{A}_1 + \mathcal{A}_2) < 1$, alors le système $(\mathcal{A}_1 + \mathcal{A}_2)X = 0$ admet une solution unique $X = 0$, donc $U^* - V^* = 0$ ou encore $U^* = V^*$.

2.4.2.4 Terminaison des itérations asynchrones avec communication flexible

A l'itération (r), la solution est estimée à une valeur calculée en fonction de la sous-solution et de la sur solution, soit $X^{(r)} = \frac{U^{(r)}+V^{(r)}}{2}$ avec une erreur maximale $e^{(r)} = \frac{V^{(r)}-U^{(r)}}{2}$. La terminaison est établie quand $||e^{(r)}||_\infty < \epsilon$, ϵ étant la précision de terminaison. La mise à jour du vecteur itéré par les différents processeurs fait croître la suite de vecteurs sous-solutions $U^{(r)}$ et fait décroître la suite de vecteurs sur solutions $V^{(m)}$ et par suite elle fait décroître l'erreur $e^{(r)}$. Dans un système distribué, chaque processeur calcule l'erreur locale qui décroît toujours quelque soit les données reues des autres processeurs. Nous proposons un algorithme de terminaison basé sur la circulation d'un jeton entre les processeurs. Nous fixons une relation de parenté entre les processeurs selon la figure 2.5. Le processeur 0 est le père du processeur 1 qui est lui même le père du processeur 2, etc..., le processeur $\alpha - 1$ est le noeud de l'arbre qui n'admet pas de processeurs fils et le processeur 0 est la racine de l'arbre qui n'admet pas de processeur père. Initialement le jeton est possédé par le processeur $\alpha - 1$. Quand ce processeur détecte sa terminaison locale ($||e_{\alpha-1}||_\infty < \epsilon$), il envoie le jeton à son père et de proche en proche jusqu'à ce que le jeton arrive au processeur racine qui envoie un message de terminaison globale à tous les processeurs. Il résulte clairement de la propriété de monotonie des suites $\{U^{(r)}\}$ et $\{V^{(r)}\}$ que cette méthode détecte la terminaison en un temps fini lorsque celle-ci s'est produite.

2.5 Etude de performance

Dans ce paragraphe nous présentons la mise en œuvre parallèle sur plusieurs types d'architecture et nous comparons les performances des itérations parallèles synchrones,

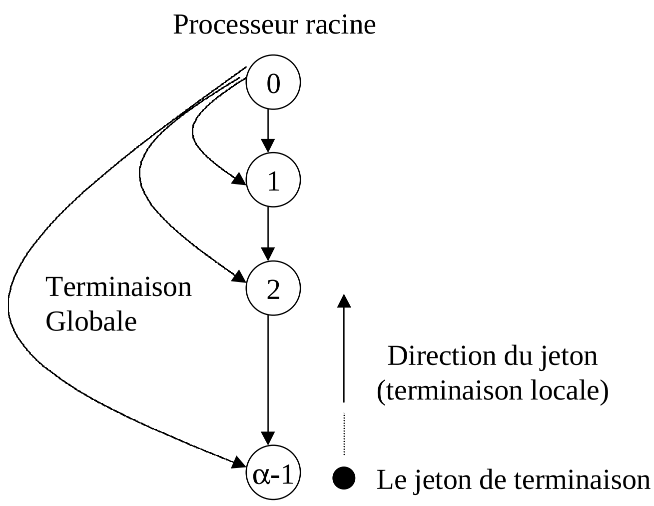

FIG. 2.5 – *algorithme de terminaison*

asynchrones et asynchrones avec communication flexible en fonction des caractéristiques matérielles et logicielles de chaque architecture.

2.5.1 Décomposition en sous-domaines du problème

La décomposition en sous-domaines consiste à répartir les blocs sur les différents processeurs. Chacun des processeurs met en œuvre le schéma itératif sur un ensemble de blocs appelé sous-domaine. Selon les équations du système (2.9), le schéma itératif de Gauss Seidel peut être exprimé sous la forme :

$$\begin{cases} (hA+I)y_i^{(r+1)} = y_{i-1}^{(r+1)} - \dfrac{h}{k}BB^*p_i^{(r)} + hBu_d, \\ (hA^*+I)p_i^{(r+1)} = p_{i+1}^{(r+1)} + C^*Cy_i^{(r+1)} - C^*z_d. \end{cases} \tag{2.35}$$

Les blocs-composantes du vecteur état sont exprimés en fonction des blocs-composantes du vecteur état adjoint de même indice. A l'itération $(r+1)$ la dépendance entre les vecteurs état et état adjoint se présente de la manière suivante:.

$$...y_{i-1}^{(r+1)} \to y_i^{(r+1)} \to y_{i+1}^{(r+1)} \to p_{i+1}^{(r+1)} \leftarrow p_i^{(r+1)} \leftarrow p_{i-1}^{(r+1)}...$$

Il est plus intéressant dans ce cas de répartir d'une faon symétrique les blocs-composantes des vecteurs état et état adjoint sur les différents processeurs afin de minimiser l'interaction entre les blocs-composantes situés sur des processeurs différents. La figure (2.6) montre une décomposition des blocs-composantes des vecteurs état et état adjoint pour deux itérations (r) et $(r+1)$.

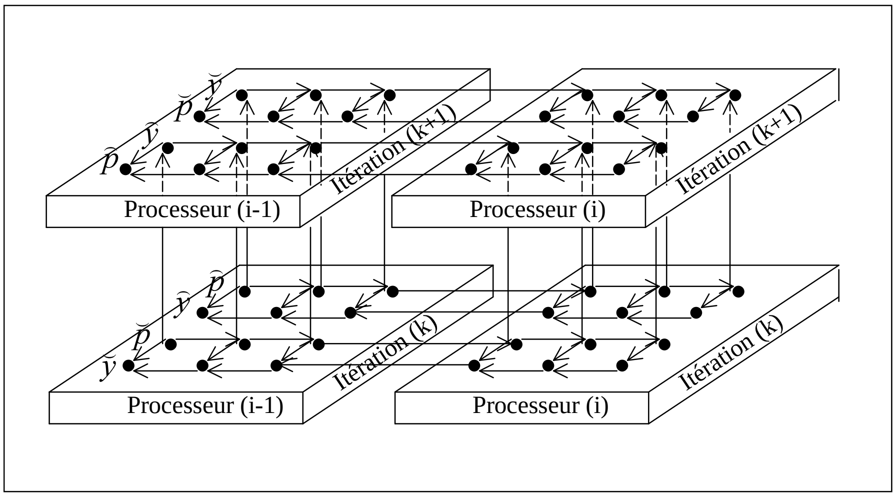

FIG. 2.6 – *Décomposition en sous-domaine*

2.5.2 Etude de performance sur CRAYT3E

La mise en œuvre sur le CRAYT3E a été effectuée comme dans le chapitre II.1 en utilisant la bibliothèque MPI dans le cas synchrone et SHMEM dans le cas asynchrone. La mise en œuvre des itérations asynchrones avec communication flexible est basée sur l'utilisation de la fonction get() de SHMEM qui est la meilleure politique de communication observée conformément aux observations du chapitre II.1. Cette fonction permet d'accéder à la valeur courante du vecteur itéré. Dans le cas asynchrone, nous utilisons la fonction put() de SHMEM qui permet après une itération d'envoyer les valeurs des blocs-composantes frontières aux autres processeurs. Les fonctions echange_données_mpi(), echange_données_put() et echange_données_get() permettent respectivement de définir l'échange de données entre les différents processeurs pour les mise en œuvre synchrone, asynchrone et asynchrone avec communication flexible. Les valeurs de début_bloc et fin_bloc délimitent le domaine sur lequel travaille le processeur courant noté par processus_courant.

Fonction : echange_données_mpi(Z,j)
Si (j = debut_bloc) **Alors**
  Si (processus_courant <> nombre_processus-1) **Alors**
    MPI_Isend($Z_{(fin\_bloc)}$,processus_courant+1)
  Fin Si
  Si (processus_courant <> 0) **Alors**
    MPI_Recv($Z_{(debut\_bloc-1)}$,processus_courant-1)
  Fin Si
Fin Si
Si (j = fin_bloc) **Alors**
  Si (processus_courant <> 0) **Alors**

MPI_Isend($Z_{(debut\_bloc)}$,processus_courant-1)
Fin Si
Si (processus_courant <> nombre_processus-1) **Alors**
MPI_Recv($Z_{(fin\_bloc+1)}$,processus_courant+1)
Fin Si
Fin Si
Fin Fonction

Fonction : echange_données_put(Z,j)
Si (j = debut_bloc) **Alors**
Si (processus_courant <> nombre_processus-1) **Alors**
shmem_put($Z_{(fin\_bloc)}$,processus_courant+1)
Fin Si
Fin Si
Si (j = fin_bloc) **Alors**
Si (processus_courant <> 0) **Alors**
shmem_put($Z_{(debut\_bloc)}$,processus_courant-1)
Fin Si
Fin Si
Fin Fonction

Fonction : echange_données_get(Z,j)
Si (j = debut_bloc) **Alors**
Si (processus_courant <> 0) **Alors**
shmem_get($Z_{(debut\_bloc-1)}$,processus_courant-1)
Fin Si
Fin Si
Si (j = fin_bloc) **Alors**
Si (processus_courant <> nombre_processus-1) **Alors**
shmem_get($Z_{(fin\_bloc+1)}$,processus_courant+1)
Fin Si
Fin Si
Fin Fonction

L'algorithme 7 met en œuvre en parallèle le schéma itératif à deux niveaux en intégrant l'état dans le sens direct et l'état adjoint dans le sens inverse. La fonction échange_données() remplace l'une des trois fonctions citées ci-dessus.

Les figures 2.7 et 2.8 montrent respectivement les temps d'exécution et les accélérations des mises en œuvre, synchrone, asynchrone et asynchrone avec communication flexible en fonction du nombre de processeurs. L 'intervalle de temps discrétisé correspond à 180 minutes avec un pas de discrétisation égal à 18 sec, soit un nombre de blocs égal à 1800, par conséquent la dimension du système est égale à 86400. Les précisions du premier et du second niveau sont égales respectivement à $\epsilon_1 = 10^{-4}$ et $\epsilon_2 = 10^{-9}$, $k = 0.0012$. Le

Algorithme 7 Schéma itératif à deux niveaux distribué

rpter
  pour j=debut_bloc,j=fin_bloc **faire**
    échange_données($\check{Y}$,j)
    échange_données($\hat{Y}$,j)
    calcul_deuxième_niveau($\check{Y}$)
    calcul_deuxième_niveau($\hat{Y}$)
  fin pour
  pour j=fin_bloc,j=debut_bloc **faire**
    échange_données($\check{P}$,j)
    échange_données($\hat{P}$,j)
    calcul_deuxième_niveau($\check{P}$)
    calul_deuxième_niveau($\hat{P}$)
  fin pour
jusqu' $(||\frac{\hat{Y}-\check{Y}}{2}||_\infty < \epsilon_1)$ et $||\frac{\hat{P}-\check{P}}{2}||_\infty < \epsilon_1)$

tableau 2.1 montre le nombre d'itérés partiels moyen par itération pour une mise en œuvre synchrone sur 4 processeurs du CRAYT3E.

Selon les figures 2.7 et 2.8, la mise en œuvre asynchrone avec communication flexible est plus performante que les autres mises en œuvre parallèles. La mise en œuvre asynchrone est moins performante que la mise en œuvre synchrone. Ceci est d au fait que les charges de calcul sont très équilibrées pour ce type de problème (cf. Table 2.1). De plus, la synchronisation des processeurs en faisant une réception bloquante, permet d'avoir un taux de convergence meilleur comparé a la mise en œuvre asynchrone où les données envoyées via shmem_put() peuvent arriver en retard et font chuter le taux de convergence. L'utilisation de la fonction shmem_get() permet d'accéder directement à la valeur courante du vecteur itéré réactualisé par les autres processeurs avant de faire une nouvelle réactualisation. Cette faon de faire permet d'une part d'avoir la valeur la plus fraîche, qui est à cet instant là, la valeur la plus proche de la solution et d'autre part d'avoir un plus faible temps de latence comparé au primitives MPI_Isend et MPI_Recv() de MPI (cf. chapitre II.1).

2.5.3 Etude de performance sur SMP

La mise en œuvre sur une SMP a été effectuée en utilisant les threads POSIX. Ces derniers sont crées par l'application et s'exécutent en concurrence sur les processeurs de la SMP. Dans le cas synchrone, touts les threads se synchronisent via une barrière de synchronisation (cf. chapitre I.2). Les communications flexibles sont naturelles, car chaque processeur de la SMP peut accéder directement en mémoire partagée à la valeur courante du vecteur itéré qui peut correspondre en fait à un itéré partiel réactualisé au cours du

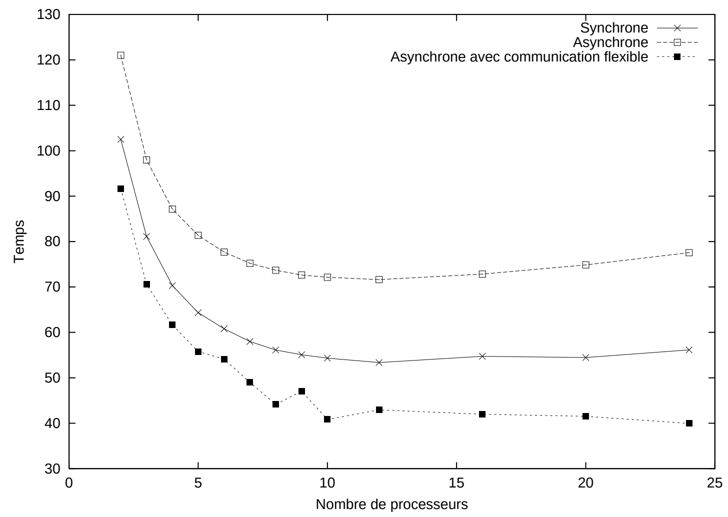

FIG. 2.7 – *Temps d'exécution sur le CRAYT3E*

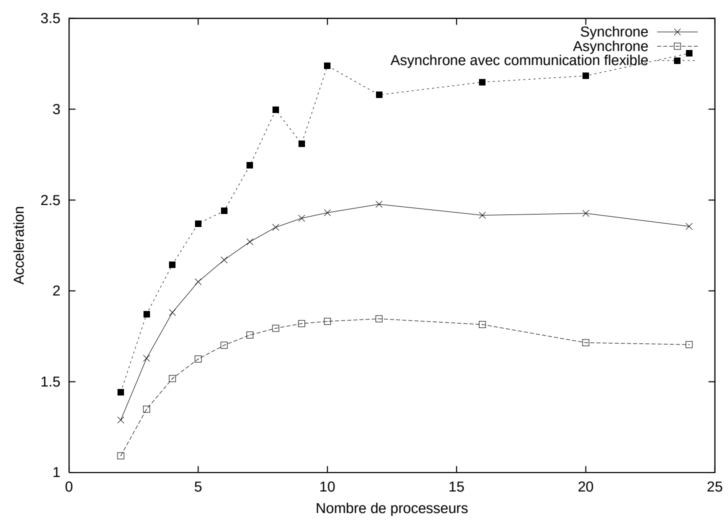

FIG. 2.8 – *Accélération des mises an œuvre parallèles sur le CRAYT3E*

itération	1	10	20	60	110	150	200	216
processeur								
0	7	7	6	5	4	3	2	2
1	7	7	7	6	5	4	3	3
2	7	7	7	6	5	4	3	3
3	7	6	6	6	4	4	2	2

TAB. 2.1 – *nombre moyen d'itérés partiels par itération sur 4 processeurs du CRAYT3E*

schéma itératif du second niveau.
Le tableau 2.2 montre les temps d'exécution et les accélérations des mises en œuvre

	Synchrones		Asynchrone avec c.f	
Threads	Temps(sec)	Accélération	Temps (sec	Accélération
1	677.13	1	677.13	1
2	512.36	1.321	465.00	1.455
3	401.90	1.684	347.51	1.948
4	347.96	1.946	313.72	2.158

TAB. 2.2 – *Performance sur une SMP*

synchrones et asynchrones avec communication flexible sur une SMP quadri processeur à mémoire partagée. On peut conclure que la mise en œuvre asynchrone avec communication flexible est plus performante que celle en synchrone. L'équilibrage de charge implique que les performances des deux types de mise en œuvre sont proches à l'opposé des performances obtenues au chapitre II.1. En comparant les performances obtenues sur le CRAYT3E et la SMP on peut conclure que la mise en œuvre asynchrone avec communication flexible en utilisant la fonction shmem_get() donne une efficacité qui est très proche de celle obtenu sur la SMP. Ceci montre que l'accès à la valeur courante du vecteur itéré donne toujours les meilleurs performances ; ce qui confirme les résultats obtenus au chapitre II.1.

