Projet de Mémoire d'Habilitation à Diriger des Recherches

Didier EL BAZ

LAAS du CNRS, Toulouse

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction Générale

La Liberté guide nos pas.
Etienne-Nicolas Méhul
Le Chant du Départ

La conception de systèmes technologiques complexes tels que les avions, les véhicules spatiaux ou les grands réseaux, requiert des simulations de très grande dimension qui engendrent d'importants besoins en matière de calcul. Il en va de même dans le domaine des services et la météorologie a à cet égard valeur d'exemple. De nombreux programmes de recherche scientifique nécessitent aussi des volumes de calcul très importants et notamment ceux qui de l'étude de l'infiniment petit à l'infiniment grand font progresser la connaissance sur la matière, l'univers ou le vivant. En réponse à ces besoins de calcul, de nouvelles architectures informatiques faisant appel à des concepts nouveaux tels que la vectorisation ou le parallélisme sont apparues sur le marché dans les années 80. Simultanément, de nombreuses méthodes numériques parallèles ont été proposées. Les progrès de la microélectronique améliorent sans cesse les performances des processeurs, toutefois dans la course aux hautes performances, le concept de parallélisme s'est imposé en matière d'architecture et l'intérêt pour les méthodes numériques parallèles ne cesse de s'accroître. Parmi les méthodes typiquement parallèles, les algorithmes itératifs asynchrones tiennent une place particulièrement importante (cf. [CM69], [Bau78], [Mie75a], [Mie75b], [Ber83], et [BT89]).

Le choix du type de synchronisation entre les tâches de calcul est un des facteurs majeurs d'une parallélisation efficace ; c'est donc un des points fondamentaux du parallélisme au même titre que l'équilibrage des charges ou les problèmes de communication. On notera que ces problèmes peuvent être difficilement dissociés. Il est important de réaliser que l'asynchronisme est une des nouvelles possibilités offertes par l'introduction du concept de parallélisme au mathématicien appliqué ou à l'informaticien. Les algorithmes itératifs asynchrones pour lesquels les composantes du vecteur itéré sont réactualisées en parallèle sans ordre ni synchronisation sont nés du désir d'utiliser au maximum toutes les ressources de calcul des nouvelles architectures informatiques et plus particulièrement des architectures de type MIMD, multiples instructions multiples données, ou SPMD, simple programme multiples données, en supprimant le temps de gestion des synchronisations et le temps d'inactivité des processeurs résultant des synchronisations. Les algorithmes itératifs asynchrones ont été appliqués avec succès à un grand nombre de problèmes : résolution d'équations aux dérivées partielles, équations différentielles ordinaires, systèmes markoviens, et problèmes d'optimisation (cf. [Ros69], [Bau78], [GS91], [GS92], [Spi86], [Mit87], [LM86], [DLM88], [LB87], [DLM88], [Ber82], et [TBT90]).

Dans ce mémoire nous présentons notre contribution à l'algorithmique parallèle et plus particulièrement à l'études des itérations asynchrones tant sur le plan théorique qu'expérimental. Notre objectif étant d'analyser cette classe d'algorithmes parallèles et de montrer son intérêt. Cette étude a commencé lors de notre thèse de Docteur Ingénieur sous la direction des Professeurs Gérard Authié et Georges Grateloup au LAAS du CNRS et a continué pendant notre stage post-doctoral INRIA au MIT au sein de l'équipe "Communication" du "Laboratory for Information and Decision Systems" en collaboration avec le Professeur Dimitri P. Bertsekas. Nos travaux se sont poursuivis ensuite au LAAS du CNRS dans le groupe Décentralisation Hiérarchisation Parallélisme dirigé par J. Bernussou puis dans le groupe Optimisation Filtrage Parallélisme dirigé actuellement par Gérard Salut. Cette étude a été effectuée en partie dans le cadre de deux contrats ATP Systèmes Complexes, le dernier étant en collaboration avec l'équipe du Professeur J. C. Miellou du Laboratoire de Calcul Scientifique de Besançon, ainsi que dans le cadre du contrat Européen Stimulation SCI 0047(H), en collaboration avec l'équipe du Professeur David J. Evans du "Parallel Algorithms Research Centre" de la "Loughborough University of Technology" UK, puis du projet Stratagème sélectionné à l'issu de l'appel d'offre des PRC. Ce travail de recherche s'effectue actuellement dans le cadre d'une collaboration avec les Professeurs Jean-Claude Miellou du Laboratoire de Calcul Scientifique de Besançon et Pierre Spitéri de l'ENSEEIHT-IRIT de Toulouse. Une partie des résultats expérimentaux présentés dans ce mémoire a été obtenue dans le cadre de deux thèses de Doctorat sous notre Direction exclusive (il s'agit des thèses de Madame Cassilda Ribeiro et de Monsieur Didier Gazen, cf. Annexe C), ainsi que des Diplômes d'Etudes Approfondies de Messieurs Olivier Plouviez et Mohamed Jarraya.

Notre contribution à la modélisation des itérations asynchrones, l'analyse de leur convergence, l'étude de leur terminaison, leur mise en œuvre sur diverses architectures informatiques parallèles, leur application à divers problèmes et l'étude de leur performance est présentée dans ce mémoire et mise en regard des principaux travaux en la matière. Cette étude se situe à l'intersection de plusieurs domaines de recherche : Mathématiques Appliquées, Informatique, et Automatique. Nous résumons ci-dessous notre contribution.

Nous avons considéré premièrement une classe importante de problèmes d'optimisation : les problèmes de flot convexes dans les réseaux qui ont de nombreuses applications en distribution, routage d'informations, finance... Nous avons donné tout d'abord un résultat de convergence pour les algorithmes de relaxation asynchrone (cf. [BEB87]). Puis nous avons établi un résultat de convergence pour les algorithmes de gradient asynchrones (cf. [EB96a]). Ces deux résultats ont été obtenus pour le modèle des itérations asynchrones proposé par Bertsekas (cf. [BT89]) ; ils ont reçu une extension dans les deux contributions suivantes qui ont porté sur la résolution de systèmes d'équations non linéaires de type $Fx^* = z$. Nous avons montré en particulier dans [EB90] que lorsque F est continue, hors diagonale monotone décroissante et diagonale monotone strictement croissante, les algorithmes de relaxation asynchrone par point convergent de manière monotone vers une solution à partir d'une $\mathcal{A}$-sur solution ou d'une $\mathcal{A}$-sous-solution. Ces résultats ont été étendus dans [EB94a] en montrant la convergence monotone de certains algorithmes de relaxation asynchrone lorsque F est continue, hors diagonale monotone décroissante et diagonale monotone croissante. De plus nous avons montré dans cette dernière contribution la convergence monotone d'algorithmes de type Richardson lorsque F présente la particularité d'être continue au sens de Lipschitz. Une premiere synthèse des résultats précédents a été présentée dans [EB91] où un lien entre les M-fonctions et les problèmes de flot a été effectué.

Dans [MEBS98] nous avons proposé une nouvelle classe d'algorithmes itératifs asynchrones permettant des communications flexibles entre processeurs et la prise en compte lors des calculs de la valeur courante de toutes les composantes du vecteur itéré. Nous nous sommes placés dans le cadre des M-fonctions et avons considéré la résolution de systèmes d'équations non linéaires issus de la discrétisation d'équations aux dérivées partielles tels que des problèmes de diffusion non linéaire ou d'Hamilton-Jacobi-Bellman. Nous avons établi un résultat de convergence monotone pour des algorithmes asynchrones par bloc avec communication flexible associés à des $\mathcal{A}$-sur applications ou des $\mathcal{A}$-sous-applications. Dans [EBSM97] nous avons donné un résultat de convergence pour des méthodes de multisplitting asynchrones avec communication flexible appliquées à la résolution de problèmes pseudolinéaires. Dans [EBSMG96] nous avons considéré des problèmes de flot convexes dans les réseaux et présenté un résultat de convergence monotone pour des algorithmes asynchrones par point avec communication flexible associés à des sur applications ou à des sous-applications sous des hypothèses identiques à celles de [BEB87] et dans un cadre plus général que celui des M-fonctions.

Diverses méthodes de terminaison pour les algorithmes itératifs asynchrones ont été présentées dans [EB96b] et [EB96c]. La première méthode est un raffinement d'un schéma de terminaison proposé par Bertsekas et Tsitsiklis dans [BT89]. Ce raffinement présente l'avantage de ne pas nécessiter d'acquittement pour tous les messages envoyés. La seconde méthode est basée sur l'utilisation du théorème de convergence asynchrone de Bertsekas et des ensembles de niveau.

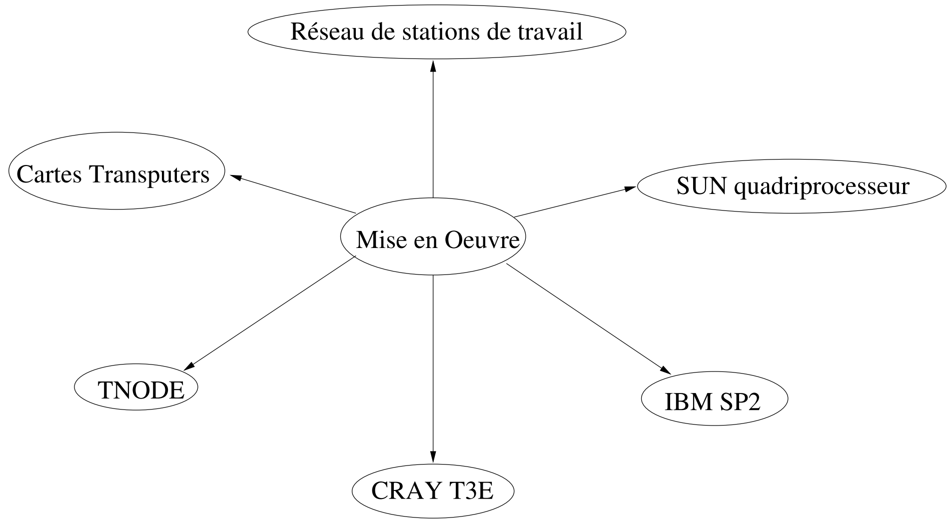


FIG. 1.1 – *Mise en œuvre*

La mise en œuvre des algorithmes itératifs asynchrones sur une machine à mémoire distribuée Tnode de Telmat a été présentée dans [EB89b] et détaillée dans [EB93a]. Une mise en œuvre plus efficace a été proposée dans [GEB95], travail qui a été effectué dans le cadre de la thèse de Didier Gazen (cf. Annexe C et [Gaz98]). Enfin la mise en œuvre des algorithmes itératifs asynchrones avec communication flexible a été présentée dans [EBSMG96] pour une machine à mémoire distribuée Tnode ; dans [EBGMS96] nous avons proposé un autre type de mise en œuvre utilisant des requêtes (ce travail a été aussi effectué dans le cadre de la thèse de Didier Gazen). Une mise en œuvre des itérations asynchrones avec communication flexible sur une machine à mémoire partagée IBM 3090-600 a été présentée dans [SMEB95]. Diverses mises en œuvre des itérations asynchrones sur supercalculateur CRAY T3E et "Symmetric Multiprocessor" ont été proposées dans [EBGJ+98] et [JEBG98] ; dans cette dernière publication nous avons aussi considéré la mise en œuvre sur un réseau de stations de travail, ces études ont été menées dans le cadre du D.E.A. puis de la thèse de notre étudiant Mohamed Jarraya (cf. [Jar97]) ainsi que de la thèse de Didier Gazen.