2.5.4 Etude de performance sur un réseau de SMP

Comme au chapitre II.1, nous avons combiné deux techniques de mise en œuvre parallèle. La première utilise la bibliothèque MPICH qui est une version de MPI-1 sur une réseau de cluster (cf. [GE99]). La deuxième met aussi en œuvre le schéma itératif à deux niveaux avec une décomposition fine sur une SMP en utilisant les Threads POSIX. Les mises en œuvre synchrones et asynchrones avec communication flexible sont faites de la même manière qu'au chapitre II.1, où nous rappelons que le triplet (a,b,c) correspond au nombre de threads créés sur chaque SMP.

Les tableaux 2.3 et 2.4 montrent les temps d'exécution et les accélérations des mises en œuvre synchrones et asynchrones avec communication flexible respectivement sur un réseau Myrinet et un réseau Ethernet. Malgré le faible temps de latence sur Myrinet les itérations synchrones sont plus performantes que les itérations asynchrones avec communication flexible. Afin d'avoir de meilleures performances avec les itérations asynchrones avec communication flexible pour ce type de problème il faut une architecture à mémoire partagée ou une architecture à mémoire distribuée pour laquelle on peut accéder directement à la valeur courante du vecteur itéré avec un temps de latence très faible comme sur le CRAY T3E et ceci en utilisant la fonction shmem_get() de SHMEM.

	Synchrones		Asynchrone avec c.f	
Threads	Temps(sec)	Accélération	Temps(sec)	Accélération
(1,0,0)	361.94			
(2,0,0)	275.69	1.312	241.01	1.501
(1,1,0)	275.24	1.314	326	1.109
(2,1,0)	220.50	1.641	242.63	1.491
(2,2,0)	191.75	1.887	205.15	1.764
(1,1,1)	213.79	1.692	259.69	1.393
(2,1,1)	183.45	1.972	212.70	1.701
(2,2,1)	171.25	2.113	185.31	1.953
(2,2,2)	159.57	2.268	160.51	2.254

Tab. 2.3 – *Performance sur un réseau Myrinet de SMP en utilisant MPICH-gmm*

	Synchrones		Asynchrone avec c.f	
Threads	Temps(sec)	Accélération	Temps(sec)	Accélération
(1,1,0)	276	1.310	419.77	0.862
(2,1,0)	228.92	1.581	280.07	1.292
(2,2,0)	194.95	1.856	285.78	1.266
(1,1,1)	214.64	1.686	277.45	1.300
(2,1,1)	183.76	1.969	229.15	1.570
(2,2,1)	185.21	1.954	220.27	1.640
(2,2,2)	179.47	2.016	183.50	1.972

Tab. 2.4 – *Performance sur un réseau Ethernet de SMP en utilisant MPICH*

2.6 Conclusion

L'étude menée dans ce chapitre concerne la résolution en parallèle d'un problème de commande optimale pour la régulation d'un processus thermique. Nous avons étudié la convergence des itérations asynchrones avec communication flexible pour deux séquences monotones de sous-solutions et de sur solutions qui convergent vers une solution unique du problème. Nous avons mis en œuvre de manière synchrone, asynchrone et asynchrone avec communication flexible un schéma itératif à deux niveaux. Le premier niveau utilise l'algorithme iteratif par bloc en intégrant l'état dans le sens direct et l'état adjoint dans le sens inverse. Le deuxième niveau met en œuvre le schéma itératif de Gauss Seidel au sein d'un seul bloc du vecteur itéré. Notons que les mises en œuvre parallèle ne donnent pas d'excellentes performances en raison de la nature très séquentielle de la méthode numérique étudiée. De plus la mise en œuvre sur diverses architectures parallèles montre que ce type de problème donne les performances les meilleures pour la mise en œuvre asynchrone avec communication flexible sur une architecture à mémoire partagée telle qu'une SMP ou une architecture à mémoire distribuée pour laquelle on peut accéder rapidement à la valeur courante du vecteur itéré. Ceci montre l'intérêt des itérations asynchrones avec communication flexible.

Chapitre 3

Résolution d'un système Markovien

3.1 Introduction

Nous étudions dans ce chapitre les performances d'itérations parallèles pour la résolution d'un système Markovien. Les multiples travaux de recherche sur ce sujet sont à l'image du grand intérêt que suscite ce domaine. On peut citer par exemple la modélisation des réseaux à commutation de circuits (cf. [Leg86] et [Leg87]), la modélisation des réseaux de files d'attente (cf. [Kle75] et [Mar79]), etc.... En particulier, de nombreux travaux ont été effectués pour résoudre des systèmes Markoviens par des méthodes itératives parallèles (cf. [Leg92], [LC93], [Cal97], [LC98], [KH99] et [MPS99]) et plus particulièrement par des méthodes itératives asynchrones (cf. [LuM86], [BLA87], [BT89], [Elb94b], [Bah98a], [Bah98b], [Bah99a] et [BFK99]).
Nous étudions dans ce chapitre la résolution d'un système Markovien modélisant des réseaux à commutation de circuits par des méthodes itératives de relaxation et d'agrégation-désagrégation qui sont basées sur le schéma itératif de Gauss-Seidel. Nous présentons deux techniques différentes qui permettent de paralléliser le schéma itératif de Gauss-Seidel : la première proposée par Amodio et Mazzia (cf. [AM95]) qui donne un schéma itératif de Gauss-Seidel modifié et la deuxième utilise une technique de coloration des bloc-composantes en rouge et noir.
Nous comparons les performances des différentes mise en œuvre parallèles synchrones et asynchrones sur les trois types d'architectures présentés au chapitre I.2.

3.2 Génération d'un système Markovien

Les processus Markoviens sont des processus stochastiques pour lesquels la prédiction de l'état futur se fait en fonction de l'unique connaissance de l'état présent. Ils sont appelés aussi : processus stochastiques sans mémoire.
Dans ce paragraphe nous rappelons les différentes formes d'un système Markovien et nous présentons une formulation du problème que nous résolvons par la suite.

3.2.1 Chaînes de Markov à temps discrets

Dans le cas d'une chaîne de Markov à temps discrets, les événements interviennent à des instants de temps notés $\{0,1,2,...\}$. Le processus Markovien forme une séquence stochastique $\{X_1,X_2,...\}$ caractérisée par la propriété suivante :

$$Proba[X_{k+1} = j|X_k = i_k,X_{k-1} = i_{k-1},...,X_0 = i_0] = Proba[X_{k+1} = j|X_k = i_k]. \qquad (3.1)$$

Etant donné l'état courant i à un instant t_k, la probabilité d'évoluer de l'état i vers l'état j ne dépend que de i et de l'instant t_k de transition. Cette probabilité est appelée probabilité de transition notée par p_{ji},

$$p_{ji}(k) = Proba[X_{k+1} = j|X_k = i]. \qquad (3.2)$$

Soit E l'ensemble des états admissibles du système, alors les probabilités de transition $p_{ji}(k)$ vérifient :

$$\sum_{j \in E} p_{ji}(k) = 1. \qquad (3.3)$$

La résolution d'un système Markovien consiste à rechercher une distribution des probabilités des états du système vers un état futur à partir de la connaissance des probabilités de transition de la chaîne de Markov considérée et d'une distribution initiale. Nous considérons le cas particulier des chaînes de Markov homogènes où les probabilités de transition sont invariantes dans le temps ($p_{ji}(k) = p_{ji},\forall k$).
Désignons par S la matrice ayant pour coefficients les probabilités de transition p_{ji}, comme tous les coefficients de S sont compris entre 0 et 1, alors la matrice S est une matrice non négative (cf. Définition A.1.4 de l'annexe). Soit p_k le vecteur probabilité ayant pour composantes les probabilités des états du système à l'instant k. Le vecteur probabilité p_{k+1} à l'instant $k + 1$ s'exprime en fonction du vecteur probabilité p_k et de la matrice des probabilités de transition S par :

$$p_{k+1} = Sp_k, \qquad (3.4)$$

on a donc $p_{k+1} = S^{k+1}p_0$. A partir d'une distribution initiale p_0, le système évolue au cours du temps selon un régime transitoire pour atteindre, sous certaines hypothèses une valeur stationnaire. Le vecteur probabilité p stationnaire est la limite quand k tends vers l'infini du vecteur probabilité p_k. Le problème à résoudre se ramène à un problème de recherche de point fixe :

$$\begin{cases} p = Sp \text{ avec } p = [\pi_1,\pi_2,...,\pi_n]^T, \\ 0 < \pi_i \leq 1,\forall 1 \leq i \leq n, \\ \displaystyle\sum_{i=1}^{n} \pi_i = 1. \end{cases} \qquad (3.5)$$

3.2.2 Chaîne de Markov à temps continu

Dans le cas d'une chaîne de Markov à temps continu, les changements d'état ne se produisent pas à des instants discrets, mais le temps de séjour dans un état suit une loi exponentielle négative. Les probabilités de transition s'écrivent en fonction du temps :

$$p_{ji}(t_1,t_2) = Proba[X(t_2) = j|X(t_1) = i],t_1 \leq t_2. \tag{3.6}$$

Dans le cas d'une chaîne homogène, les probabilités de transition sont indépendantes des temps absolus t_1 et t_2, mais restent fonction du temps de séjour $\Delta t = t_2 - t_1$. Le temps de séjour dans un état est distribué exponentiellement (cf. [Kle75]).

$$p_{ji}(\Delta t) = e^{-\nu_i \Delta t}, \tag{3.7}$$

où $\nu_i > 0$ est un paramètre qui est fonction de l'état i et $\frac{1}{\nu_i}$ est la moyenne du temps de séjour dans l'état i. Dans le cas continu, la matrice des probabilités de transition S sera remplacée par la matrice des taux de transitions Q dont les éléments q_{ij} (cf. [Tij88] et [Cas93]) sont définis par :

$$\begin{cases} q_{ii} = \dfrac{d}{d(\Delta t)}[p_{ii}(\Delta t) - 1]|_{\Delta t=0}, \\ q_{ji} = \dfrac{d}{d(\Delta t)} p_{ji}(\Delta t)|_{\Delta t=0}, \, \forall i \neq j. \end{cases} \tag{3.8}$$

La valeur $-q_{ii} = \nu_i$ représente le taux instantané de départ de l'état i et q_{ji} est le taux instantané pour passer de l'état i vers l'état j. La matrice Q de coefficients q_{ji} est appelée générateur infinitésimal de la chaîne de Markov. D'après les équations (3.8), $\sum_{i=1}^{n} q_{ji} = 0$. Le système équivalent obtenu à partir du générateur infinitésimal Q est donné par :

$$\begin{cases} Qp = 0 \text{ avec } p = [\pi_1,\pi_2,...,\pi_n]^T, \\ 0 < \pi_i \leq 1, \, \forall 1 \leq i \leq n, \\ \displaystyle\sum_{i=1}^{n} \pi_i = 1. \end{cases} \tag{3.9}$$

3.2.3 Passage des taux de transition aux probabilités de transition

Il est possible de réécrire le système $Qp = 0$ sous la forme $Sp = p$. Soit ξ un réel choisi arbitrairement, alors le système $Qp = 0$ peut s'écrire sous la forme $\xi Qp + p = p$. On pose $S = \xi Q + I_n$, donc :

$$Qp = 0 \Rightarrow Sp = p. \tag{3.10}$$

Afin de garantir la non négativité de la matrice S, on choisit $0 < \xi < \overline{\xi}$, avec

$$\overline{\xi} = \frac{1}{max_{1 \leq i \leq n} |q_{ii}|}.$$

Pour avoir une solution stationnaire unique du système, il faut que S soit primitive, c'est-à-dire que le nombre de valeurs propres de S de module $\rho(S)$ est égale à 1 (cf. [Cas93]). Cette condition est remplie si $\xi < \overline{\xi}$ est satisfaite et dans ce cas là, la matrice S peut être vue comme étant la matrice des probabilités de transition d'une chaîne de Markov à temps discret avec une période entre deux transitions égale à ξ (cf. [Cas93]).

3.2.4 Application à un processus de naissance et de mort

Nous considérons un cas particulier de système à événement discret : un processus de naissance mort. La matrice Q aura dans ce cas là une structure tridiagonale par bloc :

$$Q = \begin{bmatrix} -\nu_1 & \mu_2 & 0 & 0 & 0 & \dots \\ \lambda_1 & -\nu_2 & \mu_3 & 0 & 0 & \dots \\ 0 & \lambda_2 & -\nu_3 & \mu_3 & 0 & \dots \\ \vdots & & & \ddots & & \\ & & & & & \end{bmatrix}$$

λ_i et μ_i représentent respectivement le taux de naissance et le taux de mort pour l'état i considéré.

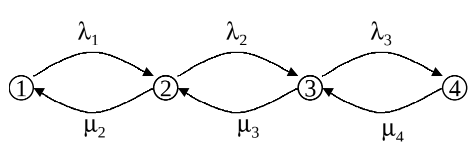

FIG. 3.1 – *processus de naissance et de mort*

Nous considérons le système Markovien :

$$Qp = 0, \tag{3.11}$$

où $p = [p_0,\ldots,p_{n-1}]$ est le vecteur probabilité stationnaire ayant n blocs et de dimension totale $z = \sum_{k=0}^{n-1} z_k$, z_k étant la dimension du bloc k. Les composantes de z_k, notées par $\pi_{k,i}$ où i est l'indice de la i-ème composante et k l'indice du k-ème bloc, vérifient, $\sum_{k=0}^{n-1}\sum_{i=0}^{z_k-1}\pi_{k,i} = 1$. Q est le générateur infinitésimal qui est une matrice tridiagonale par bloc, irréductible, apériodique et Z-matrice (cf. Définition A.1.1 de l'annexe). Sous ces hypothèses, il existe un unique vecteur p^* solution de (3.11) telle que $\sum_{k=0}^{n-1}\sum_{i=0}^{z_k-1}\pi^*_{k,i} = 1$.

3.3 Méthode itératives de résolution

Nous présentons dans ce paragraphe quelques méthodes itératives de résolution qui sont basées sur le schéma itératif de Gauss-Seidel. Ce dernier présente l'avantage de calculer les composantes du vecteur itéré en fonction des dernières valeurs réactulisées des composantes qui sont disponibles en mémoire. Ceci permet d'une part d'économiser l'espace mémoire utilisé, ce qui est très important pour un problème de très grande dimension et d'autre part d'avoir une exécution plus rapide comparée au schéma itératif de Jacobi.

3.3.1 Schéma itératif de Gauss-Seidel

Soit $Q = D - L - U$, L étant la matrice triangulaire inférieure par bloc, D la matrice diagonale par bloc et U la matrice triangulaire supérieure par bloc. Le schéma itératif de Gauss-Seidel est le suivant:

$$p^{(r)} = D^{-1}[Lp^{(r)} + Up^{(r-1)}].$$

La matrice d'itération de Gauss-Seidel est :

$$T_{GS} = [D - L]^{-1}U.$$

La convergence est assurée si : $lim_{r\to\infty}|p^{(r)} - p^*| = 0$, où p^* est la solution de (3.11) c'est-à-dire si $lim_{r\to\infty}T_{GS}^{(r)} = \mathbb{O}$, où $\mathbb{O}$ est la matrice de coefficients nuls. Ceci est réalisé, quelque soient les conditions initiales, si et seulement si toutes les valeurs propres de T_{GS} sont strictement inférieures en module à l'unité. Toutefois, cette condition n'est pas vérifiée dans le cas d'un système Markovien, car la matrice d'itérations T_{GS} admet une valeur propre unitaire.
T_{GS} est non négative et la valeur propre unitaire associée est de multiplicité 1 (matrice primitive), donc si la matrice Q est apériodique, alors le schéma itératif $p^{(r+1)} = T_{GS}p^{(r)}$ vérifie $lim_{r\to\infty}p^{(r)} = \mu p$, où μ est une constante positive (cf. [Ros84] et [BP86]).

3.3.2 Méthode de sur relaxation

La méthode de sur relaxation ou méthode SOR est dérivée de la méthode de Gauss-Seidel en ajoutant un paramètre de relaxation ω :

$$p^{(r)} = D^{-1}[(1-\omega)p^{(r-1)} + \omega(Lp^{(r)} + Up^{(r-1)})].$$

La méthode de relaxation est alors exprimée en fonction du vecteur calculé à l'aide du schéma itératif de Gauss-Seidel et du vecteur probabilité calculé à l'itération $(r-1)$, Si $\omega < 1$, on réalise une sous-relaxation et si $\omega > 1$, on réalise une sur relaxation. Le schéma itératif de Gauss-Seidel se confond avec la méthode de relaxation quand le paramètre $\omega = 1$.
La matrice correspondant à la méthode de sur relaxation est:

$$T_{SOR} = (D - \omega L)^{-1}\{(1-\omega)D + \omega U\}.$$

3.3.3 Méthode d'agrégation-désagrégation

La méthode d'agrégation-désagrégation de Takahashi [Tak75] est une méthode itérative par bloc qui comprend trois étapes :

1) l'étape d'agrégation est la recherche des probabilités des blocs connaissant les probabilités conditionnelles des états dans chaque bloc.