Les performances des algorithmes itératifs asynchrones sur les diverses machines parallèles citées précédemment ont été étudiées pour des applications très variées : recherche de chemins minimaux par programmation dynamique (cf. [GBEB82] et [EB84]), systèmes Markoviens (cf. [EB94b] et [EB96b]), équations aux dérivées partielles (cf. [EB89b], [MEBS98], et [SMEB95]), problèmes d'optimisation (cf. [EB89a], [EB93a], [EB93b], [EB96a], [EBSMG96], [GEB95], [EBGJ+98], et [JEBG98]).

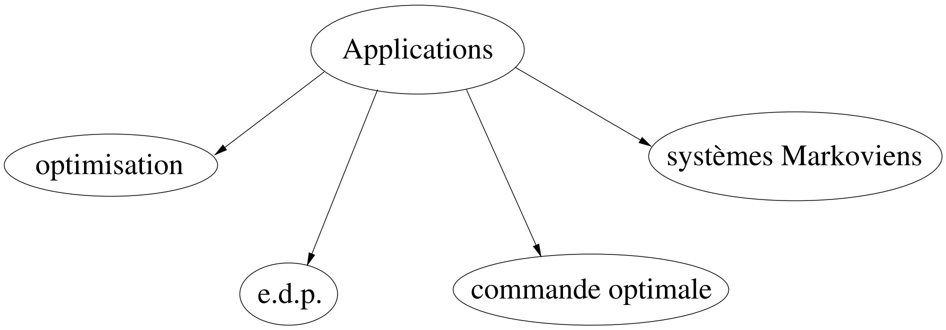


FIG. 1.2 – *Applications*

Pour finir, nous avons proposé dans [REB92b] un algorithme parallèle synchrone original pour la résolution de problèmes de multiflot convexes dans les réseaux, travail qui a été effectué dans le cadre de la thèse de Cassilda Ribeiro (cf. Annexe C et [Rib91]).

Le chapitre 2 qui a pour but de décrire en détail la classe d'algorithmes parallèles faisant l'objet de notre étude, présente aussi une synthèse des principaux résultats de la littérature relatifs à la modélisation des itérations asynchrones. A la fin de ce chapitre nous présentons une de nos principales contributions en la matière : la modélisation des itérations asynchrones avec communication flexible. La convergence des itérations asynchrones est traitée au chapitre 3 où nous exposons des résultats obtenus dans des contextes généraux et effectuons un rappel des principaux résultats théoriques en la matière afin de situer notre contribution. Notons que les résultats de convergence obtenus dans le cadre applicatif des problèmes de flot non linéaires sont présentés au chapitre 6, qui traite des applications ; l'intérêt de ces résultats n'en est pas moindre car leur impact est important, les outils mis en œuvre sont relativement sophistiqués et le travail de démonstration souvent complexe. Le problème de la terminaison des itérations asynchrones est traité au chapitre 4. Après avoir présenté une méthode très générale proposée par Bertsekas pour résoudre ce problème extrèmement complexe mélant les aspects numériques à ceux de l'informatique, nous proposons un raffinement de cette méthode ainsi qu'une approche originale différente. Diverses mises en œuvre d'algorithmes parallèles asynchrones et d'itérations asynchrones avec communication flexible sont présentées au chapitre 5 pour des architectures parallèles très variées : des machines à mémoire distribuée telles que le supercalculateur Cray T3E ou le Tnode de Telmat ; des machines à mémoire partagée de type SMP telles que la Sun Sparcstation 20 quadriprocesseur et un réseau de station de travail. Au chapitre 6 nous présentons diverses applications des itérations asynchrones. Nous portons une attention particulière à une classe de problèmes d'optimisation : les problèmes de flot dans les réseaux. A cette occasion, nous présentons notre contribution portant sur la convergence des méthodes de relaxation et de gradient asynchrones. Nous considérons aussi la résolution d'équations aux dérivées partielles (les problèmes de diffusion non linéaire et d'Hamilton-Jacobi-Bellman, sont étudiés) et de systèmes Markoviens pour lesquel nous présentons un résultat original dans un contexte ordre partiel. Le chapitre 7 est consacré à l'étude des performances des algorithmes itératifs asynchrones ainsi que des itérations asynchrones avec communication flexible pour les diverses applications traitées au chapitre 6 et pour diverses architectures parallèles : Cray T3E, Tnode, IBM SP2, Sun Sparcstation 20 quadriprocesseur, et réseau de stations de travail. Le chapitre 8 présente la conclusion générale ainsi qu'un bilan synthétique sous forme de tableaux de nos publications, encadrement, responsabilités, collaborations et travaux sur contrat. La prospective est présentée au chapitre 9. Divers rappels mathématiques figurent dans l'Annexe A. La liste des publications est présentée dans l'Annexe B. L'Annexe C présente un bref résumé des thèses que nous avons encadrées et qui ont été soutenues.

Notons que ce mémoire est accompagné d'un supplément qui est composé de copies de tirés à part d'articles que nous avons publiés dans des revues scientifiques internationales. Ces copies correspondent aux références suivantes de notre bibliographie : [BEB87], [EB90], [EB93a], [EB94a], [EB96a], [EB96c], [EBSMG96], et [MEBS98].

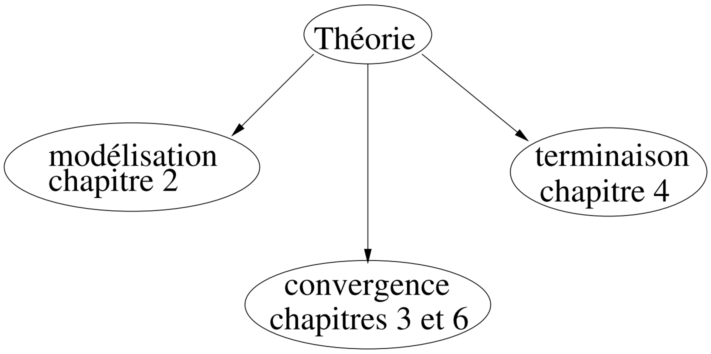


FIG. 1.3 – *Aspects théoriques abordés*

Chapitre 2

Les Algorithmes Itératifs Asynchrones et leurs Modèles

Ce qui n'est pas clair n'est pas français.
A. Rivarol
De l'universalité de la langue française

2.1 Introduction

Les premiers travaux théoriques sur les algorithmes itératifs asynchrones (cf. [CM69]) ainsi que les premières simulations (cf. [Ros69]) datent de 1969. C'est essentiellement la résolution de systèmes linéaires qui était envisagée dans ces deux études. Les premières mises en œuvre sur des systèmes distribués datent de la même époque, l'application considérée étant le routage optimal par recherche de chemins minimaux dans le réseau informatique américain Arpanet. Depuis, de nombreuses études ont été menées afin d'appliquer ces algorithmes à la résolution de problèmes très divers comme nous le verrons notamment au chapitre 6, on peut citer notamment : les équations aux dérivées partielles (cf. [Mie75a]) et [Bau78]), les équations différentielles ordinaires (cf. [Mit87]), la programmation dynamique (cf.[Ber82]) et les problèmes d'optimisation de type flot dans les réseaux (cf. [BEB87]). Nous commençons ce chapitre par des considérations intuitives sur les itérations asynchrones et leur motivation que nous complétons par une étude des différents modèles proposés dans la littérature afin de bien en présenter la portée. Nous terminons ce chapitre par notre contribution à la définition d'un nouveau modèle : les itérations asynchrones avec communication flexible.

2.2 Principe et motivation

Dans un algorithme itératif asynchrone les composantes du vecteur itéré sont réactualisées sans ordre a priori ni synchronisation. Les restrictions imposées aux algorithmes itératifs asynchrones sont très faibles : aucune composante du vecteur itéré ne doit cesser d'être réactualisée de manière définitive et les valeurs des composantes associées à des itérés trop anciens doivent cesser d'être utilisées au fur et à mesure que les calculs progressent.

A titre d'illustration nous comparons ci-dessous un fonctionnement synchrone et un fonctionnement asynchrone sur un exemple simple. Considérons le problème de point fixe suivant :

$$x^* = F(x^*)$$

où F est une application de R^2 dans R^2 et $x^* \in R^2$. Un schéma itératif de type Jacobi se prette bien à une mise en œuvre parallèle synchrone. L'algorithme itératif de type Jacobi est défini de manière recursive par :

$$x_i(j+1) = F_i(x_1(j), x_2(j)), \forall i \in \{1,2\}, j = 0, 1, 2, ...$$

où F_i est la i-ème composante de l'application F et x_i la i-ème composante du vecteur x. Les deux composantes du vecteur itéré peuvent être calculées en parallèle. Deux processeurs notés respectivement P_1 et P_2 collaborent à la recherche du point fixe de l'application F. Les processeurs P_1 et P_2 réactualisent respectivement la première et la deuxième composante du vecteur itéré. Afin de mettre en œuvre un schéma itératif Jacobi parallèle, il est nécessaire de réactualiser les composantes du vecteur itéré suivant un certain ordre et en effectuant certaines synchronisations. Un exemple type de déroulement des calculs est donné par la figure 2.1 où les rectangles blancs numérotés représentent les phases de réactualisation et les rectangles hachurés les phases de communication et d'attente, les flèches délimitant le commencement et la fin des communications. On constate que la nécessité de respecter un ordre de calcul strict et de synchroniser les processeurs peut engendrer des pertes de temps importantes. Les contraintes d'ordre de réactualisation et de synchronisation peuvent aller à l'encontre de la recherche de bonnes performances.

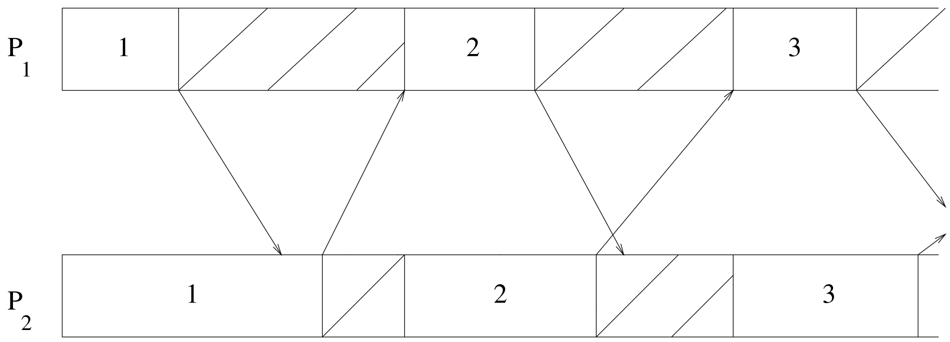


FIG. 2.1 – *Itération synchrone*

Pour illustrer les algorithmes itératifs asynchrones on peut reprendre l'exemple simple précédant. Un type de déroulement asynchrone des calculs est alors donné par la figure 2.2 où les rectangles blancs numérotés représentent les phases de réactualisation et les rectangles hachurés les phases de communication. Le numéro d'itération est incrémenté au commencement de chaque nouvelle phase de réactualisation. On note que les rectangles hachurés ne contiennent pas de période d'inactivité et que les phases de réactualisation s'enchaînent plus rapidement.

Les algorithmes itératifs asynchrones sont particulièrement adaptés aux architectures infomatiques parallèles représentées par le modèle de Dijkstra (cf. [Bou88]), qui est un modèle

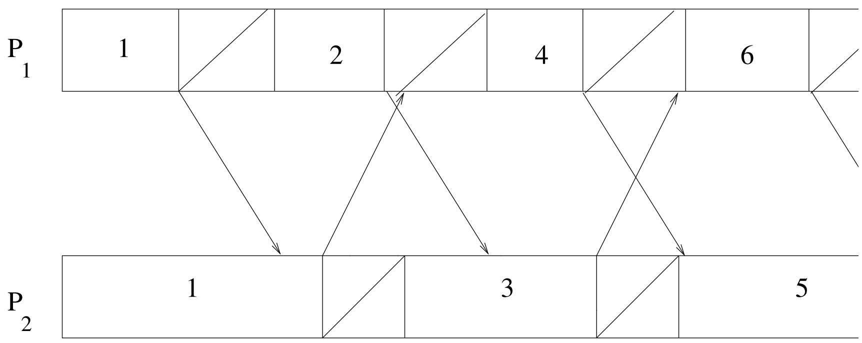


FIG. 2.2 – *Itération asynchrone*

essentiellement asynchrone de type processus interagissant avec une mémoire commune; d'où une grande simplicité de mise en œuvre.

Un autre avantage des algorithmes itératifs asynchrones réside dans l'absence de temps d'inactivité dûs aux synchronisations ainsi que dans l'absence de temps de gestion des synchronisations entre processus itératifs parallèles ou distribués. En effet la synchronisation peut détériorer les performances des algorithmes parallèles. La détérioration est essentiellement fonction du type de synchronisation retenu et de sa mise en œuvre sur la machine ainsi que de la granularité et de l'équilibrage des tâches (cf. [Kun76], [BPF89], [KW84], [Gre89], et [DLM88]).

De plus dans le cas où certaines valeurs des composantes du vecteur itéré changent très peu, il peut être intéressant de ne pas attendre systématiquement ces valeurs. De manière générale on peut espérer une meilleure utilisation des ressources surtout lorsque le nombre de processeurs est élevé.

On notera aussi qu'une mise en œuvre asynchrone permet un meilleur recouvrement des communications par les calculs.

Les algorithmes itératifs asynchrones conviennent aussi particulièrement à la nature de certains problèmes temps réel dans les grands systèmes, pour lesquels la synchronisation de nombreuses tâches de calcul distantes ne peut être envisagée de manière réaliste, en raison notamment de pannes fréquentes dans le système, comme par exemple pour le problème du routage ou pour le contrôle de flot dans les réseaux de données.

Du fait de la suppression des phases de resynchronisation et de réinitialisation (cf. [Ber82]), l'asynchronisme présente aussi l'avantage d'une meilleure adaptativité aux modifications intervenant dans le système telles que les changements de données ou de topologie.

Enfin un dernier avantage de l'asynchronisme est d'augmenter la sureté de fonctionnement. En effet les schémas de calcul asynchrones tolèrent les cas de pannes temporaires où certains itérés parviennent en un temps infini à leur destinataire. De plus dans le cas de l'allocation dynamique des tâches de calcul sur des architectures de type multiprocesseur à mémoire partagée, l'algorithme itératif peut continuer tant qu'un processeur fonctionne.

2.2.1 Problèmes soulevés

L'utilisation d'algorithmes itératifs asynchrones demande néanmoins de faire face à certains défis. L'étude de la convergence des algorithmes itératifs asynchrones parallèles ou distribués est généralement plus complexe que celle des algorithmes synchrones notamment dans le cas non linéaire. Cependant ce problème a été abondamment traité dans la littérature et l'on possède aujourd'hui de nombreux résultats comme nous le verrons au chapitre 3.

L'asynchronisme soulève aussi des difficultés au niveau de la terminaison des algorithmes. Ce problème est très complexe notamment dans le cas où le système n'a pas d'horloge globale et où les processeurs disposent uniquement d'informations locales. Néanmoins des schémas de terminaison ont été proposés. Certains ne sont pas tout à fait satisfaisants au niveau théorique mais présentent cependant un grand intérêt pratique, d'autres présentent toutes les garanties théoriques de bon fonctionnement comme nous le verrons au chapitre 4.

2.2.2 Terminologie

Pour finir ce paragraphe d'introduction sur les algorithmes itératifs asynchrones, nous tenons à faire remarquer qu'il existe plusieurs dénominations pour les algorithme étudiés ici. Ce phénomène est d'ailleurs assez fréquent dans le domaine des Mathématiques Appliquées. Ainsi les algorithmes itératifs asynchrones ont été aussi appelés : algorithmes de relaxation chaotique (cf. [CM69]), algorithmes de relaxation chaotique à retards (cf. [Mie75a]), itérations asynchrones (cf. [Bau78]), algorithmes itératifs totalement asynchrones (cf. [BT89]). Jean Claude Miellou suggère aussi : itérations générales de point fixe sur un espace produit.

Nous verrons au paragraphe suivant que dans la classe des itérations asynchrones on peut distinguer différentes sous-classes de méthodes correspondant à des degrés divers d'asynchronisme.

2.3 Les modèles

2.3.1 Un premier modèle d'itérations asynchrones

Nous considérons le problème de point fixe :

$$x^* = F(x^*), \tag{2.1}$$

où F est une application de R^n dans R^n. Dans ce sous-paragraphe nous présentons un des tous premiers modèles d'algorithmes itératifs asynchrones; la formulation utilisée ici est dérivée des modèles de Chazan et Miranker (cf. [CM69]), Miellou (cf. [Mie75a]), et Baudet (cf. [Bau78]).

Définition 2.1 : *Soit N l'ensemble des entiers naturels, $n, \alpha \in N$, $\alpha \leq n$ et la décomposition de R^n en $\prod_{i=1}^{\alpha} R^{n_i}$, $\sum_{i=1}^{\alpha} n_i = n$. Une itération asynchrone associée à l'application F de $\prod_{i=1}^{\alpha} R^{n_i}$ dans $\prod_{i=1}^{\alpha} R^{n_i}$ et au point initial $x(0) \in \prod_{i=1}^{\alpha} R^{n_i}$ est une suite $\{x(j)\}, j = 0, 1, ...,$ de vecteurs de $\prod_{i=1}^{\alpha} R^{n_i}$ définie de manière recursive pour $i = 1, ..., \alpha$:*

$$\begin{cases} x_i(j) = F_i(x_1(I_1(j)), ..., x_\alpha(I_\alpha(j))) \text{ si } i \in S(j), \\ x_i(j) = x_i(j-1) \text{ si } i \notin S(j), \end{cases} \tag{2.2}$$

où $x_i \in R^{n_i}$ représente le i-ème sous-vecteur du vecteur x et F_i la i-ème bloc composante de l'application F, $\mathcal{S} = \{S(j) | j = 1, 2, ...\}$ est une suite de sous-ensembles non vides de $\{1, ..., \alpha\}$ et $\mathcal{I} = \{I(j) = (I_1(j), ..., I_\alpha(j)) | j = 1, 2, ...\}$ est une suite d'éléments de N^α. De plus $\mathcal{S}$ et $\mathcal{I}$ satisfont les conditions suivantes pour $i = 1, ..., \alpha$:

(a) $0 \leq I_i(j) \leq j - 1, j = 1, 2, ...$

(b) $I_i(j)$ tend vers l'infini lorsque j tend vers l'infini.

(c) i apparaît un nombre infini de fois dans $\mathcal{S}$.

Les conditions ci-dessus peuvent être interprétées respectivement comme suit :

(a) les valeurs du vecteur itéré utilisées lors des calculs à l'itération j proviennent au maximum de l'itération $j - 1$;

(b) les valeurs trop anciennes des composantes du vecteur itéré doivent être définitivement écartées au fur et à mesure que les calculs progressent ;

(c) aucun sous-vecteur du vecteur itéré ne cesse d'être réactualisée.

Une itération asynchrone associée à l'application F, au point initial $x(0)$ et aux suites $\mathcal{S}$ et $\mathcal{I}$ est notée $(F, x(0), \mathcal{S}, \mathcal{I})$.

Remarque 2.1 : *Le modèle (2.2) décrit des méthodes itératives par bloc. Dans le cas où $\alpha = n$ on retrouve le cas particulier des méthodes par point.*

Remarque 2.2 : *Dans les premières définitions des itérations asynchrones (cf. [CM69] et [Mie75a]), on trouvait à la place de (b) la condition suivante :*

(b') il existe un entier s tel que $j - I_i(j) \leq s$ pour $i = 1, ..., \alpha$ et $j = 1, 2, ...$

On note que (b') implique (b). On verra au sous-paragraphe 2.3.4 que cette hypothèse correspond à un fonctionnement partiellement asynchrone.

Remarque 2.3 : *Dans certains cas on rajoute la condition suivante :*

(d) la fonction $I_i(j)$ est monotone croissante pour tout $i \in \{1, ..., \alpha\}$.

Nous verrons au chapitre 3 que l'introduction de cette hypothèse permet d'établir des résultats de convergence monotone dans le cas d'opérateurs de point fixe monotones croissants.

2.3.2 Interprétation du modèle

Un algorithme itératif asynchrone $(F, x(0), \mathcal{S}, \mathcal{I})$ peut être interprété de la manière suivante. Soit $\{P_1, ..., P_\alpha\}$ un ensemble de α processeurs d'une machine parallèle disponibles pour une application. Soit $\{t(j)\}, j = 1, 2, ...$ une suite croissante d'instants. A l'instant $t(j)$ les processeur P_i $i \in S(j)$ qui sont inactifs sont assignés à l'évaluation de $x(j)$ qui diffère de $x(j-1)$ uniquement par les valeurs des sous-vecteurs x_i (cf. figure 2.2). Un processeur P_i commence la réactualisation du sous-vecteur x_i en utilisant les valeurs $x_l(I_l(j)), l = 1, ..., \alpha$, des sous-vecteurs x_l qui sont disponibles au commencement des calculs et qui proviennent d'itérés antérieurs; la stratégie la plus naturelle étant de prendre la valeur la plus récente des composantes. A un instant ultérieur noté $t(j+k)$, avec $k \in N$ et $k > 0$, le processeur P_i terminera ces calculs et sera assigné à l'évaluation de $x_i(j+k)$.

On peut donner d'autres interprétations de l'algorithme itératif asynchrone $(F, x(0), \mathcal{S}, \mathcal{I})$. Par exemple, on peut considérer que les instants $t(j), j = 1, 2, ...$ ne correspondent plus aux instants où les processeurs inactifs sont assignés à l'évaluation des $x(j)$ mais aux instants où les processeurs qui ont terminé leur réactualisation délivrent leurs résultats à une mémoire commune dans le cas d'une architecture de type mémoire partagée, ou commencent à transmettre leurs résultats aux autres processeurs dans le cas d'une architecture de type mémoire distribuée ou plus généralement de type envoi de message (cf. figure 2.3).

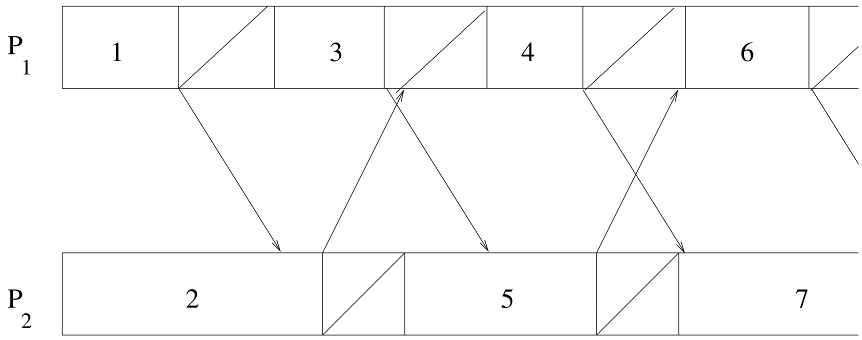


FIG. 2.3 – *Une autre interprétation des itérations asynchrones*

Les retards

Les termes $j - I_l(j)$ sont appelés retards; $j - I_l(j)$ est en fait le retard dans l'accès au l-ème sous-vecteur du vecteur itéré pour les calculs effectués à l'itération j. Dans la formulation des algorithmes itératifs asynchrones (cf. équation (2.2)), les retards permettent de représenter l'effet de deux types de phénomènes liés à l'absence de synchronisation : la différence de durée des phases de calcul et les délais de communication.