2) l'étape de désagrégation est la recherche des probabilités conditionnelles des états dans chaque bloc connaissant les probabilités des blocs et les probabilités conditionnelles des états dans les autres blocs.

3) la dernière étape est le test de convergence.

On note : $\omega^T = [\omega_0,...,\omega_{n-1}]$, où ω_k est la probabilité du bloc k ayant pour expression :

$$\omega_k = \xi_k^T.p_k = \sum_{i=1}^{z_k} \pi_{k,i},$$

et où $\xi_k^T = [1,...,1]$ est le vecteur unité de dimension z_k. La probabilité conditionnelle de l'état i du bloc k s'écrit alors:

$$\beta_{k,i} = \frac{\pi_{k,i}}{\omega_k}.$$

Soit b_k le vecteur des probabilités conditionnelles des états du bloc k,

$$b_k^T = [\beta_{k,1},...,\beta_{k,zk}] = \frac{1}{\omega_k}[\pi_{k,1},...,\pi_{k,zk}] = \frac{p_k^T}{\omega_k},$$

$b^T = [b_0^T,...,b_{n-1}^T]$ est le vecteur des probabilités conditionnelles de tous les états du système.

Dans le cas particulier d'un réseau à commutation de circuit, nous notons par Φ la somme des paramètres de Poisson de tous les trafics offerts au réseau, par Bk l'ensemble des états du module k bloquant au moins un trafic en dernier choix et par $\gamma(l,k)$ la somme des paramètres de Poisson des flux bloqués en dernier choix par l'état de probabilité conditionnelle $\beta_{k,l}$. Nous avons alors (cf. [Leg87]) :

$$R_k = \frac{\omega_{k+1}}{\omega_k} = \frac{\Phi - \sum_{Bk} \gamma(l,k).\beta_{k,l}}{k+1}$$

Les étapes de la méthode d'agrégation-désagrégation de Takahashi sont résumées dans l'algorithme 8 (cf. [Tak75], voir aussi [Leg87]).

Algorithme 8 : Agrégation-désagrégation

$r = 0$

initialisation de $b_k^{(0)}$, $k = 0,...,n-1$, en respectant $\sum_l \beta_{k,l}^{(0)} = 1$

$R_k^{(0)} = \Phi - \sum_{Bk} \gamma(l,k).\beta_{k,l}^{(0)}$, $k = 0,...,n-1$

rpter

r=r+1

pour $(k = 0,n-1)$ **faire**

$\theta^{(r)} = Q_{kk}^{-1}[\frac{1}{R_{k-1}^{(r)}}.Q_{k-1,k}.b_{k-1}^{(r)} - R_k^{(r-1)}Q_{k,k+1}.b_{k+1}^{(r-1)}]$,

$b_k^{(r)} = (1-\omega)b_k^{(r-1)} + \omega\theta^{(r)}$

si $Bk \neq \emptyset$ **alors**

Normalisation de $b_k \leftarrow \frac{b_k}{\xi_k^T.b_k}$

fin si

$R_k^{(r)} = \frac{\omega_{k+1}}{\omega_k} = \frac{\Phi - \sum_{BK} \gamma(l,k).\beta_{k,l}^{(r)}}{k+1}$

fin pour

jusqu' $(||b^{(r)} - b^{(r-1)}||_\infty < \epsilon ||b^{(r-1)}||_\infty)$

3.3.4 Résultats numériques

Nous comparons les différentes méthodes itératives séquentielles présentées au paragraphe 3.3 pour différentes dimensions du problème qui sont relatives au nombre d'états. La génération d'un problème se fait grce à un module de génération d'un système Markovien relatif à un réseau téléphonique à commutation de circuit (cf. [Leg86] et [Cal97]). Le tableau 3.1 donne le nombre d'itérations en séquentiel pour les méthodes de relaxation, de sur relaxation et d'agrégation-désagrégation. Le paramètre $\omega = 1.4$ donne le plus faible temps de calcul pour la méthode de sur relaxation et pour la méthode d'agrégation-désagrégation. Le tableau 3.1 montre que la méthode d'agrégation-désagrégation est plus rapide que les autres méthodes itératives pour un même problème. Cette méthode itérative présente une meilleure stabilité numérique puisqu'elle travaille sur des probabilités conditionnelles qui sont des probabilités plus grandes que les probabilités absolues de chaque état (cf. [Leg87]).

	relaxation		sur relaxation		(agré-désagré)gation	
dimension	itérations	temps	itérations	temps	itérations	temps
1960	112	3.49	47	1.49	28	0.91
3464	169	15.22	75	6.42	66	5.78
6110	197	48.01	89	22.02	80	20.00

TAB. 3.1 – *Comparaison en séquentiel des méthodes itératives de résolution*

3.4 Parallélisation du schéma itératif de Gauss-Seidel

Etant donné que les différentes méthodes itératives sont basée sur le schéma Gauss-Seidel, alors une parallélisation efficace de ce schéma itératif, nous permettra à priori d'obtenir une bonne efficacité des différentes méthodes itératives présentées au paragraphe 3.3. Nous donnerons dans ce paragraphe deux techniques de parallélisation du schéma Gauss-Seidel : la première est celle d'Amodio et Mazzia (cf. [AM95]) et la deuxième est basée sur la coloration des bloc-composantes en rouge et noir (cf. [Ort88]).

3.4.1 Méthode d'Amodio et Mazzia

P. Amodio et F. Mazzia ont proposé dans [AM95] une autre variante du schéma itératif de Gauss-Seidel appliquée à un système linéaire tridiagonal par bloc appelée méthode de Gauss-Seidel parallèle.

3.4.1.1 Transformation de la matrice de Gauss-Seidel

Considérons le système $Qp = 0$, où

$$Q = \begin{pmatrix} Q_{0,0} & Q_{0,1} & & & \\ Q_{1,0} & Q_{1,1} & Q_{1,2} & & \\ & \ddots & \ddots & \ddots & \\ & & & Q_{n-1,n-2} & Q_{n-1,n-1} \end{pmatrix} = \begin{pmatrix} A_0 & C_0 & & & \\ B_1 & A_1 & C_1 & & \\ & \ddots & \ddots & \ddots & \\ & & & B_{n-1} & A_{n-1} \end{pmatrix}$$

Supposons qu'on a $\alpha < n$ processeurs et considérons la décomposition suivante de Q :

$$Q = D - L - L' - U - U'.$$

Les matrices D, L, L', U et U' s'expriment de la manière suivante :

$$D = \begin{pmatrix} D_0 & & & \\ & D_1 & & \\ & & \ddots & \\ & & & D_{\alpha-1} \end{pmatrix}, D_i = \begin{pmatrix} A_{N_{(i)}} & 0 & & \\ 0 & A_{N_{(i)}+1} & & \\ & & \ddots & \\ & & & A_{N_{(i)}+n_i-1} \end{pmatrix},$$

$$L = \begin{pmatrix} L_0 & & & \\ & L_1 & & \\ & & \ddots & \\ & & & L_{\alpha-1} \end{pmatrix}, L_i = \begin{pmatrix} 0 & & & \\ -B_{N_{(i)}+1} & 0 & & \\ & \ddots & & \\ & & -B_{N_{(i)}+n_i-1} & 0 \end{pmatrix},$$

$$L' = \begin{pmatrix} 0 & & & \\ L'_1 & 0 & & \\ & \ddots & \ddots & \\ & & L'_{\alpha-1} & 0 \end{pmatrix}, L'_i = \begin{pmatrix} 0 & . & . & -B_{N_{(i)}} \\ & 0 & 0 & \\ & & \ddots & . \\ & & & 0 \end{pmatrix},$$

$$U = \begin{pmatrix} U_0 & & & \\ & U_1 & & \\ & & \ddots & \\ & & & U_{\alpha-1} \end{pmatrix}, U_i = \begin{pmatrix} 0 & -C_{N_{(i)}} & & \\ & 0 & \ddots & \\ & & \ddots & -C_{N_{(i)}+n_i-2} \\ & & & 0 \end{pmatrix},$$

$$U' = \begin{pmatrix} 0 & U'_0 & & \\ & 0 & \ddots & \\ & & \ddots & U'_{\alpha-2} \\ & & & 0 \end{pmatrix}, U'_i = \begin{pmatrix} 0 & . & . & 0 \\ . & & & . \\ 0 & & \ddots & . \\ -C_{N_{(i)}+n_i-1} & 0 & . & 0 \end{pmatrix},$$

où n_i et $N_{(i)}$ sont respectivement le nombre de blocs et le numéro du premier bloc réactualisés par le processeur i, $i \in \{0,...,\alpha-1\}$ et $N_{(i)} = \sum_{j=0}^{i} n_j$.

3.4.1.2 Schéma itératif de Gauss-Seidel modifié

Le schéma itératif de Gauss-Seidel classique est exprimé par :

$$(D - L - L')p^{(r)} = (U + U')p^{(r-1)}, r = 1,2,... \tag{3.12}$$

et la matrice d'itération est données par $G = (D - L - L')^{-1}(U + U')$
Le schéma itératif de Gauss-Seidel modifié est exprimé par :

$$(D - L - U')p^{(r)} = (U + L')p^{(r-1)}, r = 1,2,... \tag{3.13}$$

et la matrice d'itération $G_p = (D - L - U')^{-1}(U + L')$
Il a été démontré dans [AM95] que les matrices G et G_p ont le même rayon spectral et l'algorithme 10 donne l'algorithme de Gauss-Seidel modifié, appelé aussi algorithme de Gauss-Seidel parallèle.

3.4.2 Méthode de coloration en rouge et noir

La méthode de Gauss-Seidel peut être vue comme nécessitant la résolution d'un système ayant une matrice triangulaire inférieure. Dans ce cas particulier de matrice tridiagonale par bloc (cf. [Ort88]), il est intéressant de répartir les blocs en deux catégories qu'on nomme rouge et noir. Un bloc de couleur rouge ne dépend que des blocs de couleur noire et vice versa. Les processeurs peuvent réactualiser ensemble tous les blocs de couleur rouge puis les blocs de couleur noire.

En termes mathématiques, ceci revient à appliquer le schéma itératif de Gauss-Seidel au système d'équations $PQp = 0$, où P est une matrice de permutation. Dans le cas où les bloc-composantes du vecteur probabilité sont réparties en rouge et noir de la manière suivante : $p = [p_0^R, p_1^N, ..., p_{2l}^R, p_{2l+1}^N, ...]$, le système $Qp = 0$ peut s'écrire sous la forme :

Algorithme 9 : algorithme de Gauss-Seidel modifié

$B_0 \equiv C_{n-1} \equiv 0$
si $(i \neq \alpha - 1)$ **alors**
  Envoi de $p^{(r-1)}_{N_{(i)}+n_i-1}$ au processeur $i+1$
fin si
si $(i \neq 0)$ **alors**
  Reception de $p^{(r-1)}_{N_{(i)}-1}$ du processeur $i-1$
fin si
$p^{(r)}_{N_{(i)}} = A^{-1}_{N_{(i)}}(-B_{N_{(i)}}p^{(r-1)}_{N_{(i)}-1} - C_{N_{(i)}}p^{(r-1)}_{N_{(i)}+1})$
pour $(j = N_{(i)}+1,\, N_{(i)}+n_i-2)$ **faire**
  $p^{(r)}_j = A^{-1}_j(-B_j p^{(r)}_{j-1} - C_j p^{(r-1)}_{j+1})$
fin pour
si $(i \neq 0)$ **alors**
  Envoi de $p^{(r)}_{N_{(i)}}$ au processeur $i-1$
fin si
si $(i \neq \alpha - 1)$ **alors**
  Reception de $p^{(r)}_{N_{(i)}+n_i}$ du processeur $i+1$
fin si
$p^{(r)}_{N_{(i)}+n_i-1} = A^{-1}_{N_{(i)}+n_i-1}(-B_{N_{(i)}+n_i-1}p^{(r)}_{N_{(i)}+n_i-2} - C_{N_{(i)}+n_i-1}p^{(r)}_{N_{(i)}+n_i})$

$$\begin{pmatrix} \mathcal{D}^{(R)} & \mathcal{U}^{(N)} \\ \mathcal{L}^{(R)} & \mathcal{D}^{(N)} \end{pmatrix} \begin{pmatrix} p_R \\ p_N \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

avec

$$\mathcal{D}^{(R)} = \begin{pmatrix} Q_{0,0}^{(R)} & & \\ & Q_{2,2}^{(R)} & \\ & & \ddots \end{pmatrix}, \mathcal{D}^{(N)} = \begin{pmatrix} Q_{1,1}^{(N)} & & \\ & Q_{3,3}^{(N)} & \\ & & \ddots \end{pmatrix},$$

$$\mathcal{U}^{(N)} = \begin{pmatrix} Q_{0,1}^{(N)} & & \\ Q_{2,1}^{(N)} & Q_{2,3}^{(N)} & \\ & \ddots & \ddots \end{pmatrix}, \mathcal{L}^{(R)} = \begin{pmatrix} Q_{1,0}^{(R)} & Q_{1,2}^{(R)} & & \\ & Q_{3,2}^{(R)} & Q_{3,4}^{(R)} & \\ & & \ddots & \ddots \end{pmatrix}.$$

Le schéma itératif de Gauss-Seidel sera décomposé en deux parties. La première partie met en œuvre le système d'équation en bloc-composantes rouges, soit: $p_R^{(r+1)} = -\mathcal{D}^{(R)-1}\mathcal{U}^{(N)}p_N^{(r)}$ et la deuxième partie met en œuvre le système d'équations en bloc-composantes noires $p_N^{(r+1)} = -\mathcal{D}^{(N)-1}\mathcal{L}^{(R)}p_R^{(r+1)}$ (cf. [Ort88]).

3.5 Les itérations parallèles

Les itérations parallèles synchrones et asynchrones peuvent être mises en œuvre en choisissant une méthode itérative de résolution (relaxation, sur relaxation et agrégation-désagrégation) et une technique de parallélisation du schéma itératif de Gauss-Seidel (méthode d'Amodio et Mazzia ou méthode de coloration en rouge et noir).
Afin d'équilibrer au mieux les charges de calcul, nous proposons dans ce paragraphe un algorithme qui détermine le nombre de blocs associés à chaque processeur. Ensuite nous présentons quelques méthodes de terminaison des itérations asynchrones.

3.5.1 Décomposition en sous-domaines

Considérons la résolution en parallèle du système (3.11) sur α processeurs. Soit z la dimension du système Markovien et z_k la dimension du kème bloc. Notons par s_q le quotient de la division euclidienne de z par le nombre de processeurs α. L'algorithme 10 de décomposition en sous-domaines consiste à déterminer pour chaque processeur P le numéro du premier bloc, noté $n\_debut(P)$, et le numéro du dernier bloc, noté $n\_fin(P)$, de telle sorte que $(\sum_{l=n\_debut(P)}^{n\_fin(P)} z_l)$ soit le plus proche possible de s_q.
Pour un processeur donné et à partir des blocs assignés aux autres processeurs, l'algorithme additionne itérativement la taille des blocs non assignés jusqu'à ce que la somme s_l dépasse la valeur s_q tout en mémorisant l'ancien itéré $s_{lmoins1}$. La valeur retenue dans ce cas correspond au minimum entre $|s_q - s_l|$ et $|s_q - s_{lmoins1}|$.
Les deux figures 3.2 et 3.3 montrent deux répartitions différentes de charge d'un problème de dimension 6110 sur 12 processeurs: la première (cf. figure 3.2) affecte sensiblement le

Algorithme 10 : Décomposition en sous-domaines

initialisation: $n\_fin(0) = 0$, $l = 0$ et $s_q = \frac{z}{\alpha}$

pour $P = 1, \alpha - 1$ **faire**

  $n\_debut(P) = n\_fin(P-1) + 1$

  $s_{lmoins1} = 0$

  $s_l = z_l$

  rpter

    $s_{lmoins1} = s_l$

    $s_l = s_{lmoins1} + z_{l+1}$

    $l = l + 1$

  jusqu' $s_l > s_q$

  si $|s_l - s_q| < |s_{lmoins1} - s_q|$ **alors**

    $l = l + 1$

  fin si

  $n\_fin(P) = l - 1$

fin pour

$n\_debut(\alpha) = n\_fin(\alpha - 1) + 1$

$n\_fin(\alpha) = n$

même nombre de blocs sur les 12 processeurs, cependant ces blocs sont de taille différentes, les blocs intermédiaires étant particulièrement grand, et la deuxième (cf. figure 3.3) affecte les blocs selon l'algorithme 10. En faisant une répartition équitable en nombre de bloc, la charge sera très déséquilibrée, mais en calculant, selon l'algorithme 10, le nombre adéquat de blocs sur chaque processeur, on obtient un meilleur équilibrage entre les différents processeurs.