Différence de durée des phases de calcul

La différence de durée des phases de calcul provient de la différence de puissance des processeurs ou de la différence de charge qui est due à un mauvais équilibrage pouvant résulter par exemple du non-déterminisme de la durée des tâches de calcul. 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 reprendre l'exemple simple du paragraphe 2.2 où deux processeurs collaborent à la recherche du point fixe d'un opérateur F de R^2 dans R^2. Les processeurs P_1 et P_2 réactualisent respectivement la première et la deuxième composante du vecteur itéré notées respectivement x_1 et x_2. On considère que chaque processeur utilise pour ses réactualisations les valeurs des composantes disponibles en début de calcul et on suppose par simplicité que les communications sont instantanées, un exemple de déroulement des calculs est donné par la figure 2.4 où les traits horizontaux indiquent l'instant où un processeur commence la réactualisation d'une composante. On remarque que $x_1(j) = F_1(x_1(j-1), x_2(j-6))$.

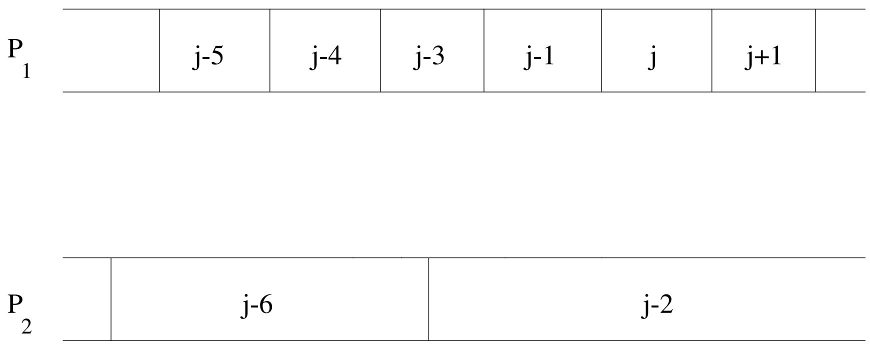


FIG. 2.4 – *Effet des différences de durée des phases de calcul*

Délais de communication

Les délais de communication résultent essentiellement de l'architecture étudiée. Dans le cas de systèmes distribués 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. Pour illustrer ce phénomène on reprend l'exemple précédent en abandonnant l'hypothèse de communication instantanée. Un exemple de déroulement des calculs est donné par la figure 2.5 où les extrémités des flèches représentent respectivement le début et la fin des communications. On remarque que $x_1(j) = F_1(x_1(j-2), x_2(j-3))$. Le processeur P_1 recevant la valeur $x_2(j-1)$ aprés l'instant $t(j)$ correspondant au commencement de la j-ème itération. Les retards traduisent ici le fait que les délais de communication sont non nuls et indéterminés.

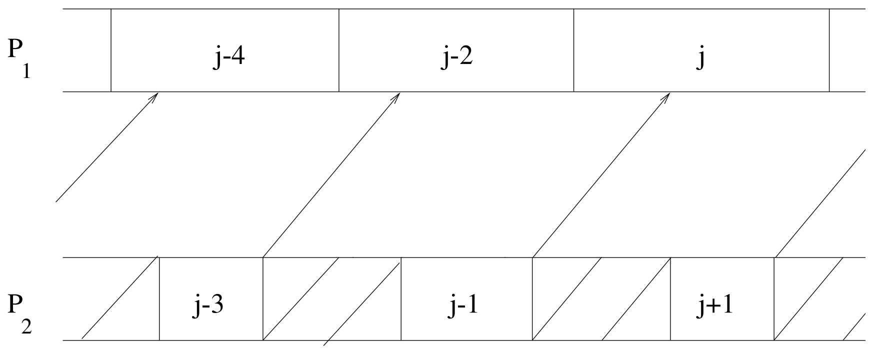


FIG. 2.5 – *Effet des délais de communication*

Exemples de retards non bornés

Il résulte de la condition (b) du sous-paragraphe 2.3.1 que la formulation des itérations asynchrones autorise des retards non bornés. Une illustration simple de cette propriété est donnée en reprenant l'exemple précédent et en supposant à nouveau que les communications sont instantanées. On suppose de plus que le processeur P_1 réactualise la composante x_1 en une unité de temps et que P_2 réactualise la composante x_2 en k unités de temps lors de la k-ième réactualisation de x_2 (cf. figure 2.6). Il est clair que la condition (b) du sous-paragraphe 2.3.1 est alors satisfaite et que le retard $j - I_2(j)$ tend vers l'infini lorsque j tend vers l'infini. Baudet indique dans [Bau78] que $j - I_2(j)$ croît comme $\sqrt{j}$ (cf. figure 2.7).

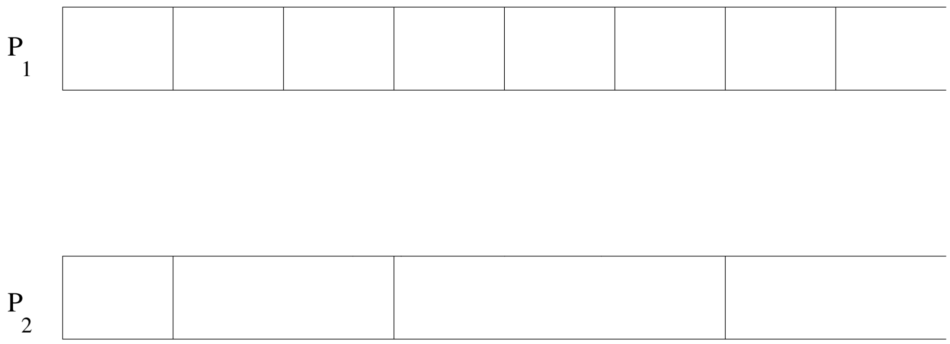


FIG. 2.6 – *Exemple de retards non bornés*

Remarque 2.4 : *La formulation (2.2) des itérations asynchrones permet de prendre en compte le cas où les valeurs des composantes du vecteur itéré sont utilisées lors des réactualisations dans un ordre différent de leur obtention. Ceci correspond notamment au cas où les informations transitent dans un système distribué par divers chemins avec des délais de communication variables.*

Remarque 2.5 : *La formulation (2.2) des itérations asynchrones permet aussi de décrire des situations où certains résultats de réactualisation ne sont jamais utilisés dans les calculs. Ce phénomène peut avoir plusieurs causes : certains résultats peuvent parvenir à d'autres processeurs mais être rapidement supplantés par des résultats plus récents, certains résultats peuvent aussi ne pas être communiqués suite à une panne temporaire.*

Généralité de la formulation

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 (2.2) des algorithmes itératifs asynchrones englobe tous les algorithmes classiques. L'algorithme de Jacobi peut être représenté par exemple de la manière suivante avec le modèle (2.2) :

$$S(j) = \{1, ..., \alpha\},\ j = 1, 2, ...$$

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

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.

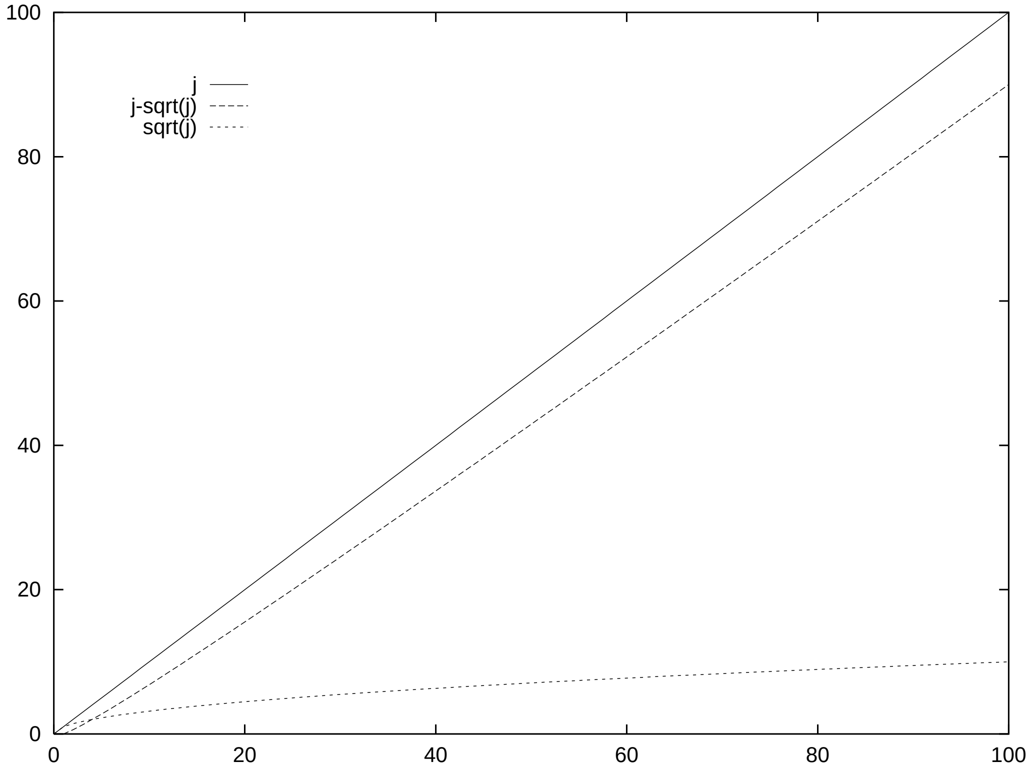


FIG. 2.7 – *Numéros d'itération et retards dans le cas de la figure 2.6*

2.3.3 Les itérations asynchrones avec mémoire

Nous présentons maintenant une manière différente de concevoir des méthodes itératives asynchrones.

Les itérations asynchrones avec mémoire ont été proposées pour traiter le cas où l'application de point fixe n'est pas définie de manière explicite et où certains processeurs sont assignés à des calculs de fonctions intermédiaires alors que d'autres réactualisent les composantes du vecteur itéré. Ces méthodes ont été étudiées notamment par Miellou (cf. [Mie75b]), El Tarazi (cf. [ET81]) et Baudet (cf. [Bau78]).

Les itérations asynchrones avec mémoire utilisent à chaque itération plusieurs valeurs de chaque composante du vecteur itéré qui peuvent être relatives à des numéros d'itérations différents. Le nombre de mémoires est égal au nombre de valeurs utilisées moins une.

A titre d'illustration nous considérons l'exemple simple de la mise en œuvre de la méthode de Newton sur deux processeurs notés respectivement P_1 et P_2. Soit $\mathcal{A}$ une application non linéaire de R^n dans R^n et le problème :

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

L'opérateur de Newton associé à l'application $\mathcal{A}$ s'écrit :

$$F(x) = x - [\mathcal{A}'(x)]^{-1}\mathcal{A}(x).$$

Une décomposition naturelle des calculs consiste à assigner le calcul de $\mathcal{A}'$ au processeur P_1 et le calcul de F au processeur P_2. Les processeurs P_1 et P_2 qui ne sont pas synchronisés vont chacun à leur propre rythme. Le calcul de F fait appel à deux valeurs de x qui peuvent être relatives à des numéros d'itération différents. Nous avons là un exemple simple d'algorithme asynchrone avec une mémoire.

Le problème (2.1) peut donc être généralisé de la manière suivante. Soit F une application de domaine inclus dans un espace vectoriel $[R^n]^m$ et à valeur dans l'espace vectoriel R^n, on recherche le point fixe de F qui vérifie :

$$x^* = F(x^*, x^*, ..., x^*). \tag{2.3}$$

Nous donnons maintenant la définition des itérations asynchrones avec mémoire.

Définition 2.2 : *Soit $\alpha \in N$, $\alpha \leq n$ et la décomposition de R^n en $\prod_{i=1}^{\alpha} R^{n_i}$, $\sum_{i=1}^{\alpha} n_i = n$. Soit F une application de $[\prod_{i=1}^{\alpha} R^{n_i}]^m$ dans $\prod_{i=1}^{\alpha} R^{n_i}$. Une itération asynchrone avec $m-1$ mémoires associée à l'application F et au sous-ensemble X des m premiers vecteurs $\{x(0), x(1), ..., x(m-1)\}$ est une suite $\{x(j)\}$, $j = 0, 1, ...$ de vecteurs de $\prod_{i=1}^{\alpha} R^{n_i}$ telle que pour $i = 1, ..., \alpha$ et $j = m, m+1, ...$*

$$\begin{cases} x_i(j) = F_i(z^1, ..., z^m) \ si \ i \in S(j), \\ x_i(j) = x_i(j-1) \ si \ i \notin S(j), \end{cases} \tag{2.4}$$

où z^r, $1 \leq r \leq m$ est le vecteur formé des sous-vecteurs $z_l^r = x_l(I_l^r(j))$, $1 \leq l \leq \alpha$, la suite $\mathcal{S} = \{S(j) | j = m, m+1, ...\}$ représente une suite de sous-ensemble non vide de $\{1, ..., \alpha\}$ et

$\mathcal{I}$ défini par : $\mathcal{I} = \{(I_1^1(j), ..., I_\alpha^1(j), I_1^2(j), ..., I_\alpha^m(j)) | j = m, m+1, ...\}$ *est une suite d'éléments de* $[N^\alpha]^m$*. De plus* $\mathcal{S}$ *et* $\mathcal{I}$ *satisfont les conditions suivantes pour* $i = 1, ..., \alpha$ *:*

(a) $\max_{r \in \{1,...,m\}}\{I_i^r(j) \mid 1 \leq r \leq m\} \leq j-1$*, pour tout* $j = m, m+1, ...$

(b) $\min_{r \in \{1,...,m\}}\{I_i^r(j) \mid 1 \leq r \leq m\}$ *tend vers l'infini lorsque* j *tend vers l'infini.*

(c) i *apparaît un nombre infini de fois dans* $\mathcal{S}$*.*

Une itération asynchrone avec $m-1$ mémoires associée à l'application F, au sous-ensemble X des m premiers vecteurs $\{x(0), x(1), ..., x(m-1)\}$ et définie par $\mathcal{S}$ et $\mathcal{I}$ est notée $(F, X, \mathcal{S}, \mathcal{I})$.

2.3.4 Le modèle de Bertsekas

Dans ce sous-paragraphe nous présentons une formulation d'algorithmes itératifs asynchrones qui diffère du modèle (2.2) et qui a un grand intérêt en liaison avec le Théorème de convergence asynchrone de Bertsekas que nous étudierons au sous-paragraphe 3.2.3. Notons que ce théorème présente un large champs d'application. L'approche développée par Bertsekas et son équipe dans [Ber83], [BT89], [BT91], et [TBT90] qui est présentée ici va nous permettre d'éclairer davantage la nature des itérations asynchrones par une formulation différente de celle qui a été utilisée jusqu'ici et de rentrer dans de plus amples détails, en étudiant différentes sous-classes d'algorithmes itératifs asynchrones qui présentent un grand intérêt pour des applications variées.

Soit $X_1, X_2, ..., X_n$, des ensembles et X leur produit cartesien :

$$X = X_1 \times X_2 \times ... \times X_n.$$

Les éléments de x de X sont aussi notés:

$$x = (x_1, x_2, ..., x_n),$$

où x_i est élément de $X_i, i = 1, ..., n$. Nous considérons des fonctions $F_i : X \to X_i, i = 1, ..., n$ et la fonction $F : X \to X$ telle que :

$$F(x) = (F_1(x), F_2(x), ..., F_n(x))^t, \forall x \in X.$$

Le problème consiste à trouver le point fixe de F, c'est à dire un élément $x^* \in X$ tel que

$$x^* = F(x^*),$$

où de manière équivalente

$$x_i^* = F_i(x^*), \ \forall i = 1, ..., n.$$

Le modèle des algorithmes totalement asynchrones de Bertsekas est le suivant.

Définition 2.3 : *On suppose qu'il existe une suite d'événements* $T = \{0, 1, 2.,,,\}$ *pour lesquels une ou plusieurs composantes* x_i *du vecteur itéré* x *sont réactualisées par un des processeurs de l'architecture parallèle ou distribuée. Soit* T^i *la sous-suite d'événements pour lesquels* x_i *est réactualisée. Nous supposons que le processeur réactualisant* x_i *peut ne pas avoir accès aux valeurs les plus récentes des composantes de* x*, de plus pour tout* $t \notin T^i$*,* x_i

reste inchangé. Une itération asynchrone est une suite $\{x(t)\}$ de vecteurs de R^n telle que pour tout $i \in \{1, ..., n\}$:

$$\begin{cases} x_i(t+1) = F_i(x_1(\tau_1^i(t), ..., x_n(\tau_n^i(t)), \forall t \in T^i, \\ x_i(t+1) = x_i(t), \forall t \notin T^i. \end{cases} \tag{2.5}$$

où $\tau_l^i(t)$ satisfait :

$$0 \leq \tau_l^i(t) \leq t, \forall l \in \{1, ..., n\}, \forall t \in T. \tag{2.6}$$

On donne maintenant les hypothèses portant sur les calculs et les communications.

Hypothèse 2.1 (Hypothèse d'asynchronisme total) : *Les ensembles T^i sont infinis, de plus si $\{t_k\}$ est une sous-suite d'éléments de T^i qui tend vers l'infini, alors $\lim_{k\to\infty} \tau_l^i(t_k) = \infty$ pour tout $l \in \{1, ..., n\}$.*

On remarque que l'inégalité (2.6) et l'Hypothèse 2.1 permettent de retrouver les hypothèses (a) à (c) du paragraphe 2.3.1. Les algorithmes satisfaisant l'Hypothèse 2.1 sont appelés algorithmes totalement asynchrones par Bertsekas.

Remarque 2.6 : *Dans le modèle (2.5), les éléments de T sont les indices de la séquence d'instants auxquels les réactualisations sont délivrées. La connaissance des ensembles T et T^i n'est pas nécessaire a priori pas plus qu'il n'est nécessaire d'implémenter en chaque processeur une horloge globale associée à la suite d'évènements T. En fait, la variable t ne correspond pas nécessairement à un temps, elle peut être interprétée comme une variable artificielle utilisée pour indexer des instants correspondants à la réactualisation d'une composante.*

Remarque 2.7 *La différence $t - \tau_l^i(t)$ représente le retard dans l'accès à la l-ème composante du vecteur itéré lors de la réactualisation de la i-ème composante à l'instant t. Ce modèle est un peu plus général que (2.2) car la réactualisation simultanée de deux composantes du vecteur itéré peut faire appel à des valeurs différentes de la même composante. On note toutefois que contrairement au modèle (2.2) présenté au paragraphe 2.3.1, l'interprétation du modèle (2.5) est uniquement associée aux instants de fin de réactualisation.*

A partir de ce modèle, Bertsekas et son équipe ont distingué différentes sous-classes d'algorithmes itératifs asynchrones présentant un grand intérêt essentiellement lorsqu'on ne peut établir de résultat de convergence pour les algorithmes itératifs totalement asynchrones.

Les itérations partiellement asynchrones introduites par Bertsekas et Tsitsiklis dans [BT89] (voir aussi [TBT90]) ont un caractère moins général que les itérations totalement asynchrones. Cependant les algorithmes itératifs patiellement asynchrones peuvent présenter un grand intérêt lorsque l'excès d'asynchronisme entraîne la divergence ou ne permet pas de garantir la convergence. Le modèle des itérations partiellement asynchrones est assez semblable à celui des itérations totalement asynchrones. Cependant des bornes sont placées sur les retards et la durée de l'intervalle entre deux réactualisations consécutives d'une même composante. En conservant les mêmes notations, nous présentons maintenant les conditions d'asynchronisme partiel.

Hypothèse 2.2 (Hypothèse d'asynchronisme partiel) : *Il existe un entier positif B tel que :*

(a) Pour tout $i \in \{1, ..., n\}$ et pour tout t, au moins un élément de l'ensemble $\{t, t+1, ..., t+B-1\}$ appartient à T^i.

(b) $t - B < \tau_l^i(t) \leq t, \forall i, l \in \{1, ..., n\}$ et $t \in T^i$,

(c) $\tau_i^i(t) = t, \forall i \in \{1, ..., n\}$ et $t \in T^i$.

Il résulte de cette hypothèse que :

(a) chaque composante est réactualisée au moins une fois au cours d'un intervalle de longueur B, c'est à dire contenant B réactualisations ;

(b) les informations utilisées pour la réactualisation d'une composante ont un retard maximal égal à B ; l'hypothèse (b) qui fixe une borne sur le retard est à rapprocher de l'hypothèse (b') du sous-paragraphe 2.3.1 ;

(c) lors de la réactualisation de la composante qui lui est assignée, chaque processeur utilise la dernière valeur de cette même composante; cette hypothèse est très naturelle dans le cas de la mise en œuvre sur une machine à mémoire distribuée ou sur un réseau de processeurs.

On note que les conditions (a) à (c) sont plus restrictives que les hypothèses d'asynchronisme total mais qu'elles peuvent être satisfaites dans de nombreuses situations pratiques. Le caractère restrictif des hypothèses (a) à (c) dépend de la valeur de B qui est aussi appelée mesure d'asynchronisme. Bertsekas distingue deux classes principales d'algorithmes itératifs partiellement asynchrones : celle pour laquelle B doit être arbitrairement petit et celle pour laquelle B peut être aussi grand que l'on veut mais fini. Suivant le problème, on sera amené à restreindre la valeur de B afin d'établir un résultat de convergence pour les algorithmes itératifs asynchrones.

2.3.5 Un nouveau concept : les itérations asynchrones avec communication flexible

Nous présentons maintenant une nouvelle classe de méthodes itératives asynchrones : les itérations asynchrones avec communication flexible. Cette classe d'algorithmes asynchrones a été tout d'abord proposée dans [MEBS98] pour des problèmes faisant intervenir des M-fonctions (cf. paragraphe A.1.2 de l'Annexe A) et des applications de point fixe qui sont des $\mathcal{A}$-sous-applications ou des $\mathcal{A}$-sur applications (cf. Définitions A.17 et A.18 de l'Annexe A). Les itérations asynchrones avec communication flexible peuvent être aussi associées à des applications de point fixe qui sont des sous-applications ou des sur applications (cf. [EBSMG96] et Définitions A.19 et A.20 de l'Annexe A).