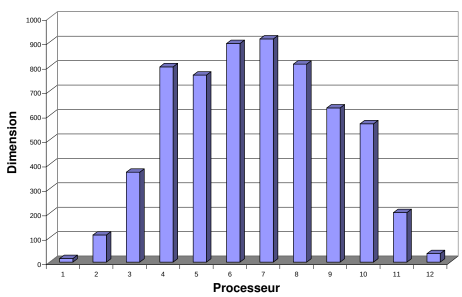

FIG. 3.2 – *Charge très déséquilibrée*

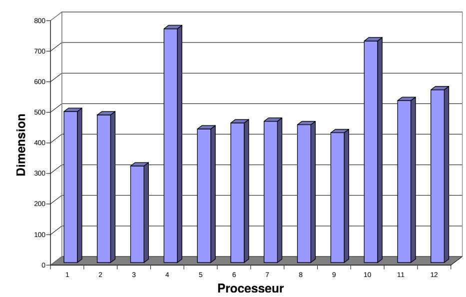

FIG. 3.3 – *équilibrage avec l'algorithme 10*

3.5.2 Terminaison des itérations asynchrones

Sur une architecture à mémoire distribuée, la terminaison des itérations parallèles asynchrones présente de nombreuses difficultés vu que chaque processeur possède uniquement des informations locales, il peut ne pas y avoir d'horloge globale et les délais de communication peuvent être très longs.
Contrairement aux chapitres II.1 et II.2, où nous avons présenté des méthodes de terminaison simples à mettre en œuvre et se prêtent bien à un contexte de convergence monotone, la terminaison des itérations asynchrones pour ce type d'application nécessite un algorithme général. Nous présentons deux méthodes de terminaison : la méthode de Bertsekas et Tsitsiklis (cf. [BT89] et [BT91]) basée sur l'acquittement des tous les messages reus et la méthode d' El Baz (cf. [EB96b]) basée sur les messages d'activation.

3.5.2.1 Méthode de terminaison avec acquittement systématique

L'algorithme itératif asynchrone doit être modifié afin qu'il converge en temps fini vers un point fixe suffisamment proche de la solution du problème.

Initialement un seul processeur est actif qui est appelé processeur racine et tous les autres processeurs sont inactifs. Dans l'état inactif un processeur ne transmet ni message ni acquittement. Un processeur P inactif devient actif quand il reoit un message provenant d'un autre processeur P'; encore appelé le père de P. Le message qui a activé P, joue un rle particulièrement important, on lui donne aussi le nom de message critique. Dans l'état actif un processeur transmet les messages et envoie immédiatement un acquittement pour tout message reçu à l'exclusion du message critique. Un processeur P actif devient inactif s'il satisfait sa condition de terminaison locale, s'il a envoyé l'acquittement de tous les messages reçus à l'exception du message critique et s'il a reu l'acquittement de tous les messages envoyés. En devenant inactif un processeur P envoie l'acquittement du message critique. La terminaison globale est détectée au moment où le processeur racine devient inactif. L'algorithme de détection de la terminaison est donné par la machine à états finis pour un processeur de la figure 3.4.

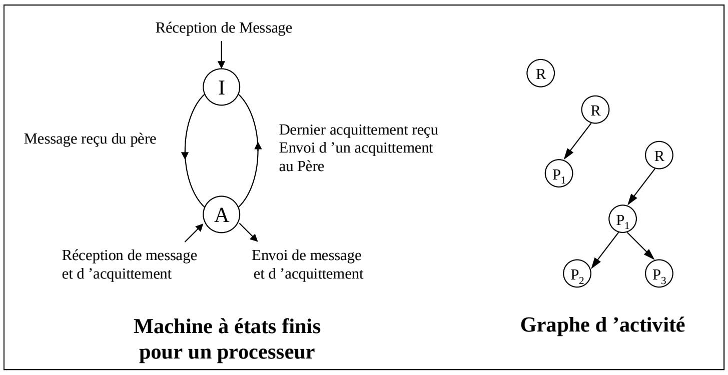

FIG. 3.4 – *Modèle de l'algorithme de détection de la terminaison avec acquittement*

La réactualisation d'une composante du vecteur itéré ne peut avoir lieu que lorsque sa valeur change d'une manière significative. Dans ce cas, cette valeur peut être communiquée aux autres processeurs, sinon elle ne sera ni communiquée ni affectée à cette composante. L'algorithme asynchrone se termine lorsqu'il n'y a aucune modification des valeurs des composantes du vecteur itéré pour tous les processeurs (toutes les conditions de terminaison locales sont satisfaites) et aucun message n'est en transit dans le réseau de communication.

Tous les processeurs sont progressivement activés par la réception de message. Le graphe d'activité évolue en fonction des messages critiques reçus, de la satisfaction des conditions de terminaison locales et de la réception d'acquittements.
L'implémentation a été effectuée en associant à chaque processeur actif une liste des processeurs fils. Cette liste contient un ensemble d'attributs: le compteur des messages envoyés à un processeurs fils, un compteur de messages reus et un compteur d'acquittement. Un processeur fils qui devient inactif est automatiquement supprimé de cette liste. Un processeur père devient inactif quand la liste des fils est vide et sa terminaison locale est satisfaite. Un message de terminaison globale est envoyé par le processeur racine au moment ou il devient inactif.

Cette méthode de terminaison nécessité deux fois plus de communications qu'un algorithme itératif classique en raison de l'envoi d'acquittements et des conditions plus restrictives que les conditions du théorème général de convergence asynchrone de Bertsekas [BT89] afin de garantir la convergence de l'algorithme asynchrone modifié (cf. [BT91]).

Bertsekas et Tsitsiklis ont suggéré dans [BT91] une autre procédure de terminaison à savoir l'algorithme de "snapshot", ou cliché de Chandy et Lamport (cf. [CL85]). Cette méthode est basée sur la production de messages de marquage et l'enregistrement des états de liens et des processeurs lorsque les messages de marquage sont délivrés.

3.5.2.2 Méthode de terminaison sans acquittement systématique

Cette méthode de terminaison repose sur le l'hypothèse que le temps de transmission t_t des messages entre toute paire de processeurs adjacents est borné par une constante positive δ qui dépend de l'architecture du réseau qui relie les différents processeurs. On a :

$$t_t < \delta.$$

On suppose aussi que les messages entre toute paire de processeurs adjacents sont reçus suivant leur ordre d'émission (cf. [EB96b] et [Elb98]). Le fonctionnement d'un processeur mettant en œuvre l'algorithme itératif asynchrone modifié est représenté par la machine à états finis de la figure 3.5 pour laquelle chaque processeur peut avoir trois états : actif (A), inactif (I), et terminal (T).

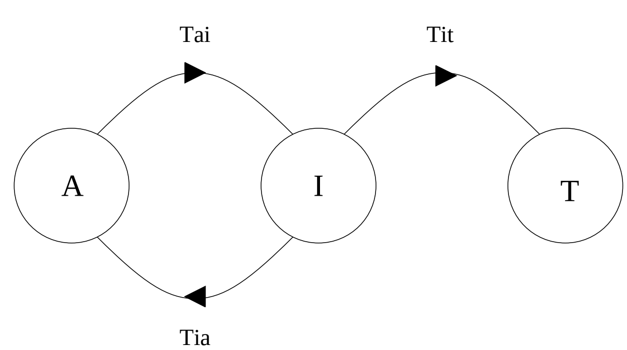

FIG. 3.5 – *Modèle de terminaison sans acquittement*

Initialement, un seul processeur est actif. Ce processeur est appelé la racine. Tous les autres processeurs sont inactifs. Quatre types de message peuvent être émis par chaque processeur: les valeurs des composantes du vecteur itéré, les messages d'activité, les messages d'inactivité et les messages de terminaison.

Dans l'état actif (A), un processeur P évalue le test de terminaison local sur la base des dernières valeurs des itérés qui sont disponibles dans sa mémoire locale. Si le test de terminaison local est satisfait, alors le processeur n'effectue pas de réactualisation ; sinon les composantes du vecteur itéré qui sont assignées à P sont réactualisées, les valeurs qui résultent de ces calculs sont envoyées aux processeurs adjacents, et le processeur P attend des messages d'activité pendant un laps de temps égal à $2.\delta$. Le processeur P ajoute le processeur P' à la liste de ses fils dès la reception d'un message d'activité provenant du processeur P'. Le processeur P retire le processeur P' de la liste de ses fils dès la reception d'un message d'inactivité provenant de P'.

Dans l'état d'inactivité (I), un processeur est en attente de message.

Dans l'état terminal (T) un processeur ne fait plus rien.

Dans l'état actif (transition Tia), un processeur P évalue le test de terminaison local sur la base des dernières valeurs des itérés qui sont disponibles dans sa mémoire locale. Si le test de terminaison local est satisfait, alors le processeur n'effectue pas de réactualisation ; sinon les composantes du vecteur itéré qui sont assignées à P sont réactualisées, les valeurs qui résultent de ces calculs sont envoyées aux processeurs adjacents, et le processeur P attend des messages d'activité pendant un laps de temps égal à $2.\delta$. Le processeur P ajoute le processeur P' à la liste de ses fils dès la reception d'un message d'activité provenant du processeur P'. Le processeur P retire le processeur P' de la liste de ses fils dès la reception d'un message d'inactivité provenant de P'.

Un processeur actif devient inactif (transition Tai) si la liste de ses fils est vide et son test de terminaison local est satisfait. Lorsque la transition Tai est tirée, le processeur envoie un message d'inactivité à son père .

Le processeur racine R passe immédiatement de l'état inactif à l'état terminal (Transition Tit). Lorsque la transition Tit est tirée, des messages de terminaison sont envoyés aux nœuds adjacents de R dans l'arbre recouvrant des processeurs noté T. Un processeur inactif P différent de R passe de l'état inactif à l'état terminal lorsqu'il reçoit un message de terminaison provenant d'un sommet adjacent de niveau inférieur dans l'arbre T des processeurs. Lorsque la transition Tit est tirée, le processeur P envoie des messages de terminaison aux nœuds adjacents de niveau supérieur dans l'arbre T.

Le fonctionnement de la méthode peut être résumé de la manière suivante : initialement, seul le processeur racine R est actif et tous les autres processeurs sont inactifs. Tous les processeurs deviennent progressivement actifs dès la réception de la valeur d'un sous-ensemble de composantes du vecteur itéré calculées par d'autres processeurs. Un graphe d'activité est engendré ; la topologie de ce graphe évolue au fur et à mesure que les différents messages sont reçus et les tests de terminaison locaux sont satisfaits.

Dans [EB96b], il a été démontré de manière formelle que la méthode présentée dans ce paragraphe détecte la terminaison en un temps fini lorsque celle ci s'est produite.

3.6 Etude de performance

Nous étudions dans ce paragraphe les performances des algorithmes itératifs parallèles pour les différentes méthodes étudiées au paragraphe 3.3. Nous présentons par la suite le temps de calcul, le nombre d'itération et l'efficacité des différentes mises en œuvre pour le cas d'un système Markovien de dimension 6110 répartie sur 31 blocs de taille différente. La précision de terminaison est $\epsilon = 10^{-4}$.

3.6.1 Etude de performance sur CRAY T3E

Nous comparons tout d'abord les performances des itérations parallèles synchrones en considérant les différents schémas itératifs parallèles. Ensuite nous comparons les performances des itérations asynchrones pour les différentes méthode de terminaison présentées au paragraphe 3.5.2. Finalement, nous comparons les meilleures performances des itérations synchrones avec les meilleures performances des itérations asynchrones. Nous notons par DIV le cas où on obtient une divergence.

a) Comparaison des performances des itérations synchrones

Méthode de relaxation									
processeurs	1	2	3	4	6	8	10	12	16
itérations	197	210	220	232	248	265	286	302	336
temps	48.01	29.77	26.66	22.00	13.30	13.65	13.66	11.15	9.25
efficacité		0.808	0.600	0.545	0.601	0.438	0.351	0.358	0.324
Méthode de sur relaxation									
itérations	89	104	116	148	190	DIV	DIV	DIV	DIV
temps	22.02	15.01	14.23	14.26	10.38	DIV	DIV	DIV	DIV
efficacité		0.733	0.515	0.385	0.353	DIV	DIV	DIV	DIV
Méthode d'agrégation-désagrégation									
itérations	80	92	101	112	124	DIV	DIV	DIV	DIV
temps	20.00	13.38	12.51	10.87	6.89	DIV	DIV	DIV	DIV
efficacité		0.747	0.532	0.457	0.483	DIV	DIV	DIV	DIV

TAB. 3.2 – *Performance des itérations parallèles synchrones sur CRAYT3E*

Les résultats numériques du tableau 3.2 sont relatifs aux différentes méthodes numériques parallèles pour lesquelles nous avons adopté le schéma Gauss-Seidel exclusivement à l'intérieur de chaque processeur. Ces résultats montrent qu'on ne peut avoir de très bonnes efficacités en parallèle, car généralement le gain apporté par le parallélisme en effectuant une décomposition du problème en sous-domaines se fait au détriment du taux de convergence. Au delà de 6 processeurs les méthodes de relaxation et d'agrégation-désagrégation divergent.

Méthode de relaxation									
processeurs	1	2	3	4	6	8	10	12	16
itérations	197	197	197	198	198	214	215	254	312
temps	48.01	27.92	23.87	18.80	12.80	11.52	11.29	10.34	8.64
efficacité		0.859	0.670	0.638	0.625	0.520	0.425	0.386	0.347
Méthode de sur relaxation									
itérations	89	88	88	87	87	126	171	DIV	DIV
temps	22.02	12.67	10.81	8.39	5.78	6.96	9.14	DIV	DIV
efficacité		0.868	0.678	0.655	0.634	0.394	0.240	DIV	DIV
Méthode d'agrégation-désagrégation									
itérations	80	79	78	78	76	91	116	DIV	DIV
temps	20.00	11.49	9.67	7.61	5.09	5.12	6.32	DIV	DIV
efficacité		0.870	0.688	0.656	0.654	0.487	0.316	DIV	DIV

TAB. 3.3 – *Performance des itérations parallèls synchrones sur CRAYT3E en appliquant la méthode d'Amodio et Mazzia*

Le tableau 3.3 montre que le taux de convergence est pratiquement conservé en parallèle. Le nombre d'itération chute très peu jusqu'au 6 processeurs. Quand le nombre de processeurs dépasse 6, le taux de convergence commence à chuter. En effet, à partir de 6 processeurs, il y aura au moins un processeur qui réactualise un seul bloc. Cette méthode est efficace quand le nombre de blocs du vecteur itéré affectés à un processeur est supérieur ou égal à 2, car alors le schéma d'Amodio et Mazzia est effectivement mis en œuvre.

Le tableau 3.4 est relatif à la coloration en rouge et noir ; il montre que le nombre d'itérations en parallèle reste très proche du nombre d'itération en séquentiel et reste pratiquement constant lorsque le le nombre de processeurs augmente. La décomposition en rouge et noir permet de maintenir le taux de convergence de la mise en œuvre en parallèle constant et par suite d'avoir de meilleures performances. Contrairement à la méthode de Amodio et Mazzia, cette méthode permet de conserver le taux de convergence en parallèle même si un ou plusieurs processeurs réactualisent une seule composante du vecteur itéré. Avec la technique de coloration en rouge et noir nous obtenons de meilleures performances pour la méthode d'agrégation-désagrégation, mais pour la méthode de relaxation ou de sur relaxation les performances sont proches de celles obtenues avec le schéma d'Amodio et Mazzia.

b) Comparaison des performances des itérations parallèles asynchrones

Nous avons mesuré expérimentalement le délai de communication entre deux processeurs du CRAY T3E et nous avons obtenu une borne supérieure égale à 0.5ms, donc $2 * \delta = 1ms$ conformément au paragraphe 3.5.2. Les tableaux 3.5 et 3.6 montrent que la méthode de terminaison sans acquittement systématique de tous les messages donne généralement de meilleures performances que celle avec acquittement. Ceci prouve que le nombre de messages d'activité et d'inactivité est négligeable comparativement au nombre de messages relatifs aux valeurs des composantes du vecteur itéré qui sont échangées entre les processeurs et donc au nombre de messages d'acquittement de l'algorithme de Bertsekas et Tsitsiklis.
Nous n'avons pas mis en œuvre en asynchrone la méthode d'agrégation-désagrégation, car il n'existe pas dans la littérature de résultats théoriques de convergence pour cette approche.

c) Comparaison des performances des itérations synchrones et asynchrones

En comparant sur le tableaux 3.6 et 3.4 les meilleures performances obtenues pour la mise en œuvre synchrone et les meilleures performances obtenues pour la mise en œuvre asynchrone, nous pouvons conclure que les itérations asynchrones sont plus performantes que les itérations synchrones pour la méthode de relaxation quand les charges de calcul sont déséquilibrées (mise en œuvre sur 2,3,4,10,12 et 16 processeurs) et moins performantes avec des charges qui sont relativement équilibrées (mise en œuvre sur 6 et 8 processeurs). Par contre, avec la méthode de sur relaxation, les itérations synchrones sont toujours plus performantes.