Les itérations asynchrones avec communication flexible sont des méthodes très générales grâce auxquelles chaque processeur peut avoir accès à l'état courant des autres processeurs. Ainsi cette nouvelle classe d'algorithmes autorise des échanges de valeurs qui n'étaient pas prévus par les modèles itératifs asynchrones étudiées dans [CM69], [Mie75a], et [Bau78]. En effet, la valeur courante de chaque composante du vecteur itéré peut être communiquée aux autres processeurs ou acquise par les autres processeurs à n'importe quel moment et éventuellement sans aucune règle prédéterminée, alors que les communications se produisent uniquement à la fin de chaque phase de réactualisation dans les schémas itératifs asynchrones de la Définition 2.1. Notons que cette nouvelle propriété n'implique pas nécessairement une augmentation du nombre d'accès à une mémoire commune mais offre au contraire la possibilité d'acquérir la valeur courante de chaque composante du vecteur itéré lorsque cela est nécessaire. Par contre le nombre de communications peut être plus important dans le cas d'une architecture à mémoire distribuée.

Pour illustrer le concept de communication flexible, nous présentons brièvement deux manières de mettre en œuvre ces algorithmes sur une architecture à mémoire distribuée. La première manière correspond au cas où les processeurs envoient une requête aux autres processeurs au commencement de chaque réactualisation afin de recevoir leur état courant (cf. figure 2.8, où nous avons repris l'exemple simple présenté au sous-paragraphe 2.2 et qui consiste en la recherche du point fixe d'une application à l'aide de deux processeurs). La deuxième mise en œuvre correspond au cas où chaque processeur transmet la valeur courante des composantes qui lui sont assignées en fonction d'une politique fixée ; par exemple la valeur courante des composantes du vecteur itéré peut être communiquée aux autres processeurs lorsqu'un nombre déterminé de pas de calculs intermédiaires est atteint (cf. figure 2.9 pour un exemple identique à celui considéré ci-dessus). Toutes ces mises en œuvre sont détaillées au chapitre 5.1.

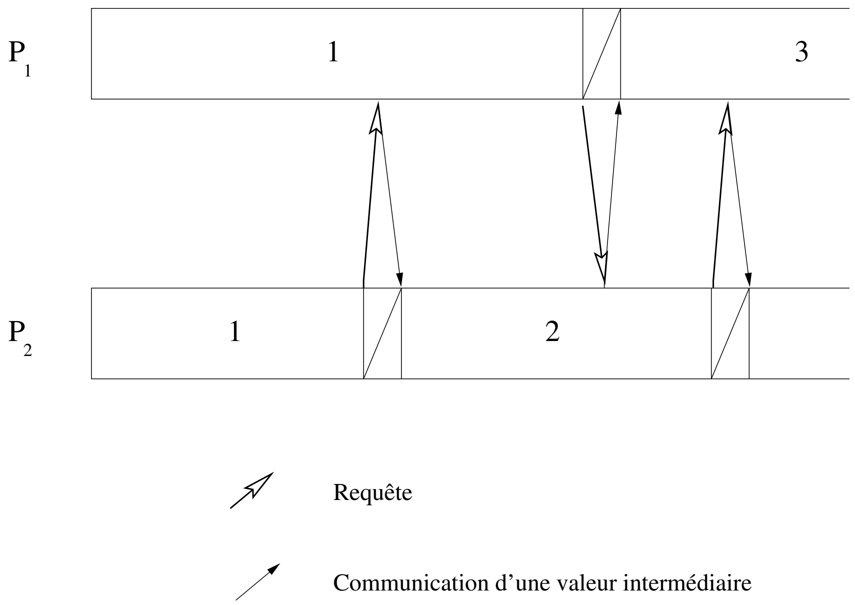


FIG. 2.8 – *Itérations asynchrones avec communication flexible, mise en œuvre avec des requêtes*

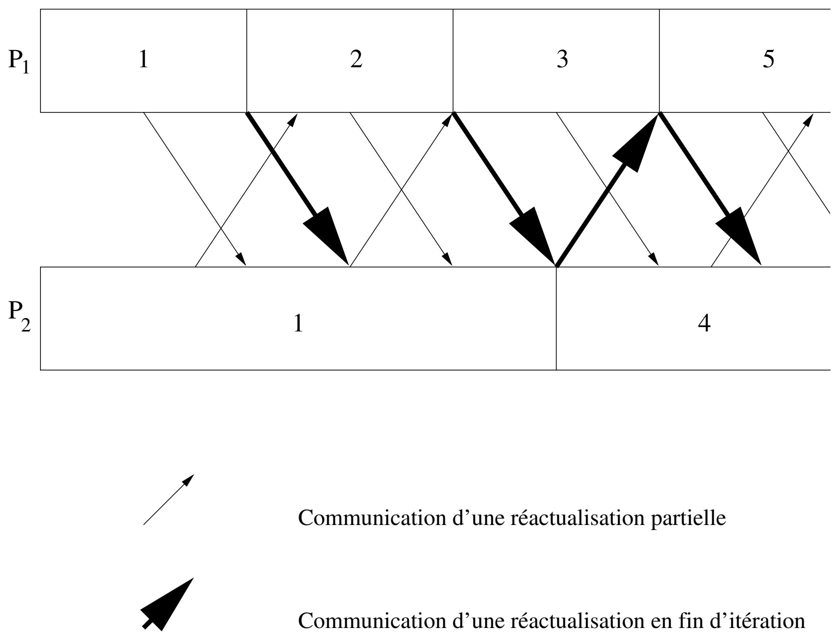


FIG. 2.9 – *Itérations asynchrones avec communication flexible, cas d'une politique fixée*

2.3.6 Un premier modèle d'itérations asynchrones avec communication flexible

Nous présentons maintenant un premier modèle d'itérations asynchrones avec communication flexible. Nous donnerons au sous-paragraphe suivant un second modèle qui selon nous est plus général et permet de décrire en particulier des algorithmes beaucoup plus complexes.

Les itérations asynchrones avec communication flexible sont étroitement liées aux concepts de $\mathcal{A}$-sur application, $\mathcal{A}$-sous-application, sur application et sous-application (cf. Annexe A) qui permettent d'engendrer des séquences monotones de vecteurs. L'utilisation de valeurs résultant de pas intermédiaires de réactualisation présente un intérêt tout particulier dans ce contexte car elle peut accélérer la convergence. De manière plus explicite, les méthodes itératives asynchrones avec communication flexible sont basées sur l'utilisation d'applications de point fixe qui approchent la solution des sous-problèmes. Ces applications peuvent être construites à l'aide de processus itératifs.

A titre d'exemple, nous avons considéré dans [MEBS98] un problème de diffusion non linéaire. La discrétisation de ce problème avec des schémas adéquats donne le système non linéaire:

$$\mathcal{A}(x^*) = 0, \tag{2.7}$$

où l'application $\mathcal{A}$ est une M-fonction. Soit F l'application de point fixe associée (cf. Remarque A.2 de l'Annexe A) et $n, \alpha \in N$ avec $\alpha \leq n$ et la décomposition de R^n en $\prod_{i=1}^{\alpha} R^{n_i}$, $\sum_{i=1}^{\alpha} n_i = n$. En introduisant la matrice bloc-diagonale $C(x)$ dérivée de la méthode de Newton nous avons montré dans [MEBS98] que l'application $\check{F}$ définie de la manière suivante est une $\mathcal{A}$-

sous-application associée à F.

$$\check{F}_i(x) = x_i^q, \ \forall i \in \{1, \ldots, \alpha\}, \tag{2.8}$$

où q est un entier positif, $x_i^0 = x_i$, et x_i^q est le q-ème iteré produit par l'algorithme :

$$x_i^q = x_i^{q-1} - C_i^{-1}(x_i^{\rho'(q-1)}).\,\mathcal{A}_i(x_i^{q-1}; x), q = 1, 2, \ldots \tag{2.9}$$

où $0 \leq \rho'(q-1) \leq q-1$, et $C_i(x)$ est le i-ème bloc diagonal de la matrice $C(x)$, $\mathcal{A}_i$ est la i-ème bloc-composante de l'application $\mathcal{A}$ et $(x_i^{q-1}; x)$ est le vecteur de $\prod_{i=1}^{\alpha} R^{n_i}$, dont le i-ème sous-vecteur est égal à x_i^{q-1}, les autres sous-vecteurs étant égaux à x_l, $l = 1, ..., \alpha$, $l \neq i$.

Nous donnons ci-dessous une première définition des itérations asynchrones avec communication flexible.

Définition 2.4 : *Soit $\mathcal{A}$ une M-fonction continue surjective et soit F l'application de point fixe associée (cf. Remarque A.2 de l'Annexe A). 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 (cf. Définitions A.25 et A.29 de l'Annexe A) sur $\prod_{i=1}^{\alpha} R^{n_i}$ et au point initial $x(0)$ satisfaisant $x(0) \leq F(x(0))$ est une suite $\{x(j)\}$ de vecteurs telle que pour $i = 1, ..., \alpha$:*

$$\begin{cases} x_i(j) = \check{F}_i(\tilde{x}(j)), \ si \ \{i\} = S(j), \\ x_i(j) = x_i(j-1), \ si \ \{i\} \neq S(j), \end{cases} \tag{2.10}$$

où $\mathcal{S} = \{S(j) | j = 1, 2, ...\}$ est une suite de singletons de $\{1, ..., \alpha\}$ satisfaisant pour $i = 1, ..., \alpha$:
(a) i apparaît un nombre infini de fois dans $\mathcal{S}$;
le vecteur $\tilde{x}(j)$ est implicitement défini par :

$$\tilde{x}_l(j) \in \langle \max\{x_l(I_l(j)), \tilde{x}_l(q)\}, x_l(j-1)\rangle_l, \ l = 1, ..., \alpha, \tag{2.11}$$

$\langle , \rangle_l$ étant un segment d'ordre dans R^{n_l} (cf. la relation (A.4)) et $q = \max_{k \in K_{s(j)}(j)} k$, $\{K_i(j)\}$ satisfaisant pour $i \in \{1, ..., \alpha\}$ et $j = 1, 2, ...,$ $K_i(j) = \{k \in N \mid S(k) = \{i\}, 0 \leq k < j\}$ (on pose $q = 0$ et $\tilde{x}(0) = x(0)$, si $K_{S(j)}(j) = \emptyset$), et $\mathcal{I} = \{(I_1(j), ..., I_\alpha(j)) | j = 1, 2, ...\}$ est une suite d'éléments de N^α satisfaisant pour $i = 1, ..., \alpha$:
(b) $0 \leq I_i(j) \leq j-1, j = 1, 2, ...$
(c) $I_i(j) = j-1$, si $\{i\} = S(j)$;
(d) $I_i(j)$ tend vers l'infini lorsque j tend vers l'infini.

On définit de manière analogue en renversant les inégalités une méthode itérative asynchrone avec communication flexible relative à une sur application $\hat{F}$, du premier ou du second type associée à une application de point fixe F (cf. Définitions A.23 et A.27 de l'Annexe A) et au point initial $x(0) \in P$" tel que $x(0) \geq F(x(0))$. On définit aussi de manière analogue une itération asynchrone avec communication flexible relative à une $\mathcal{A}$-sur application $\hat{F}$, ou une $\mathcal{A}$-sous-application $\check{F}$ qui est du premier ou du second type associée à une application de point fixe F (cf. Définitions A.22, A.24, A.26, et A.28 de l'Annexe A).