Méthode de relaxation									
processeurs	1	2	3	4	6	8	10	12	16
itérations	197	198	198	198	198	198	198	198	198
temps	48.01	28.05	23.95	18.78	12.06	10.37	10.43	10.45	10.55
efficacité		0.855	0.667	0.638	0.663	0.578	0.460	0.382	0.284
Méthode de sur relaxation									
itérations	89	87	87	87	87	87	87	87	87
temps	22.02	12.55	10.71	8.41	5.44	4.74	4.80	4.81	4.90
efficacité		0.875	0.685	0.653	0.673	0.579	0.458	0.380	0.280
Méthode d'agrégation-désagrégation									
itérations	80	75	75	75	75	75	75	75	75
temps	20.00	10.92	9.32	7.32	4.71	4.10	4.19	4.20	4.28
efficacité		0.915	0.714	0.682	0.702	0.608	0.477	0.396	0.291

TAB. 3.4 – *Performance des itérations synchrones sur CRAYT3E avec une technique de coloration en rouge et noir*

Méthode de relaxation									
processeurs	1	2	3	4	6	8	10	12	16
itérations	197	265	408	431	565	822	1115	997	1700
temps	48.01	27.24	23.09	16.92	12.41	11.78	9.74	8.51	8.49
efficacité		0.881	0.692	0.709	0.644	0.509	0.492	0.386	0.353
Méthode de sur relaxation									
itérations	89	180	262	299	426	576	3822	DIV	DIV
temps	22.02	18.76	15.13	11.94	9.49	8.19	35.19	DIV	DIV
efficacité		0.586	0.485	0.460	0.386	0.335	0.062	DIV	DIV

TAB. 3.5 – *Performance des itérations parallèles asynchrones sur le CRAYT3E avec la méthode de terminaison de Bertsekas*

Méthode de relaxation									
processeurs	1	2	3	4	6	8	10	12	16
itérations	197	244	369	399	560	853	1301	1127	1734
temps	48.01	25.05	20.85	15.63	12.19	11.74	10.10	9.08	7.87
efficacité		0.958	0.767	0.767	0.655	0.511	0.475	0.440	0.380
Méthode de sur relaxation									
itérations	89	149	237	268	362	635	3484	DIV	DIV
temps	22.02	15.52	13.65	10.69	8.26	8.71	25.16	DIV	DIV
efficacité		0.709	0.537	0.514	0.444	0.315	0.087	DIV	DIV

TAB. 3.6 – *Performance des itérations parallèles asynchrones sur le CRAYT3E avec la méthode de terminaison sans acquittement systématique*

3.6.2 Etude de performance sur SMP

Nous comparons dans ce sous-paragraphe les performances des itérations asynchrones avec les meilleures performances obtenues pour les itérations synchrones. Les résultats du tableau 3.7 montrent que les performances de itérations asynchrones sont nettement meilleures que celles des itérations synchrones en utilisant la méthode de relaxation et la méthode de sur relaxation. L'accès rapide en mémoire permet d'avoir des performances optimales pour les itérations asynchrones. Ces résultats confirment ceux obtenus dans les chapitres II.1 et II.2 et montrent que les performances des itérations asynchrones sont toujours excellentes sur une architecture à mémoire partagée.

	Méthode de relaxation					
	Synchrone			Asynchrone		
threads	itérations	temps	efficacité	itérations	temps	efficacité
1	197	374.67		197	374.67	
2	197	225.01	0.832	217	183.77	1.019
3	197	188.39	0.662	286	135.83	0.919
4	197	149.19	0.627	495	106.26	0.881
	Méthode de sur relaxation					
1	89	169.04		89	169.04	
2	88	109.88	0.769	97	83.11	1.016
3	88	92.21	0.611	133	63.56	0.886
4	87	71.94	0.587	236	51.24	0.824

TAB. 3.7 – *Comparaison de performance des itérations parallèles synchrones et asynchrones sur SMP*

3.6.3 Etude de performance sur un réseau de SMP

Nous comparons les performances des itérations synchrones et asynchrones sur un réseau de type Myrinet reliant des machines SMP. Dans le cas asynchrone nous mettons en œuvre la méthode de terminaison sans acquittement systimatique des messages qui a un cot plus faible que celle basée sur les acquittements. Nous rappelons que le n-uplet $(a_1,...,a_n)$ représente le nombre de threads créés sur chaque machine SMP. Les résultats du tableau 3.8 montrent que les performances des itérations synchrones sont meilleures que celles des itérations asynchrones. Ceci prouve que sur ce type d'architecture, un faible déséquilibre de charge ne permet pas d'avoir des performances meilleures pour les itérations asynchrones comparativement aux itérations synchrones. Ceci est d en fait à la faible granularité. Les performances des itérations asynchrones en utilisant la méthode de sur relaxation chutent trop sur cette architecture et on a pratiquement pas d'accélération avec une mise en œuvre sur deux machines monoprocesseurs. En effectuant la mise en œuvre sur tous les processeurs des machines SMP, les performances des itérations parallèles augmentent; ce qui confirme l'intérêt de l'utilisation des threads.

	Méthode de relaxation					
	Synchrone			Asynchrone		
threads	itérations	temps	efficacité	itérations	temps	efficacité
(1,0)	197	311.27		197	311.27	
(1,1)	197	179.63	0.866	304	209.51	0.742
(2,1)	198	168.48	0.615	350	165.40	0.627
(2,2)	197	103.87	0.749	579	105.73	0.735
	Méthode de sur relaxation					
(1,0)	89	142.19		89	142.19	
(1,1)	88	81.27	0.874	229	158.12	0.449
(2,1)	88	74.81	0.633	234	113.24	0.418
(2,2)	87	46.23	0.768	369	67.73	0.524

TAB. 3.8 – *Comparaison de performance des itérations parallèles synchrones et asynchrones sur un réseau myrinet de SMP*

De manière générale nous constatons que les résultats relatifs aux mises en œuvre sur SMP ou sur réseau de SMP font apparaître des schémas asynchrones convergents. Les raisons de la convergence sont certainement à chercher dans le fait que les expérimentations ont été faites pour un faible nombre de processeurs et pour des charges relativement équilibrées ce qui implique que les retards seront généralement bornés par des valeurs assez faibles; on peut s'en convaincre en comparant le nombre maximal d'itérations des schémas asynchrones avec le nombre d'itérations des schémas synchrones. Il est à noter aussi que sur les SMP les accès aux données sont très rapides et que le débit du réseau Myrinet est important.

3.7 Conclusion

Nous avons étudié dans ce chapitre les performances des itérations parallèles pour résoudre un système Markovien modélisant un réseau téléphonique à commutation de circuit. Nous avons mis en œuvre les méthodes de relaxation, de sur relaxation et d'agrégation-désagrégation qui sont basées sur le schéma de Gauss-Seidel. Nous avons présenté deux techniques de parallélisation du schéma Gauss-Seidel qui permettent de conserver un bon taux de convergence. La mise en œuvre asynchrone a été effectuée en utilisant deux algorithmes de terminaison : le premier est basé sur l'acquittement de tous les messages reus et le deuxième est basé sur les messages d'activations. Les résultats numériques ont montré que la parallélisation avec la technique de coloration en rouge et noir permettent d'avoir de bonnes performances pour les itérations parallèles synchrones. Les performances des itérations asynchrones en utilisant l'algorithme de terminaison basé sur les messages d'activation sont nettement meilleures que celles en utilisant l'algorithme de terminaison basé sur les acquittements systématiques. Les itérations asynchrones sont plus performantes que les itérations synchrones sur le une SMP. Sur une architecture à mémoire distribué telle que le CRAY T3E, les performances des itérations asynchrones sont meilleures pour la méthode de relaxation quand les charges de calcul sont déséquilibrées et moins bonnes quand les charges de calcul sont relativement équilibrées. Avec la méthode de sur relaxation les itérations synchrones sont plus performantes que les itérations asynchrones. Enfin sur un réseau de SMP, les performances des itérations asynchrones ne sont pas bonnes essentiellement lorsque la granularité est faible et que les tches de calcul sont équilibrées.

Chapitre 4

Conclusion générale

Les travaux effectués dans ce mémoire concernent l'étude et la mise en œuvre des algorithmes itératifs parallèles synchrones, asynchrones et asynchrones avec communication flexible sur trois types d'architectures parallèles : un supercalculateur CRAY T3E à mémoire distribuée en utilisant les deux bibliothèques de passage de message MPI et SHMEM, une machine multiprocesseur symétrique "SMP" à mémoire partagée en utilisant les threads Posix et un réseau de SMP en combinant la bibliothèque des threads Posix et la bibliothèque de passage de message MPICH. Nous avons proposé des mises en œuvre parallèles originales pour toutes les architectures. En ce qui concerne plus particulièrement le réseau de SMP nous avons considéré deux niveaux de parallélisme : un premier niveau entre les différents "threads" ou fils au sein d'une même machine SMP et un deuxième entre les "threads" distants sur les différentes machines SMP du réseau. La mise en œuvre synchrone a été effectuée de deux faons : sur une architecture de type passage de message, nous avons synchronisé les processus au moment de la réception de données et ceci en utilisant une primitive bloquante pour la réception et une primitive non bloquante pour l'envoi, alors que sur une architecture à mémoire partagée telle qu'une SMP, la synchronisation entre les threads a été effectuée en mettant en œuvre une barrière de synchronisation en utilisant les variables Mutex et les variables conditions de Posix. Nous avons montré, dans le cas de l'utilisation de la bibliothèque SHMEM, que la synchronisation entre les processus ne peut pas se faire en utilisant les primitives d'envoi et de réception, car elles sont de nature asynchrone. Dans le cas asynchrone et asynchrone avec communication flexible, nous avons montré que la mise œuvre est plus naturelle en utilisant SHMEM sur le CRAY T3E et les threads sur une SMP ou un réseau de SMP.

Les résultats expérimentaux ont montré que les performances des algorithmes itératifs parallèles dépendent de plusieurs facteurs. Un des facteurs les plus importants est le déséquilibre des charges de calcul qui est propre à la nature du problème et à la méthode de résolution utilisée. Dans le cas où les charges de calcul sont déséquilibrés, la synchronisation des différents processeurs cause une chute de performance due aux temps d'attente.

Les performances des itérations parallèles asynchrones sont nettement meilleures que celles des itérations synchrones quand le déséquilibre des charges de calcul est important c'est ce que l'on constate notamment dans le cas de l'utilisation de la méthode de type gradient pour la résolution d'un problème d'optimisation de type flot dans les réseaux. Ceci montre l'intérêt des itérations asynchrones dans le cas d'une charge très déséquilibrée. Avec une charge relativement équilibrée comme dans le cas de la méthode à deux niveaux pour résoudre un problème de commande optimale ou du système markovien), les performances des itérations synchrones et asynchrones sont très proches. Dans ce cas là, d'autres facteurs ont particulièrement d'influence afin de départager l'une ou l'autre des itérations parallèles. Sur une architecture à mémoire partagée, l'accès à la valeur courante du vecteur itéré par une simple lecture de la mémoire commune permet d'avoir une mise en œuvre des itérations asynchrones avec communication flexible plus efficace qu'une mise en œuvre synchrone qui nécessite la mise en place d'une section critique. Ceci reste valable aussi sur une architecture parallèle à mémoire distribuée telle que le CRAY T3E en utilisant la primitive de communication shmem_get() qui a un très faible temps de latence et permet d'accéder très rapidement à la valeur courante du vecteur itéré réactualisée par les autres processeurs. Dans le cas où l'environnement ne permet pas à un accès distant rapide aux données, la mise en œuvre synchrone devient plus performante. Les résultats expérimentaux obtenus en examinant les différents problèmes ont montré que les performances des itérations asynchrones avec communication flexible sont optimales et meilleures que celles des itérations asynchrones classiques quand la granularité est assez élevée.

De manière plus générale, l'ensemble des expérimentations numériques effectuées met en évidence que pour les applications de très grande dimension et les méthodes numériques associées autorisent un grand degré de parallélisme. Comme le fait remarquer Daniel Szyld, l'asynchronisme est certainement l'unique solution pour mettre en œuvre les méthodes itératives parallèles sur des machines à parallélisme massif. D'autre part pour une question de cot, l'asynchronisme est une excellente solution pour mettre en œuvre les méthodes itératives parallèles sur des "clusters" ou réseaux de machines hétérogènes qui sont de plus en plus utilisés.

Le travail effectué dans ce mémoire ouvre sur plusieurs perspectives. En ce qui concerne les problèmes de flot, il serait intéressant de mettre en œuvre les itérations parallèles pour différentes décompositions du graphe et de chercher une décomposition optimale qui permette d'obtenir les meilleures performances. La résolution du problème de commande optimale peut se faire avec d'autres méthodes de discrétisation et notamment la méthode de Runge Kutta. Il serait dans ce cas très intéressant d'étudier la convergence des itérations asynchrones avec communication flexible et de mettre en œuvre la résolution sur les différentes architectures envisagées. Enfin, le travail de recherche examiné dans ce mémoire peut être élargi en considérant d'autres types d'applications comme la résolution d'équations aux dérivées partielles discrétisées et d'autres architectures comme l'ORIGIN 2000 où la NEC SX5 pour le problème de commande optimale et l'étude des systèmes markoviens.

Annexe A

Rappels mathematiques

Nous présentons dans cette annexe un ensemble de définitions mathématiques et des résultats qui ont été utilisés dans les différents chapitres de ce mémoire

A.1 Rappels d'algèbre linéaire

Nous considérons le système d'équations linéaires :

$$Ax^* = 0. \tag{A.1}$$

Définition A.1.1 : *Toute matrice dont les coefficients diagonaux sont strictement positifs et dont les coefficients hors diagonaux sont négatifs ou nuls est appelé une Z-matrice.*

Définition A.1.2 : *L'ordre partiel naturel dans R^n, où encore ordre composante par composante est défini de la manière suivante :*

$$\textit{Pour tout } x,y \in R^n,\ x \leq y \textit{ si et seulement si } x_i \leq y_i,\ i = 1,\ldots,n. \tag{A.2}$$

Définition A.1.3 : *Soit $x_i,y_i \in R$, tels que $x_i \leq y_i$. Alors un segment d'ordre dans R noté $\langle x_i,y_i\rangle_i$ est défini comme suit :*

$$\langle x_i,y_i\rangle_i = \{z_i \in R \mid x_i \leq z_i \leq y_i\}.$$

De manière analogue, soit $x,y \in R^n$, tels que $x \leq y$. Alors un segment d'ordre dans R^n noté $\langle x,y\rangle$ est défini de la manière suivante :

$$\langle x,y\rangle = \{z \in R^n \mid x \leq z \leq y\}. \tag{A.3}$$

On peut introduire un ordre partiel sur l'espace des matrices (n,m) analogue à celui défini dans l'équation (A.2). Pour tout $A,B \in L(R^n,R^m)$ nous dirons que $A \leq B$ si et seulement si $a_{il} \leq b_{il}$, $i = 1,\ldots,n$, $l = 1,\ldots,m$.

Définition A.1.4 : *Une matrice $A \in L(R^n,R^m)$ est non négative si*

$$A \geq 0.$$

Définition A.1.5 : *Toute Z-matrice N telle que l'inverse de N existe et est non négative est appelée une M-matrice.*

Définition A.1.6 : *Une matrice N est appelée H-matrice si la matrice constituée par les éléments diagonaux de N et moins la valeur absolue des éléments hors-diagonaux de N est une M-matrice.*

A.2 Notions de M-fonctions

Considérons dans ce cas le système d'équations non linéaires :

$$\mathcal{A}(x^*) = 0 \tag{A.4}$$

Définition A.2.1 : *Une application $\mathcal{A}$ de R^n dans R^n est diagonale monotone croissante si pour tout $x \in R^n$ et $i \in \{1,...,n\}$, les fonctions $\mathcal{A}_i^i$ définies comme suit :*

$$\begin{cases} \mathcal{A}_i^i : \{t \in R \mid x + te^i \in R^n\} \to R, \\ \\ \mathcal{A}_i^i(t) = \mathcal{A}_i(x + te^i), \end{cases} \tag{A.5}$$

sont monotones croissantes, $\mathcal{A}_i$ étant la i-ème composante de l'application $\mathcal{A}$ et $e^i \in R^n$, $i = 1,\ldots,n$, le i-ème vecteur unitaire de la base canonique.