Ce modèle a été utilisé avec succés pour la résolution d'équations aux dérivées partielles grâce à des méthodes de sous-domaine et pour la résolution de problèmes d'optimisation de type flot dans les réseaux grâce à des méthodes de relaxation approchée, comme nous le verrons au chapitre 6.

Remarque 2.8 : *L'originalité de cette nouvelle classe de méthodes provient des communications flexibles entre processeurs. En effet dans ce nouveau modèle le sous-vecteur $\tilde{x}_l(j)$ communiqué par le processeur P_l au processeur P_i et utilisé lors de la réactualisation de $x_i(j)$ prend sa valeur dans le segment conique $\langle \max\{x_l(I_l(j)), \tilde{x}_l(q)\}, x_l(j-1)\rangle$, où $\tilde{x}_l(q)$ représente la valeur du sous-vecteur x_l utilisée lors de la dernière réactualisation de x_i, le terme $x_l(I_l(j))$ traduit le comportement non déterministe de l'algorithme itératif asynchrone et $x_l(j-1)$ est la valeur de x_l à l'itération $j-1$. L'introduction des intervalles d'ordre nous permet de prendre en compte le cas où $\tilde{x}_l(j)$ 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 produite par exemple par quelques pas du processus itératif qui engendre $\check{F}$. La valeur $\tilde{x}_l(j)$ 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. Il en résulte une meilleure interaction entre communication et calcul (cf. figures 2.8 et 2.9) ; cette propriété est particulièrement intéressante dans le cas d'une convergence monotone.*

Remarque 2.9 : *Afin de simplifier la présentation nous avons supposé que les sous-ensembles $S(j)$ sont des singletons. Cette hypothèse ne diminue en rien la portée des itérations asynchrones avec communication flexible (cf. [EBGMS96]). En effet le cas général où les $S(j)$ sont des sous-ensembles quelconques de $\{1, ..., \alpha\}$ peut être aisément modifié pour rentrer dans le cadre des hypothèses définies dans ce paragraphe. A cette fin, il suffit d'ordonner arbitrairement les réactualisations qui étaient simultanées en modifiant les suites $\mathcal{S}$ et $\mathcal{I}$.*

Remarque 2.10 : *L'hypothèse (c) de la Définition 2.4 n'est pas très restrictive, elle est au contraire tout à fait naturelle dans le cas d'une allocation statique des charges de calcul aux processeurs ; en effet, tout processeur utilise alors la dernière valeur de chaque composante qui est disponible dans sa mémoire locale.*

Remarque 2.11 : *Contrairement à la formulation des itérations asynchrones donnée dans la Définition 2.1, le modèle (2.10) permet de considérer le cas où les valeurs des composantes d'un même sous-vecteur qui sont utilisées lors d'une réactualisation correspondent à des itérations différentes.*

Remarque 2.12 : *Si le modèle de la Définition 2.4 permet de prendre en compte les itérés partiels, il possède toutefois une limitation. En effet, dans l'équation (2.11) le segment conique est majoré par $x_l(j-1)$ et non par $x_l(j)$ ce qui empèche de prendre en compte tout itéré partiel postérieur à $x_l(j-1)$ et qui peut néanmoins être disponible pour le calcul de $x_i(j)$. Le modèle du paragraphe suivant ne possède pas cet inconvénient.*

2.3.7 Un second modèle d'itérations asynchrones avec communication flexible

Nous présentons dans ce sous-paragraphe un modèle d'itérations asynchrones avec communication flexible que nous avons introduit dans [EBSMG96] et qui est selon nous plus général que le modèle précédent ; il permet aussi de décrire des algorithmes qui ne peuvent être décrit par le premier modèle.

Définition 2.5 : *Soit $\mathcal{A}$ une M-fonction continue surjective et soit F l'application de point fixe associée (cf. Remarque A.2 de l'Annexe A). Une itération asynchrone avec communication flexible relative à une sous-application $\check{F}$ du premier ou du second type associée à F sur $\prod_{i=1}^{\alpha} R^{n_i}$ et au point initial $x(0)$ satisfaisant $x(0) \leq F(x(0))$ est une suite $\{x(j)\}$ de vecteurs telle que pour $i = 1, ..., \alpha$:*

$$\begin{cases} x_i(j) \in \langle \check{F}_i'(x_1(I_1(j)), ..., x_\alpha(I_\alpha(j))), \check{F}_i(x_1(I_1(j)), ..., x_\alpha(I_\alpha(j))) \rangle \ si \ \{i\} = S(j), \\ x_i(j) = x_i(j-1), \ \ si \ \{i\} \neq S(j), \end{cases} \tag{2.12}$$

où $\check{F}'$ est une sous-application continue pour l'ordre (cf. Définition A.21 de l'Annexe A) telle que pour tout $i \in \{1, .., n\}$ et x tel que $x_i \leq F_i(x)$ on a : $\check{F}_i'(x) \in \langle x_i, \check{F}_i(x) \rangle_i$, $\mathcal{S} = \{S(j) | j = 1, 2, ...\}$ est nous le rappelons une suite de singletons de $\{1, ..., \alpha\}$ et $\mathcal{I} = \{(I_1(j), ..., I_\alpha(j)) | j = 1, 2, ...\}$ est une suite d'éléments de N^α satisfaisant pour $i = 1, ..., \alpha$:

(a) i apparaît un nombre infini de fois dans $\mathcal{S}$;

(b) $0 \leq I_i(j) \leq j - 1, j = 1, 2, ...$

(c) $I_i(j) = j - 1$, si $\{i\} = S(j)$;

(d) $I_i(j)$ tend vers l'infini lorsque j tend vers l'infini.

On définit de manière analogue une itération asynchrone avec communication flexible relative à une sur application $\hat{F}$ du premier ou du second type associée à F sur $\prod_{i=1}^{\alpha} R^{n_i}$ et au point initial $x(0)$ satisfaisant $x(0) \geq F(x(0))$ à l'aide des relations suivantes :

$$\begin{cases} x_i(j) \in \langle \hat{F}_i(x_1(I_1(j)), ..., x_\alpha(I_\alpha(j))), \hat{F}_i'(x_1(I_1(j)), ..., x_\alpha(I_\alpha(j))) \rangle \text{ si } \{i\} = S(j), \\ x_i(j) = x_i(j-1), \text{ si } \{i\} \neq S(j), \end{cases} \tag{2.13}$$

Remarque 2.13 : *Dans ce second modèle d'itérations asynchrones avec communication flexible, la valeur de $x_i(j)$ qui est reçue ou acquise par un autre processeur que celui qui l'a calculée peut correspondre à une nouvelle réactualisation de x_i obtenue grâce à la sous-application $\check{F}$ ou par quelques pas d'un processus itératif permettant de construire la sous-application $\check{F}$, la seule restriction étant que la valeur de $x_i(j)$ est comprise dans le segment d'ordre $\langle \check{F}_i'(x_1(I_1(j)), ..., x_\alpha(I_\alpha(j))), \check{F}_i(x_1(I_1(j)), ..., x_\alpha(I_\alpha(j))) \rangle$. Ainsi la valeur $x_i(j)$ produite par l'algorithme est bien un itéré ou un itéré partiel provenant de calculs en cours. Les itérés partiels peuvent être transmis ou acquis à n'importe quel instant et sans règle fixée a priori.*

Remarque 2.14 : *Contrairement au modèle précédent ce modèle d'itérations asynchrones avec communication flexible permet d'utiliser tous les itérés partiels produits par l'algorithme (cf Remarque 2.12). Notons que les itérés et les itérés partiels sont étiquetés par un numéro.*

Remarque 2.15 : *le modèle de la Définition 2.5 permet aussi de modéliser des algorithmes asynchrones pour lesquels l'opérateur de point fixe n'est pas défini de manière classique mais prend sa valeur dans un segment d'ordre. Ce type d'opérateur peut être engendré par exemple à la suite d'un tirage aléatoire sur plusieurs opérateurs de point fixe.*

Remarque 2.16 : *Le modèle de la Définition 2.5 ainsi que celui de la Définition 2.4 présentent le grand avantage de combiner le formalisme très général des méthodes par blocs à la possibilité d'utiliser dans les calculs la valeur courante de chaque composante du vecteur itéré.*

2.3.8 Modèles de type espace d'état

Parmi les autres modèles de la littérature, on peut citer le modèle de Kaszkurewicz et Bhaya (cf. [KB90] et [BKM91]) pour la résolution de systèmes d'équations linéaires pertubées par un opérateur bloc-diagonal non linéaire Lipschitzien à l'aide de méthodes de décomposition asynchrones avec des délais bornés. Ce modèle repose sur un processus d'augmentation de la dimension du système.

Beidas et Papavassilopoulos ont proposé dans [BP93] et [BP94] un modèle linéaire asynchrone avec des retards bornés de nature stochastique qui sont dus aux communications.

Ces modèles conduisent à des formulations de type espace d'état qui sont très lourdes.

2.4 Conclusion

Contrairement au schémas itératifs synchrones les algorithmes itératifs asynchrones réactualisent les composantes du vecteur itéré sans ordre ni synchronisation. En ce sens ils n'ont aucun équivalent séquentiel et sont donc de nature typiquement parallèle.

Les restrictions imposées à ce type d'algorithme sont très faibles : aucune composante du vecteur itéré ne doit cesser d'être réactualisée et les valeurs anciennes des composantes doivent être abandonnées au fur et à mesure que les calculs progressent. Par voie de conséquence les itérations asynchrones ont un formalisme très général recouvrant les schémas de calcul classiques tels que Jacobi parallèle, série-parallèle... Un schéma de calcul ordonné n'étant après tout qu'un cas particulier de schéma chaotique.

L'asynchronisme se formalise essentiellement par l'introduction de retards ; les retards ne signifiant nullement que les itérations asynchrones sont des méthodes parallèles inefficaces.

Tout au long de ce chapitre, nous avons présenté plusieurs modèles d'itérations asynchrones : itérations partiellement asynchrones, itérations totalement asynchrones, itérations asynchrones avec mémoire, et itérations asynchrones avec communication flexible (qui constituent l'essentiel de notre contribution en ce qui concerne les aspects de modélisation). Ces modèles sont plus ou moins sophistiqués suivant le degré d'asynchronisme qui est toléré et la complexité des calculs ou des échanges de données. En particulier, les modèles proposés pour les itérations asynchrones avec communication flexible présentent l'intérêt de combiner le formalisme très général des méthodes par blocs à la plus grande liberté en matière d'échange de données entre processeurs (qu'il s'agisse de communication par passage de message ou de processus d'écriture/lecture dans une mémoire partagée).

Chapitre 3

Convergence

La girafe semble un mécanisme construit par assemblage de morceaux provenant de machines hétérogènes, mais qui fonctionne cependant à la perfection. Monsieur Palomar, continuant à observer les girafes et leur course, se rend compte qu'une harmonie compliquée commande tout ce trépignement dysharmonique ...
Italo Calvino
Palomar

L'étude de la convergence est un problème complexe en raison de l'absence de synchronisation et donc du comportement chaotique des schémas itératifs, ce qui se traduit au niveau du modèle par des retards pouvant être éventuellement infinis. Cependant un très grand nombre de résultats généraux ont été établis dans des contextes très variés ; nous les présentons dans ce chapitre ainsi que notre contribution personnelle.

3.1 Systèmes d'équations

3.1.1 Le cas linéaire, une condition nécessaire et suffisante de convergence

Soit le système d'équations

$$\mathcal{A}x^* = z, \tag{3.1}$$

où x^* et z sont deux vecteurs de R^n.