Définition A.2.2 : *Une application $\mathcal{A}$ de R^n dans R^n est diagonale monotone strictement croissante si pour tout $x \in R^n$ et $i \in \{1,...,n\}$, les fonctions $\mathcal{A}_i^i$ définies comme suit :*

$$\begin{cases} \mathcal{A}_i^i : \{t \in R \mid x + te^i \in R^n\} \to R, \\ \\ \mathcal{A}_i^i(t) = \mathcal{A}_i(x + te^i), \end{cases} \tag{A.6}$$

sont strictement monotones croissantes.

Définition A.2.3 : *Une application $\mathcal{A}$ de R^n dans R^n est hors-diagonale monotone décroissante si pour tout $x \in R^n$ et $i,l \in \{1,...,n\}$, $l \neq i$, les fonctions $\mathcal{A}_i^l$ définies comme suit :*

$$\begin{cases} \mathcal{A}_i^l : \{t \in R \mid x + te^l \in R^n\} \to R, \\ \mathcal{A}_i^l(t) = \mathcal{A}_i(x + te^l), \end{cases} \tag{A.7}$$

sont monotones decroissantes.

Définition A.2.4 : *Une application $\mathcal{A}$ de R^n dans R^n est inverse monotone croissante si*

$$\mathcal{A}(x) \leq \mathcal{A}(y) \text{ pour tout } x,y \in R^n \text{ implique que } x \leq y. \tag{A.8}$$

Définition A.2.5 : *Une application $\mathcal{A}$ de R^n dans R^n est une M-fonction au sens de Rheinboldt (cf. [Rhe70] et [OR70]) si $\mathcal{A}$ est hors-diagonale monotone décroissante et inverse monotone croissante.*

Par la suite nous utiliserons la notation suivante. Pour toute suite $\{x(j)\} \subset R^n$, $\lim_{j\to\infty} x(j) = +\infty$ ($\lim_{j\to\infty}\ x(j) = -\infty$) si $\lim_{j\to\infty} x_i(j) = +\infty$ ($\lim_{j\to\infty}\ x_i(j) = -\infty$) pour au moins un indice $i \in \{1,...,n\}$.

Définition A.2.6 : *L'application $\mathcal{A} : R^n \to R^n$ est coercive pour l'ordre si pour toute suite $\{x(j)\} \subset R^n$, telle que $x(j) \leq x(j+1)$, $j = 0,1,\ldots,$ et $\lim_{j\to\infty} x(j) = +\infty$, on a $\lim_{j\to\infty}\ \mathcal{A}(x(j)) = +\infty$ et pour toute suite $\{x(j)\} \subset R^n$, telle que $x(j) \geq x(j+1), j = 0,1,\ldots,$ et $\lim_{j\to\infty} x(j) = -\infty$, on a $\lim_{j\to\infty} \mathcal{A}(x(j)) = -\infty$.*

Théorème A.2.1 : *Soit $\mathcal{A} : R^n \to R^n$ une M-fonction continue. Alors $\mathcal{A}$ est surjective si et seulement si $\mathcal{A}$ is coercive pour l'ordre.*

Remarque : Soit $\mathcal{A} : R^n \to R^n$ une M-fonction continue surjective. Alors le système (A.4) possède une solution unique.

Théorème A.2.2 : *(cf. [Mie86]) F est monotone croissante sur $\prod_{i=1}^{\alpha} R^{n_i}$. c'est à dire :*

$$F(x) \leq F(y) \text{ pour tout } x \leq y,\ x,y \in \prod_{i=1}^{\alpha} R^{n_i}.$$

Définition A.2.7 : *(cf. [Rhe70] et [OR70]) Un vecteur $x \in R^n_+$ est une $\mathcal{A}$-sur solution si $\mathcal{A}(x) \geq 0$.*

Définition A.2.8 : *(cf. [Rhe70] et [OR70]) Un vecteur $x \in R^n_+$ est une $\mathcal{A}$-sous-solution si $\mathcal{A}(x) \leq 0$.*

Définition A.2.9 : *Un vecteur $x \in R^n$ est une sur solution si $F(x) \leq x$.*

Définition A.2.10 : *Un vecteur $x \in R^n$ est une sous-solution si $F(x) \geq x$.*

Théorème A.2.3 : *Si $x \in R^n$ est une $\mathcal{A}$-sur solution alors x est une sur solution. Si $x \in R^n$ est une $\mathcal{A}$-sous-solution alors x est une sous-solution.*

Remarque : En général la réciproque est fausse. Toutefois dans le cas linéaire si F correspond à l'application de Jacobi associé à une décomposition par point de la matrice $\mathcal{A}$ on a alors $\alpha = n$ et $R^{n_i} = R$ pour tout i et $x \geq F(x)$ est équivalent à $\mathcal{A}x \geq 0$.

A.3 Notions de H-accrétivité

Soit α un entier naturel et soit E un espace de Banach réflexif produit fini d'une famille d'espaces de Banach $\{E_i\}$, $i \in \{1,...,\alpha\}$, de normes respectives $\|.\|_i$. On considère maintenant un problème de la forme :

$$\Lambda^d(x) + \Lambda(x) \ni 0,\, x \in E. \tag{A.9}$$

où l'application $\Lambda \ : \ D(\Lambda) \subset E \to E$ est une application univoque et l'application Λ^d est une multi-application diagonale. On pose ;

$$D(\Lambda) = \prod_{i=1}^{\alpha} D_i,$$

où $D_i \in E_i$ et on décompose l'application $\Lambda(x)$ de la manière suivante :

$$\Lambda(x) = \{\Lambda_1(x),...,\Lambda_\alpha(x)\}.$$

Pour tout $w \in D(\Lambda)$, et tout $i,l \in \{1,...,\alpha\}$, soit Λ_{li}^w l'application de D_i dans E_l qui à tout $x_i \in D_i$ associe :

$$\Lambda_{li}^w(x_i) = \Lambda_l(x_i; w).$$

Remarque : Si l'on considère la décomposition du système (A.9) par blocs, on note que l'application Λ_{li}^w est relative à l'interaction entre le l-ème bloc et le i-ème bloc de l'application Λ si $l \neq i$ et au i-ème bloc diagonal si $l = i$. Pour tout $i \in \{1,...,\alpha\}$, soit E_i^*, muni de la norme $\|.\|_i^*$, le dual de E_i. En applicant le théorème de Hahn-Banach on peut considérer la multiapplication G_l de E_l dans E_l^* qui à tout $x_l \in E_l$ associe $G_l(x_l)$ tel que :

$$\exists g_l \in G_l(x_l),(x_l,g_l)_l = \mid x_l \mid_l^2 \ \text{ et } \ \mid x_l \mid_l = \mid g_l \mid_l^* ,$$

où $(.)_l$ désigne le produit de dualité entre E_l et E_l^*

Hypothèse A.3.1 : *Il existe une Z-matrice notée N, de type (α,α), et de coefficients n_{li} telle que : pour tout $i \in \{1,...,\alpha\}$ et tout $x,x' \in D(\Lambda), \exists g_l \in G_l(x_l - x'_l)$ satisfaisant :*

$$(\Lambda_l(x) - \Lambda_l(x'),g_l)_l \geq \sum_{i=1}^{\alpha} n_{li}\|x_l - x'_l\|_l.\|x_i - x'_i\|_i. \tag{A.10}$$

Définition A.3.1 : *Sous l'Hypothèse A.3.1 on dit que la matrice N est une minorante Z-accrétive de Λ. Si de plus N est une M matrice on dit alors que c'est une minorante M-accrétive de Λ.*

Hypothèse A.3.2 : *Pour tout $l \in \{1,...,\alpha\}$, le domaine D_l est quasi-dense au sens de Kato (cf. [Kat64]).*

Hypothèse A.3.3 : *Pour tout $l \in \{1,...,\alpha\}$ et $w \in D(\Lambda)$, l'application Λ_{ll}^w est hémi-continue sur D_l.*

Remarque : L'Hypothèse A.3.2 traduit le fait que D_l est un ouvert dense dans E_l. Les Hypothèses A.3.2 et A.3.3 correspondent à des propriétés de régularité d'une part du domaine $D(\Lambda)$ et d'autre part, de l'application Λ.

Théorème A.3.1 : *Sous les Hypothèses A.3.2 et A.3.3, l'Hypothèse A.3.1 est équivalente à l'ensemble des conditions suivantes.*

$$\forall l \in \{1,...,\alpha\}, \forall w \in D(\Lambda), \forall x_l, x_l' \in D_l, \forall g_l \in G_l(x_l - x_l'), (\Lambda_{ll}^w(x_l) - \Lambda_{ll}^w(x_l'), g_l)_l \geq n_{ll} \|x_l - x_l'\|_l^2, \tag{A.11}$$

$$\forall l,i \in \{1,...,\alpha\} \mid i \neq l, \forall w \in D(\Lambda), \forall x_i, x_i' \in D_i, \|\Lambda_{li}^w(x_i) - \Lambda_{li}^w(x_i')\|_l \leq -n_{li} \|x_i - x_i'\|_i. \tag{A.12}$$

Remarque : Les conditions (A.11) et (A.12) traduisent respectivement, une condition d'accrétivité forte pour le l-ème sous-problème diagonal et une condition de Lipschitz pour les termes de couplage entre les blocs l et i.

Définition A.3.2 : *Sous les Hypothèses A.3.2, A.3.3, et A.3.1 si N est une minorante M-accrétive de l'application Λ on dira que Λ est H-accrétif.*

Définition A.3.3 : *Soit Λ^d une multi-application diagonale telle que*

$$\Lambda^d(x) = \{\Lambda_1^d(x_1),...,\Lambda_\alpha^d(x_\alpha)\} \subset E;$$

pour tout $l \in \{1,...,\alpha\}$ l'application Λ_l^d qui est une multi-application de $D(\Lambda_l^d) \subset E_l$ dans E_l est m-accrétive si

$$\forall x_l, x_l' \in D(\Lambda_l^d), \forall \eta_l \in \Lambda_l^d(x_l), \forall \eta_l' \in \Lambda_l^d(x_l'), \exists g_l \in G_l(x_l - x_l') \mid (\eta_l - \eta_l', g_l)_l \geq 0. \tag{A.13}$$

A.4 Applications contractantes

Considérons le problème de point fixe suivant :

$$F(x^*) = x^*, \tag{A.14}$$

où F est une application de domaine $D(F) \in E$ dans E, avec E un espace de Banach.

Définition A.4.1 : *Soit α un entier naturel et soit E un espace vectoriel produit d'une famille d'espaces de Banach $\{E_i\}$, $i \in \{1,...,\alpha\}$, de normes respectives $\|.\|_i$. L'application ψ de $E = \prod_{i=1}^{\alpha} E_i$ dans R_+^α qui à tout $x \in E$ fait correspondre :*

$$\psi(x) = (\|x_1\|_1,...,\|x_\alpha\|_\alpha), \tag{A.15}$$

est une norme vectorielle canonique sur E.

Définition A.4.2 : *On appelle norme vectorielle type sur R^n, l'application qui à tout $x \in R^n$ associe $\psi(x) =\mid x \mid$ où $\mid x \mid$ est le vecteur de R_+^n obtenu en remplaçant chaque composante de x par sa valeur absolue.*

Définition A.4.3 : *Soit $x^\star \in D(F)$ et T une matrice réelle de dimension (α,α). Si F et T vérifient :*

$$\psi(F(x^\star) - F(x)) \leq T\psi(x^\star - x),\ \forall x \in D(F), \tag{A.16}$$

alors T est une majorante linéaire de F en $x^\star$ pour la norme vectorielle ψ. Si F et T vérifient :

$$\psi(F(x) - F(x')) \leq T\psi(x - x'),\ \forall x,x' \in D(F), \tag{A.17}$$

alors T est une majorante linéaire de F pour la norme vectorielle ψ. Dans ce dernier cas si T est non négative on l'appelle matrice de Lipschitz.

Définition A.4.4 : *Si T est une majorante linéaire de F en $x^\star$ pour la norme vectorielle ψ de rayon spectral $\rho(T) < 1$, alors F est contractante en $x^\star$ pour la norme vectorielle ψ et T est une matrice de contraction de F en $x^\star$. De plus si T est une majorante linéaire de F pour la norme vectorielle ψ de rayon spectral strictement inférieur à 1, alors F est contractante sur $D(F)$ pour la norme vectorielle ψ, on dit aussi que F est une P-contraction sur $D(F)$ et T une matrice de contraction de F.*

Nous pouvons énoncer le résultat suivant sur l'existence d'une solution du problème de point fixe (A.14) (cf. [OR70]).

Théorème A.4.1 : *Si l'application F est contractante pour la norme vectorielle ψ sur le domaine fermé $D(F) \subset E$ et $F(D(F)) \subset D(F)$. Alors F admet un point fixe unique $x^* \in D(F)$.*

Soit T une matrice réelle de dimension (α,α) qui est non négative et de rayon spectral $\rho(T) < 1$. Pour tout $\nu \in]\rho(T),1[$ il existe (cf. [OR70]) un vecteur w^ν de composantes $w_i^\nu > 0$, $\forall i \in \{1,...,\alpha\}$ vérifiant :

$$| w^\nu |_2 = 1 \text{ et } Tw^\nu \leq \nu w^\nu, \tag{A.18}$$

où $| \ . \ |_2$ désigne la norme Euclidienne. De plus si T est irréductible, on peut prendre $\nu = \rho(T)$ et $w^{\rho(T)}$ est vecteur propre de T. Nous introduisons sur $E = \prod_{i=1}^{\alpha} E_i$ la norme pondérée :

$$\| x \|_T^w = \max_{i=1,\ldots,\alpha} \frac{\|x_i\|_i}{w_i^\nu}. \tag{A.19}$$

On a alors le résultat suivant (cf. [Mie75a]).

Théorème A.4.2 : *Soit F une application de $D(F) \subset E$ à valeurs dans E, T une matrice réelle de dimension (α,α) qui est non négative et de rayon spectral $\rho(T) < 1$ et $x,x' \in D(F)$, tels que :*

$$\psi(F(x) - F(x')) \leq T\psi(x - x'), \tag{A.20}$$

on a alors :

$$\|F(x) - F(x')\|_T^w \leq \nu \|x - x'\|_T^w. \tag{A.21}$$

Une application F contractante sur $D(F)$ pour la norme vectorielle ψ de matrice de contraction T est contractante sur $D(F)$ pour la norme pondérée $\|,\|_T^w$.

A.5 Méthode du gradient conjugué

Dans cette section nous présentons quelques éléments théoriques sur la méthode du gradient conjugué qui est très utilisée pour résoudre les problèmes d'optimisation et pour la résolution des systèmes d'équations. Nous allons présenter cette méthode dans le cadre de l'optimisation d'un problème quadratique, par la suite nous allons expliciter les résultats pour le cas général. Etant donnée le problème suivant :

$$\text{minimiser } f(x) = \frac{1}{2} x^T Q x - b^T x, x \in \mathbb{R}^n, \tag{A.22}$$

où Q est une matrice définie positive. La résolution de (A.22) revient à résoudre le système d'équation linéaire :

$$Qx = b. \tag{A.23}$$

Etant donnée une matrice définie positive Q, les vecteurs non nuls $d^1,...,d^k$ sont Q-conjugués, si :

$$d^{j^T} Q d^j = 0, \forall i,j \text{ tel que } i \neq j, \tag{A.24}$$

les vecteurs $d^1,...,d^k$ sont alors linéairement indépendants, car si par exemple d^k est linéairement dépendant de $d^1,...,d^{k-1}$, alors il existe $(\alpha^1,...,\alpha^{k-1})$ tel que :

$$d^k = \alpha^1 d^1 + ... + \alpha^{k-1} d^{k-1} \tag{A.25}$$

En multipliant à gauche l'égalité (A.25) par $d^{k^T}Q$, nous trouvons
$d^{k^T}Qd^k = \alpha^1 d^{k^T}Qd^1 + ... + \alpha^{k-1}d^{k^T}Qd^{k-1} = 0$,
ce qui est impossible car $d^k \neq 0$ par hypothèse. Donc le nombre de vecteurs Q-conjugués est au plus égal à n. Si on engendre une suite de vecteurs Q-conjugués construits à partir du gradient de f, alors on pourra construire un algorithme qui converge en au plus n itérations.
Etant donnée un ensemble de directions Q-conjugués $d^0,...,d^{k-1}$, on peut définir la méthode des directions conjuguées pour résoudre le problème (A.1) par :

$$x^{k+1} = x^k + \alpha^k d^k, \tag{A.26}$$

où x^0 est un vecteur donné de $\mathbb{R}^n$ et α^k est déterminée en minimisant $f(x^k + \alpha^k d^k)$, soit :

$$f(x^k + \alpha^k d^k) = min_\alpha f(x^k + \alpha d^k) \tag{A.27}$$

Nous utilisons pour la suite la notation $g^k = \nabla f(x^k) = Qx^k$. Nous avons le résultat suivant :

Proposition A.5.1 *si $x^1,...,x^n$ sont des vecteurs engendrés par la méthode des directions conjuguées, alors pour tout $k = 0,1,...,n-1$, ,*

$$g^{k+1^T} d^i = 0, \forall i = 0,...,k. \tag{A.28}$$

Etant donnée un ensemble de vecteurs $\xi^0,...,\xi^{n-1}$ linéairement indépendants, nous pouvons construire un ensemble de directions mutuellement conjuguées $d^0,...,d^{n-1}$ de la faon suivante :

$$d^0 = \xi^0 \tag{A.29}$$

$$d^i = \xi^i + \sum_{j=0}^{i-1} c_{ij} d^j, \text{ pour tout } i = 1,2,...,n-1, \tag{A.30}$$

où les coefficients c_{ij} sont choisis de telle sorte que d^i soit Q-conjugué des précédentes directions $d^{i-1},...,d^0$.

$$c_{ij} = -\frac{\xi^{i^T} Q d^j}{d^{j^T} Q d^j}, \forall i = 1,...,n-1. \tag{A.31}$$

(A.29) et (A.31) montrent que le sous-espace engendré par $d^0,...,d^i$ est égal au sous-espace engendré par $\xi^0,...,\xi^i$ pour tout $i = 0,...,n-1$.
La méthode du gradient conjugué se base sur la procédure de détermination des directions conjuguées appliquée à l'ensemble des vecteurs $\xi^0 = -g^0,...,\xi^{n-1} = -g^{n-1}$. L'égalité (A.30)

peut se simplifier davantage pour cet ensemble de directions conjuguées et on a le résultat suivant :

Proposition A.5.2 *Les directions de la méthode du gradient conjugué sont engendrées par :* $d^0 = -g^0$,
$d^k = -g^k + \beta^k d^{k-1}$, $k = 1,...,n-1$,
où β^k *est donnée par :* $\beta^k = \frac{{g^k}^T g^k}{{g^{k-1}}^T g^{k-1}}$.

La méthode se termine avec une solution optimale après au maximum n itérations. La méthode du gradient conjugué commence au point x^0 avec $d^0 = -g^0$, $x^1 = x^0 + \alpha^0 d^0$, où α^0 est déterminée par une recherche unidirectionnelle du minimum de $f(x^0 + \alpha^0 d^0)$.
Etant calculé x^1, nous pouvons calculer $g^1 = \nabla f(x^1)$, $\beta^1 = \frac{{g^1}^T g^1}{{g^0}^T g^0}$ et $d^1 = -g^1 + \beta^1 d^0$.
Avec les valeurs de x^1 et de d^1, on peut déterminer α^1 de la même manière que pour α^0 et ainsi de suite jusqu'à ce que le vecteur d^k s'annule.
L'algorithme du gradient conjugué est le suivant :

x^0 donné
$d^0 = -g^0$
tant que non Terminé **faire**
  $\alpha^k | f(x^k + \alpha^k d^k) = min_\alpha f(x^k + \alpha d^k)$
  $x^{k+1} = x^k + \alpha^k d^k$
  $\beta^{k+1} = \frac{\nabla f^T(x^{k+1}) \nabla f(x^{k+1})}{\nabla f^T(x^k) \nabla f((x^k)}$
  $d^{k+1} = -\nabla f(x^{k+1}) + \beta^{k+1} d^k$
  si $(d^{k+1} < \epsilon)$ **alors**
    Terminé
  fin si
fin tant que

Cet algorithme peut se généraliser pour le cas d'un problème non quadratique. En effet en minimisant $f(x^k + \alpha d^k)$ on a :$\nabla f(x^{k+1}) d^k = 0$, et comme $d^{k+1} = -\nabla f^{k+1} + \beta^{k+1} d^k$ alors $\nabla f(x^{k+1})^T d^{k+1} = -||\nabla f(x^{k+1})||^2 + \beta^{k+1} \nabla f(x^{k+1})^T d^k = -||\nabla f(x^{k+1})^T d^k||^2 < 0$, et par suite d^{k+1} est une direction de descente.

Bibliographie

[AM95] P. Amodio, F. Mazzia. *A parallel Gauss-Seidel method for block tridiagonal linear systems.* Siam J. Sci. Comput, 6:1451-1561, 1995.

[Bau78] G. M. Baudet. *Asynchronous iterative methods for multiprocessors.* Journal Assoc. Comput. Mach., vol. 2, pages 226–244, 1978.

[Bah91] J. Bahi. *Algorithmes asynchrones pour des systmes diffrentiels algbriques, simulation numriques sur des ensembles de circuits lectriques.* Thse de Doctorat de l'Universit de Franche Comt, 1991.

[Bah96] J. Bahi. *Asynchronous Runge-Kutta methods for differential-algebraic systems.* In Joubert et al., editor, Advances in Parallel Computing, volume 11, pages 205–212. Elsevier Science B.V., North Holland, 1996.

[Bah98a] J. Bahi. *Algorithmes parallèles asynchrones pour des systèmes singuliers.* CRAS série 1, 326:1421–1425, 1998.

[Bah98b] J. Bahi. *Mthodes itratives dans des espaces produits. Application au calcul parallle.* Habilitation diriger des recherches, Universit de Franche-Compt, 1998.

[Bah99a] J. Bahi. *Asynchronous iterative algorithms for nonexpansive linear systems.* Journal of Parallel and distributed Computing 59:1–20, 1999.

[Bah99b] J. Bahi. *Parallel chaotic algorithms for singular linear systems.* Parallel Algorithms and Applications, 14:19–35, 1999.

[BeE87] D. P. Bertsekas, D. El Baz. *Distributed asynchronous relaxation methods for convex network flow problems.* SIAM J. Control and Optimization, 25:74–85, 1987.

[Ber82] D. P. Bertsekas. Distributed dynamic programming. *IEEE Trans. Automat. Control*, 27:610–616, 1982.

[Ber83] D. P. Bertsekas. *Distributed asynchronous computation of fixed points*. Mathematical Programming, vol. 27, pages 107–120, 1983.

[Ber95] D. P. Bertsekas. *Nonlinear Programming.* Athena Scientific, Belmont Mass., 1995.

[BACEB84] J. Bernussou, G. Authie, J.L. Calvet, D. El Baz. *Routage distribu dans les rseaux de communication.* RAIRO, APII, 18:161-172, 1984.

[BCEZ95] D. P. Bertsekas, D. Castanon, J. Eckstein, and S. Zenios. *Parallel computing in network optimization.* Handbooks in Operation Research and Management Science, 7:331–399, 1995.

[BFK99] P. Buchholz, M. Fischer, P. Kemper. *Distributed steady state analysis using Kronecker algebra.* 3^{rd} International Workshop on the Numerical Solution of Markov Chains, (NSMC'99), Saragosse (Espagne), 76-98, 1999.

[BGP98] J. Briat, I. Ginzburg, M. Pasin. *Athapascan-0 User Manual.* Version 2.4.13-mar 10,1998.

[BGM96] J. Bahi, E. Griepentrog, J.C. Miellou. *Parallel treatment of a class of differential-algebraic systems.* SIAM Journal on Numerical Analysis 23 : 5 : 1969-1980, (1996).

[BLA87] J. Bernussou, F. Le Gall, G. Authie. *About some iterative synchronous and asynchronous methods for Markov chain distribution computation.* IFAC'87- 10th word congress, Munich, 1987.

[BMPS95] R. Bru, V. Migallón, J. Penads, D. B. Szyld. *Parallel, synchronous and asynchronous two-stage multisplitting methods.* Electronic Transaction on Numerical Analysis. 3:24-38, 1995.

[BMR97] J. Bahi, J.C. Miellou, K. Rhofir. *Asynchronous multisplitting methods for nonlinear fixed point problems.* Numerical Algorithms, 15 :3 :4 : 315-345, 1997.

[BP86] G.P. Barker et R.J. Plemmons. *Convergent iterations for computing stationary distributions of Markov chains.* SIAM J. Alg. Disc. Meth., 7,3:390-398, 1986.

[BRM97] J. Bahi, K. Rhofir. J.C. Miellou. *Parallel solution of linear DAEs by mutisplitting waveform relaxation methods.* Linear Algebra and applications, a paratre, 2000.

[BT89] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation, Numerical Methods.* Prentice Hall Englewood Cliffs N.J., 1989.

[BT91] D.P. Bertsekas, J. Tsitsiklis. *Some aspects of parallel and distributed iterative algorithms survey.* Automatica, vol 27:3–21, 1991.

[Ca81] J.L. Calvet. *Le concept de partition en analyse numrique. Applic-tion aux problmes de commande.* Note interne LAAS, CDH N. 81, I. 21, 1981.

[Ca84] J.L. Calvet. *Mthodes itratives et partitionnement application aux problmes de commande.* Thse d'Etat, 1984.

[CaDG90] J.L. Calvet, J. Dantos De Melo, J.M. Garcia. *A fast parallel dynamic programming algorithm for optimal control.* American Control Confernce, San Diego (USA), 2930-2931, 1991.

[Cal97] C. Calmettes. *Rsolution numrique de systmes markoviens, parallisation de la mthode d'Arnoldi.* Thse de Doctorat de l'Universit Pau Sabatier, 1997.

[Car99] A.D. Carissimi. ATHAPASCAN-0 : Exploitation de la multiprogrammation lgre sur grappes de multiprocesseurs. Thse de Doctorat de l'Institut National Polytechnique de Grenoble, 1999.

[Cas93] C.G. Cassandras. *Modeling and performance analysis.* Aksen Associates Incorporeted Publishers, Irwin Edition, 1993.

[CaV92] J.L. Calvet, G, Viargues. *Parallelism in dynamic programming : on issue to feedback control.* 1^{st} IEEE Conference on Control Applications Dayton (USA), 13-16, 1992.

[CDG97] J. Chergui, I. Dupays, D. Girou. *Message Passing Interface.* IDRIS, Cours interne, 1997.

[CG96] J. Chergui, E. Gondet. *Les fonctions SHMEM : Optimisation des communications sur T3D/T3E.* IDRIS, Cours interne, 1996.

[Che80] N.E. Cheik Obeid. *Commande optimale en boucle ferme par intermittence de systmes multivariables. Application un processus thermique.* Thse de 3me cycle, Universit de Besanon, 1980.

[CL85] K.M. Chandy and L. Lamport. *Distributed snapshots: Determining global state of distributed systems.* ACM Trans. on Computer Systems, 3:63–75, 1985.

[CM69] D. Chazan and W. Miranker. *Chaotic relaxation.* Linear Algebra Appl., vol. 2, pages 199–222, 1969.

[CMP98] M.J. Castel, V. Migalln, J. Penads. *Convergence of non-stationary Parallel multisplitting methods for hermitian positive definite matrices.* Mathematics of Computations, 67 :209-220, 1998.

[Com76] P. Compte. *Algorithme de relaxation-dcentralisation.* Thse de 3me cycle, Universit de Besanon, 1976.

[CT93] M. Cosnard, D. Trystram. *Algorithmes et architectures parallles.* Inter Editions, Paris, 1993.

[CZ91] E. Chajakis, S.A. Zenios. *Synchronous and asynchronous implementations of relaxation algorithms for non linear network optimization.* Parallel Computing, 17: 873-894, 1991.

[DCG90] J. Dantas De Melo, J.L. Calvet, J.M. Garcia. *Vectorization and multitasking of dynamic programming in control. Experimentation on a CRAY-2.* Parallel Computing, 13 :3 :261-269, 1990.

[Don71] J. D. P. Donnely. *Periodic chaotic relaxation.* Linear Algebra Appl., 4:117–128, 1971.

[Elb84] D. El Baz. *Etude d'algorithmes itratifs de calcul parallle, application la rsolution distribue du problme de routage optimale dans un rseau maill commutation de paquets.* Thse de Docteur Ingnieur, Rapport LAAS 2985, Toulouse, 1984.

[Elb89] D. El Baz. *Mise en œuvre d'algorithmes itratifs distribus asynchrones sur un rseau de transputers.* Calculateurs Parallles, 3:31-40, 1989.

[Elb90] D. El Baz. *M-functions and parallel asynchronous algorithms.* SIAM Journal on Numerical Analysis, vol. 27, no. 1, pages 136–140, February 1990.

[Elb93] D. El Baz. *Asynchronous implementation of relaxation and gradient algorithms for convex network flow problems.* Parallel Computing, 19:1019-1028, 1993.

[Elb94a] D. El Baz. Nonlinear systems of equations and parallel asynchronous iterative algorithms. In G.R. Joubert, D. Trystram, and Peters F. J., editors, *Advances in Parallel Computing 9, Parallel Computing. Trends and Applications*, pages 89–96. Elsevier Science B.V., North Holland, 1994.

[Elb94b] D. El Baz. Parallel iterative algorithms for the solution of markov systems. In *Proceedings of the 33 rd IEEE Conference on Decision and Control*, pages 2524–2527, Orlando, U.S.A., 1994.

[Elb96] D. El Baz. *Asynchronous gradient algorithms for a class of convex separable network flow problems.* Computational Optimization and Applications, vol. 5, pages 187–205, 1996.

[EB96b] D. El Baz. *An efficient termination method for asynchronous iterative algorithms* on message passing architectures. In Proceedings of the International Conference on Parallel and Distributed Computing Systems, Dijon, volume 1, pages 1–7, 1996.

[EGJSM98] D. El Baz, D. Gazen, M. Jarraya, P. Spiteri, and J.C. Miellou. *Flexible communication for parallel asynchronous methods with application to nonlinear optimization problem.* In E. D'Hollander, G. Joubert, F. Peters, and U. Trottenberg, editors, Advances in Parallel Computing: Fundamentals, Applications and New Directions, volume 12, pages 429–436, ParCo 97, Bonn, Germany, 1998. Elsevier Science B.V., North Holland.

[Elb98] D. El Baz, *Contribution à l'algorithmique parallèle, le concept d'asynchronisme, étude théorique, mise en œuvre et application.* Habilitation à Diriger les Recherches de l'Institut National Polytechnique de Toulouse, 1998.

[ESM97] D. El Baz, P. Spiteri, and J.C. Miellou. *Asynchronous multisplitting methods with flexible communication for pseudolinear p.d.e.* In

Proceedings of the Eighth International Colloquium on Differential equations, pages 145–152, Plovdiv, Roumanie, 18-23 August, 1997. VSP International Science Publishers, Utrecht.

[ESMJ99a] D. El Baz, P. Spiteri, J.C. Miellou, M. Jarraya. *Perturbation of fixed point iterative methods : Termination.* Book of Abstrcats of the 8th ILAS Linear Algebra conference, Barcelona, 1999.

[ESMJ99b] P. Spiteri, D. El Baz, J.C. Miellou, M. Jarraya *Perturbation of linear fixed point iterative methods : Roundoff error.* Book of Abstracts of the 8th ILAS Linear Algebra conference, Barcelona, 1999.

[ESMJ00] D. El Baz, P. Spiteri, J.C. Miellou, M. Jarraya. *New results on the distributed termination of asynchronous iterations.* Rapport LAAS N. 00066, 2000.

[ET81] M.N. El Tarazi. Contraction et ordre partiel pour l'étude d'algorithmes synchrones et asynchrones en analyse numérique. *Thèse d'Etat, Faculté des Sciences et Techniques de l'Université de Franche-Comté Besançon*, 1981.

[ET82] M. N. El Tarazi. *Some convergence results for asynchronous algorithms.* Numerisch Mathematik, vol. 39, pages 325–340, 1982.

[ET84] M. N. El Tarazi. *Algorithmes mixtes asynchrones. Etude de convergence monotone.* Numer. Math. 44 :363-369, 1984.

[ESMG96] D. El Baz, P. Spiteri, J.C. Miellou, and D. Gazen. *Asynchronous iterative algorithms with flexible communication for nonlinear network flow problems.* Journal of Parallel and Distributed Computing, 38:1–15, 1996.

[EW78] A.N. El Awtani. *Réalisation, identification et commande d'un processus thermique à entrée et sorties multiples.* Thèse de 3ème cycle, université de Besanon, 1978.

[Fly66] M. J. Flynn. *Very high-speed computing systems.* Proc. IEEE 54:1901-1909, (1966).

[Fro95] A. Frommer. *Parallele asynchrone iterationen.* Henberger, J. Wissenschaftliches Redinm, Akademic-Vulag, Berlin, 1995.

[FS92] A. Frommer and D.B. Szyld. *H-splitting and two-stage iterative methods.* Numerische Mathematik, 63:345–356, 1992.

[FS94] A. Frommer and D.B. Szyld. *Asynchronous two-stage iterative methods.* Numerische Mathematik, 69:141-153, 1994.

[FS97] A. Frommer and D. Szyld. *Asynchronous weighted additive schwarz methods.* Electronic Transactions on Numerical Analysis, 5:48–61, 1997.

[FS98] A. Frommer, D.B. Szyld. *Asynchronous iterations with flexible communicatiom for linear systems.* Calculateurs Parallèles Réseaux et Systèmes Répartis, 10, 4:421-429 , 1998.

[FS99] A. Frommer, D.B. Szyld. *On asynchronous iterations.* Temple University, Department of Mathematics, Report 99-5-31, 1999.

[FS00] A. Frommer, D.B. Szyld. *An Algebraic Convergence Theory for Restricted Additive Schwarz methods Using Weighted Max Norms.* Temple University, Department of Mathematics , Philadelphia, report 19122-6094.

[FSc97] A. Frommer and H. Schwandt. *A unified representation and theory of algebraic additive schwarz and multisplitting methods.* SIAM Journal on Matrix Analysis and Applications, 1997.

[FSS97] A. Frommer, H. Schwandt, D.B. Szyld. *Asynchronous weighted additive Schwarz methods.* Electronic Transaction on Numerical Analysis, 5:48-61, 1997.

[Gaz98] D. Gazen. *Contribution à l'étude d'algorithmes parallèles pour des problèmes d'optimisation de type flot dans les réseaux.* Thèse de l'Université Paul Sabatier de Toulouse, 1998.

[GBEB82] G. Authie, J. Bernussou, D. El Baz. *Distributed asynchronous iterative control algorithms, optimal routing applications.* Proceedings of the IFAC Symposium Component and Instruments for Distributed Control Systems, 49-58, 1982.

[GE99] W. Gropp, E.Ewing lusk. *User's Guide for MPICH, a portable Implementation of MPI, version 1.2.0.* Mathematics and computer Science Division MIT, 1999.

[Gir91] L. Giraud. *Implmentations parallles de mthodes de sous-domaines synchrones et asynchrones pour la rsolution de problmes aux limites.* Thse de Doctorat en Informatique de l'Institut National Polytechnique de Toulouse, 1991.

[GS91] L. Giraud, P. Spiteri. *Résolution parallèle de problèmes aux limites non linéaires.* M.2 A.N., 25:73–100, 1991.

[GS92] L. Giraud, P. Spiteri. *Implementations of parallel solutions for nonlinear boundary value problems.* In D.J. Evans, G.R. Joubert, and H. Liddel, editors, Parallel Computing'91. Advances in Parallel Computing, volume 4, pages 203-211. Elsevier Science B.V., North Holland, Amesterdam, 1992.

[Gui97] R. Guivarch. *Rsolution parallle de problmes aux limites coupls par des mthodes de sous-domaines synchrones et asynchrones.* Diplme de Doctorat en Informatique de l'Institut National Polytechnique de Toulouse, 1997.

[Hav87] M. Haviv. *Aggregation/disaggregation methods for computing the stationary distributions of Markov chains.* SIAM J. numer. Anal, 24,4:952-966,1987.

[Jac76a] C. Jacquemard. *Comparaison entre algorithmes de relaxation.* Colloque d'analyse numérique, PORT BAIL, 1976.

[Jac76b] C. Jacquemard. *Sur les algorithmes de relaxation: convergence monotone, comparaison d'algorithmes, accélération convexe dans le cas linéaire , exemple en contrôl optimal.* Congrés A.F.C.E.T, Chantenay-Malabry, 1976.

[Jac77] C. Jacquemard. *Contribution à l'étude d'algorithme de relaxation à convergence monotone.* Thèse de 3ème Cycle, Université de Besançon, 1977.

[JE99] M. Jarraya, D. El Baz. *Asynchronous iterations for the solution of Markov systems.* 3^{rd} International Workshop on the Numerical Solution of Markov Chains, (NSMC'99), Saragosse (Espagne), 335-338, 1999.

[JE00a] M. Jarraya, D. El Baz. *A distributed iterative algorithm for optimal control problems with block partitions.* Proceedings of International Conference on Artificial and Computational Intelligence for Decision, Control and Automation in Engineering and Industrial Applications ,(ACIDCA2000), Monastir, (Tunisie), 73-78, 2000.

[JE00b] M. Jarraya, D. El Baz. *A new implementation of asynchronous iterations with flexible communication on a network of symmetric multiprocessors.* 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2000), Las Vegas, Nevada, (USA), II:777-783, 2000.

[JEG98] M. Jarraya, D. El Baz, D. Gazen. *Mise en œuvre de méthodes itératives asynchrones avec communication flexible II, Implémentation sur CRAY T3E, SMP et réseau de stations.* Calculateurs Parallèles Réseaux et Systèmes Répartis,10:439–447, 1998.

[JPS81] J. Julliand, G.R. Perrin, P. Spiteri. *Simulation d'exécutions parallèles d'algorithmes numériques asynchrones.* 1ère conférence AMSE, Lyon 1981 et communication au colloque d'Analyse numérique d'Aussois, 1981.

[Kat64] I. Kato. *Demi-continuity, hemi-continuity and monotonicity.* Bull. Amer. Math. Soc., 70:548–550, 1964.

[KH99] W. Knottenbelt, P. Harisson. *Distributed disk-based solution techniques for large Markov Models.* 3^{rd} International Workshop on the Numerical Solution of Markov Chains, (NSMC'99), Saragosse (Espagne), 58-75, 1999.

[Kle75] L. Kleinrock. *Queueing systems, volume1.* John Wiley and Sons,1975.

[KVZ72] M.A. Krasnosel'skii, G.M. Vainikko, P.P. Zabreiko, ta.B. Rutitskii, V.Ya. Stetsnko, *Approximate solution of operator equations.* Wolters-Noordhoff Publishung, Groningen, 1972.

[Lan77] B. Lang. *Calcul hybride décentralisé de la commande optimale d'un processus dynamique continue.* Thèse de Docteur Ingénieur, Université de Besançon, 1977.

[LC93] F. Le Gall, C. Calmettes. *Solving Markov systems on a distributed memory computer.* Workshop on parallel processing in education. Impact Tempus JEP's and Hungarian Transputer users group's, Miskolc (Hongrie), 73-78, 1993.

[LC98] F. Le Gall, C. Calmettes. *Combinaison d'une technique de rduction d'tats la mthode d'Arnoldi pour la rsolution parallle d'un processus GQBD.* Calculateurs Parallles Rseaux et Systmes Rpartis, 10:411-420, 1998.

[Leg86] F. Le Gall, *Génération du système markovien decrivant un réseau à commutation de circuits.* Rapport de recherche, Mai 1986.

[Leg87] F. Le Gall. *Contribution l'tude des rseaux commutation de circuits: Modlisation et rsolution numrique* Thse d'Etat, 1987.

[Leg92] F. Le Gall. *Solving a bloc-tridiagonal Markov system on Parallel Computers.* Advances in Parallel Computing, 4:213-220, 1992

[LM79] F. Lhote, J.C. Miellou. *Algorithmes de dcentralisation et de coordination par relaxation en commande optimale.* Cepadues Ed., Analyse et commande des systmes complexes, Monographie AFCET, Ed. Titli, 1979.

[LLMS80] F. Lhote, B. Lang, J.C. Miellou, P. Spiteri. *Relaxation methods for parallel in line calculations of the optimum control of large systems.* 9th IFIP Conference Varsovie, Optimization Techniques, Part I:324-330, Springer Verlag, 1980.

[LMS86] B. Lang, J.C. Miellou, P. Spiteri, *Asynchronous relaxation algorithms for optimal control problems.* Mathematic and Computers in Simulation, 28:227-242, 1986.

[Lio68] J.L. Lions. *Contrôle optimal des systèmes gouvernés par des équations aux dérivées partielles.* Dunod, 1968.

[Lue73] D. G. Luenberger. *Introduction to linear and nonlinear programming.* Addison-Wesley, 1973.

[LuM86] B. Lubachevsky, D. Mitra. *A chaotic asynchronous algorithm for computing the fixed point of a non negative matrix of unit spectral radius.* J.A.C.M., 33, 1:130-150, 1986.

[Mar79] R.A. Marie. *An approximate analytic method for general queuing networks.* IEEE Transactions on Software Engineering, 5:5:530-538, 1979.

[MES98] J. C. Miellou, D. El Baz, P. Spiteri. *A new class of iterative algorithms with order intervals.* Mathematics of Computation, 67:237–255, 1998.

[Mie72] J. C. Miellou. *Méthode de l'état adjoint par relaxation utilisée pour résoudre un problème de commande optimale.* RAIRO, R-1 81-87,1972.

[Mie73] J.C. Miellou. Problmes de contrle optimal H-dcentralisation : la mthode de relaxation dcentralisation. C.R.A.S., Srie A, t.277, pp.609-612, 1973.

[Mie75a] J. C. Miellou. *Algorithmes de relaxation chaotique retards.* RAIRO, vol. R1, pages 55–82, 1975.

[Mie75b] J. C. Miellou. *Itérations chaotiques à retards, étude de la convergence dans le cas d'espaces partiellement ordonnés.* C.R.A.S. Paris, 280:233–236, 1975.

[Mie86] J. C. Miellou. Asynchronous iterations and order intervals. In *Parallel Algorithms and Architectures.* North Holland, 1986.

[MiS85a] J. C. Miellou, P. Spiteri. *A parallel asynchronous relaxation algorithm for optimal control problems.* Proceedings of the International Conference on Mathematical Analysis and its applications, 289-302, Kuwait, 1985.

[MiS85b] J. C. Miellou and P. Spiteri. *Un critre de convergence pour des mthodes gnrales de point fixe.* Modl. Math. Anal. Numr., vol. 19, pages 645–669, 1985.

[Mit87] D. Mitra. *Asynchronous relaxations for the numerical solution of differential equations by parallel processors.* SIAM J. Sci. Stat. Comput., 8:43-58, 1987.

[Mon95] T. Monteil. *Etude des communications sur rseau, de la couche physique aux appels de procdures distance, Applications possibles pour le logiciel LANDA.* Rapport LAAS Numro 95006, 1995.

[MGGB99] T. Monteil, J.M Garcia, D. Gauchard, O. Brun. *Communication Kernel for high speed networks in the parallel environment LANDA-HSN* 1st IEEE Computer Society International Workshop on Cluster Computing (Iwcc 99), Melbourne, Australie, 63-70, 1999.

[MPS96] V. Migallón, J. Penadés and D.B. Szyld. *Block two stage methods for singular systems and Markov Chains.* Numerical Linear Algebra with Applications, 3:413-426, 1996.

[MPS99] V. Migallón, J. Penadés and D.B. Szyld. *Experimental study of parallel iterative solutions of Markov chains with block partitions* 3^{rd} International Workshop on the Numerical Solution of Markov Chains, (NSMC'99), Saragosse (Espagne),96-110, 1999.

[OLW85] D.P. O'Leary, R.E. WHITE. *Multi-splittings of matrices and parallel solution of linear systems.* SIAM J. Alg. Disc. Meth., 6 :4 :630-640, 1985.

[OR70] J.M. Ortega, W.C. Rheinboldt. *Iterative solution of Nonlinear Equations in Several Variables.* Academic Press, New York, 1970.

[Ort88] J. Ortega. *Introduction to parallel and vector solution of linear systems.* Plenum, New York, 1988.

[PBGM74] L. Pontriaguine, V. Boltianski, Gamkreliozer, Michtchienkoe. *Théorie mathématique des processus optimaux.* Edition de Moscou, 1974.

[PSS93] B. Philipe, Y. Saad, W.J. Stewart. *Numerical methods in Markov chains modeling.* Operations Rechearch, 40,6:1156-1179, 1992.

[RCM75] F. Robert, M. Charney, and F. Musy. *Itérations chaotiques série-parallèle pour des équations non-linéaires de point fixe.* Aplikace Matematiky, 20:1–37, 1975.

[REB98] K. Rhofir, El Kyal, J. Bahi *Rsolution d'un systme diffrentiel algbrique raide par une mthode de multidcomposition asynchrone.* Calculateurs Parallles Rseaux et Systmes Rpartis, 10:459–467, 1998.

[Rhe70] W.C. Rheinboldt. *On M-functions and their application to nonlinear gauss-seidel iterations and to network flows.* J. Math. Anal. and Appl., 32:274–307, 1970.

[Roc70] R. T. Rockafellar. *Convex analysis.* Princeton University Press, Princeton New Jersey, 1970.

[Roc84] R. T. Rockafellar. *Network flows and monotropic optimization.* John Wiley & Sons, New York, 1984.

[Ros69] J. L. Rosenfeld. *A case study in programming for parallel processors.* Communications of the ACM, 12:645–655, 1969.

[Ros84] D. Rose. *Convergent regular splittings for singular M-matrices.* SIAM J. Disc. Meth, 5,1:133-144, 1984.

[Saa95] Y. Saad. *Iterative methods for sparse linear systems* PWS, 1995.

[SBA89] P. Spiteri, S. Benjelloum, and A. Authie. *Parallel algorithms for solving the obstacle problem.* In Computational Mechanics Publications, 5-th International Symposium on Numerical Methods in Engineering, Lausanne, pages 275–281. Springer Verlag, 1989.

[MSGL91] J.C. Miellou, P. Spiteri, L. Giraud, A. Laouar. *Subdomain decomposition methods with overlapping and asynchronous iterations.* In M. Chipot et al., editor, Progress in Partial Differential Equations, pages 166–183. Pitman Research Notes in Mathematics Series, 1991.

[SME95] P. Spiteri, J.C. Miellou, D. El Baz. *Asynchronous alternating Schwarz method for the solution of nonlinear partial differential equations.* Rapport LAAS 95309-IRIT/LCS, page 37, 1995.

[Spi83] P. Spiteri. *Simulation d'excutions parallles pour la rsolution d'inquations variationnelles stationaires.* Revue E.D.F. Informatique et Mathmatique Appliques, Srie C, 1:149-158, 1983.

[Spi84] P. Spiteri. *Contribution à l'étude de grands systèmes non linéaires, comportement d'algorithmes itératifs, stabilité des systèmes continus.* Thèse d'Etat de la Faculté des Sciences et des Techniques de l'Universite de Franche-Comté, 1984.

[Spi86] P. Spiteri. *Parallel asynchronous algorithms for solving boundary value problems.* Parallel Computing, 7:773-784, 1997.

[SJ92] D.B Szyld, M.T. Jones, *Two-stages and multispliting methods for the parallel solution of linear systems.* Siam J. Matrix Anal. Appl, 13, 2:671-679, 1992.

[Ste94] W.I. Stewart. *An introduction to the numerical solution of Markov chains.* Princeton university Press, NJ

[Szy97] D. Szyld. *Different models of parallel asynchronous iterations with overlapping blocks.* Temple University report 97-81, 1997.

[Tak75] Y. Takahashi. *A lumping method for numerical calculations of stationary distributions for Markov chains.* Rapport technique B.18, Tokyo Institute of Technology, Juin 1975.

[Tij88] H.C. Tijims. *Stochastic modelling and analysis.* A computational approach. Wiley, 1988.

[TB86] J. Tsitsiklis and D. Bertsekas. Distributed asynchronous optimal routing in data networks. *IEEE Trans. Auto. Contr.*, 31:325–332, 1986.

[TBT90] P. Tseng, D. Bertsekas, and J. Tsitsiklis. Partially asynchronous parallel algorithms for network flow and other problems. *SIAM J. Control and Optimization*, 28:678–710, 1990.

[TsB87] P. Tseng and D. P. Bertsekas. *Relaxation methods for problems with strictly convex separable costs and linear constraints.* Math. Programming, vol. 38, pages 303–321, 1987.

[ZDE95] B. Zhongzhi, W. Deren, and D. J. Evans. Models of asynchronous parallel matrix multisplitting relaxed iterations. *Parallel Computing*, 25:565–582, 1995.

[ZL88] S. Zenios and R. Lasken. *The Connection machines CM-1 and CM-2, solving nonlinear network problems.* Proceedings of the International Conference on Supercomputing, pages 648–658. ACM Press, New York, N. Y., 1988.

[ZM86] S. Zenios and J. Mulvey. *Nonlinear network programming on vector supercomputers: A study on the Cray x-mp.* Operations Research, 34:667–682, 1986.