Dans le cas linéaire, Chazan et Miranker ont établi le résultat suivant :

Théorème 3.1 : *(cf. [CM69]) Tout algorithme asynchrone défini par le modèle (2.2), où F correspond alors à une matrice de type Jacobi par point, converge vers la solution du problème (3.1) si et seulement si $\mathcal{A}$ est une H-matrice (cf. Définition A.6 de l' Annexe A).*

Remarque 3.1 : *Ce résultat de convergence qui est historiquement le premier a avoir été publié est le seul pour lequel la condition suffisante de convergence est aussi une condition nécessaire.*

Remarque 3.2 : *dans le cas où $\mathcal{A}$ est une matrice définie positive, Donnely (cf. [Don71]) a établi un résultat de convergence pour des schémas chaotiques périodiques surrelaxés pour lesquels on a :*

$$S(j) = j\,mod(n), \tag{3.2}$$

3.1.2 Non linéarité et ordre partiel

Remarque 3.3 : *Soit à résoudre le système non linéaire $\mathcal{A}x^* = z$ de l'équation (3.1). Soit $n, \alpha \in N$, l'ensemble des entiers naturels et $\alpha \leq n$. Nous considérons la décomposition de R^n en $\prod_{i=1}^{\alpha} R^{n_i}$ avec $\sum_{i=1}^{\alpha} n_i = n$. Chaque sous-espace R^{n_i} est muni de l'ordre partiel naturel, ou encore ordre composante par composante, associé au cône $K_i = R_+^{n_i}$ des vecteurs de composantes non negatives dans R^{n_i} (cf. Définition A.2 de l'Annexe A). Soit le vecteur $x \in R^n$ on a alors la décomposition suivante de x :*

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

et la décomposition de $\mathcal{A}$:

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

Par la suite, le vecteur $\{x_1, \ldots, x_{i-1}, \hat{x}_i, x_{i+1}, \ldots, x_\alpha\}$ sera aussi noté $(\hat{x}_i; x)$.

On considère pour tout $i \in \{1, ..., \alpha\}$, le problème :

$$\mathcal{A}_i(\hat{x}_i; x) = 0, \tag{3.3}$$

et l'application F de R^n dans R^n qui satisfait :

$$F(x) = \hat{x} = \{\hat{x}_1, \ldots, \hat{x}_i, \ldots, \hat{x}_\alpha\}, \tag{3.4}$$

où $\hat{x}_i$ est solution de (3.3).

Notre contribution a porté sur les résultats suivants :

Théorème 3.2 : *(cf. [EB94a]) Si $\mathcal{A}$ est continue, diagonale monotone croissante et hors diagonale monotone décroissante (cf. Définitions A.7 et A.9 de l' Annexe A) et s'il existe une $\mathcal{A}$-sur solution $y(0)$ et une $\mathcal{A}$-sous-solution $x(0)$ (cf. Définitions A.13 et A.14 de l' Annexe A), alors l'application de point fixe par point F définie dans la Remarque 3.3 (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*

$$\overline{F}_i(x) = \ max_{x \in F_i(x)} x \ \text{et} \ \underline{F}_i(x) = \ min_{x \in F_i(x)} x, \forall i \in \{1, ..., n\}, \forall x \in D, \tag{3.5}$$

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 (2.2) et associée à $\overline{F}$ ou $\underline{F}$ converge de manière monotone à partir de y^0 ou x^0, respectivement.

Théorème 3.3 : *(cf. [EB90]) Si $\mathcal{A}$ est continue, diagonale monotone strictement croissante et hors diagonale monotone décroissante (cf. Définition A.8 de l' Annexe A) 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 dans la Remarque 3.3 est univoque et tout algorithme asynchrone par point défini par le modèle (2.2) 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 3.4 : *L'application F est continue et monotone croissante sur un segment d'ordre délimité par la $\mathcal{A}$-sous-solution et la $\mathcal{A}$-sur solution (cf. [EB90]).*

Théorème 3.4 : *(cf. [EB90]) Si $\mathcal{A}$ est une M-fonction continue surjective (cf. Définitions A.11, A.12, et Théorème A.1 de l' Annexe A), alors toute itération asynchrone définie par le modèle (2.2) et associée à l'application de point fixe F définie dans la Remarque 3.3 converge vers l'unique solution du problème (3.1) quel que soit le point initial.*

Théorème 3.5 : *(cf. [EB94a]) 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 (2.2) et associée à l'application de point fixe $F : R^n \to R^n$ telle que :*

$$F(x) = x - \frac{1}{\alpha}\mathcal{A}x, \forall x \in R^n, \tag{3.6}$$

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

Remarque 3.5 : *L'application F est continue et monotone croissante sur $D = \{x \in R^n \mid x^0 \leq x \leq y^0\}$. De plus on a :*

$$x^0 \leq F(x^0) \leq F(y^0) \leq y^0. \tag{3.7}$$

Les schémas itératifs asynchrones considérés sont alors de type Richardson.

Théorème 3.6 : *(cf. [Mie86]) Si $\mathcal{A}$ est une M-fonction continue surjective, alors toute itération asynchrone définie par le modèle (2.2) et relative à une $\mathcal{A}$-sur application ou une $\mathcal{A}$-sous-application du premier ou du second type (cf. Définitions A.22, A.24, A.26, et A.28 de l'Annexe A) associée à l'application de point fixe F de la Remarque 3.3 converge de manière monotone vers la solution du problème (3.1) 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 2.3.6.

Théorème 3.7 : *(cf. [MEBS98]) Si $\mathcal{A}$ est une M-fonction continue surjective, alors toute itération asynchrone avec communication flexible définie par le modèle (2.10) et relative à une $\mathcal{A}$-sur application ou une $\mathcal{A}$-sous-application du premier ou du second type associée à l'application de point fixe F de la remarque 3.3 converge de manière monotone vers la solution du problème (3.1) à partir d'une $\mathcal{A}$-sur solution ou d'une $\mathcal{A}$-sous-solution, respectivement.*

Le résultat suivant généralise directement un résultat de convergence pour les algorithmes itératifs asynchrones avec communication flexible appliqués à des problèmes d'optimisation non linéaire de type flot dans les réseaux.

Théorème 3.8 : *(cf. [EBSMG96]) 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 (2.12) et relative à une sur application ou une sous-application du premier ou du second type (cf. Définitions A.23, A.25, A.27, et A.29 de l'Annexe A) 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.*

Remarque 3.6 : *Les résultats précédents s'appliquent à une large classe de problèmes incluant des systèmes issus de la discrétisation d'équations aux dérivées partielles et des problèmes d'optimisation comme nous le verrons au Chapitre 6. En particulier, Le Théorème 3.2 généralise un premier résultat pour les méthodes de relaxation asynchrones pour la résolution de problèmes de flot convexes dans les réseaux (cf. [BEB87]). De même le Théorème 3.5 généralise un résultat pour les méthodes de gradient asynchrones toujours dans le cas des problèmes de flot dans les réseaux (cf. [EB96a]). Dans le même contexte, le Théorème 3.8 généralise les résultats de [EBSMG96]. Les résultats du Théorème 3.7 ont été étendus au cas des méthodes de type "multisplitting" dans [EBSM97].*

3.1.3 Cas d'opérateurs H-accrétifs

Pour des problèmes du type :

$$\Lambda^d(x) + \Lambda(x) \ni 0, \;\; x \in E, \tag{3.8}$$

où E est un espace de Banach réflexif produit fini d'une famille d'espaces de Banach $\{E_i\}$, $i \in \{1, ..., \alpha\}$, de norme respective $\|.\|_i$, l'application $\Lambda \; : \; D(\Lambda) \subset E \to E$ est une application univoque, et l'application Λ^d est une multi-application diagonale (cf. paragraphe A.1.3 de l'Annexe A), Miellou et Spiteri ont établi le résultat suivant :

Théorème 3.9 : *(cf. [MS85b] et [Spi84]) Si l'application Λ est H-accrétive et les composantes par bloc de l'application Λ^d sont m-accrétives (cf. Définitions A.31 et A.32 de l'Annexe A), alors tout algorithme asynchrone défini par (2.2) où F est définie d'une manière analogue à celle de la Remarque 3.3 converge vers la solution unique du problème (3.8) à partir d'un élément quelconque de E.*

Remarque 3.7 : *Ce résultat de convergence repose sur des propriétés de contraction (cf. aussi [MS85a]).*

3.2 Problèmes de point fixe

Dans ce paragraphe nous effectuons un rappel des principaux résultats de convergence pour les itérations asynchrones appliquées à la résolution de problèmes modélisés directement comme la recherche d'un point fixe d'un opérateur non linéaire. Les résultats présentés sont issus des travaux pionniers en la matière de Miellou et de son équipe au Laboratoire de Calcul Scientifique de Besançon, à savoir : Bahi, Boulbrachêne, Comte, Cortey-Dumont, El Tarazi, Jacquemard, Laouar, et Spiteri.

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 a la décomposition suivante de $x \in E$:

$$x = \{x_1, \ldots, x_\alpha\}. \tag{3.9}$$

La norme vectorielle canonique ψ de $E = \prod_{i=1}^{\alpha} E_i$ dans R_+^α fait correspondre à tout $x \in E$:

$$\psi(x) = (\|x_1\|_1, ..., \|x_\alpha\|_\alpha), \tag{3.10}$$

Soit F une application de $D(F) \subset E$ à valeurs dans $D(F)$, telle que :

$$D(F) \neq \emptyset, \tag{3.11}$$

où $\emptyset$ est l'ensemble vide. F est décomposée de la manière suivante :

$$F(x) = \{F_1(x), \ldots, F_{\alpha(x)}\}. \tag{3.12}$$

On considère le problème de point fixe :

$$x^* = F(x^*). \tag{3.13}$$

Les théorèmes suivants comptent parmi les principaux résultats de convergence pour les itérations asynchrones.

3.2.1 Contraction

Théorème 3.10 : *(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.36 de l'Annexe A), alors l'algorithme itératif asynchrone défini par (2.2) appartient à $D(F)$ et converge vers x^*.*

Remarque 3.8 *Ce résultat peut être étendu au cas des applications de point fixe avec paramètre de relaxation $\omega \in]0, \frac{2}{1+\rho(T)}[$ (où T est la matrice de contraction de F en x^*), qui sont notées F_ω, de domaine $D(F) \subset E$, à valeurs dans $D(F)$, et définies par :*

$$F_\omega(x) = (1 - \omega)x + \omega F(x), \tag{3.14}$$

Théorème 3.11 : *(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 $\|,\|_T^w$. (cf. paragraphe A.2.2 de l'Annexe A), alors l'algorithme itératif asynchrone défini par (2.2) appartient à $D(F)$ et converge vers x^*.*

Nous donnons ci-dessous un résultat de convergence pour les algorithmes itératifs asynchrones avec mémoire.

Théorème 3.12 : *(cf. [Bau78]) Si $D(F) \neq \emptyset$ et si F est m-contractante sur un sous-ensemble non vide D de $D(F) \subset E$, c'est à dire si il existe une matrice non négative T de rayon spectral inférieur à 1 telle que pour tout $x^1, ..., x^m, y^1, ..., y^m \in D$,*

$$\psi(F(x^1, ..., x^m) - F(y^1, ..., y^m)) \leq T \ max \ [\psi(x^1 - y^1), ..., \psi(x^m - y^m)], \tag{3.15}$$

et $F(D^m) \subset D$, alors tout algorithme itératif asynchrone avec mémoire défini par (2.4) associé à l'application F et initialisé par un ensemble quelconque de vecteurs de D^m converge vers x^* qui est l'unique point fixe de F.

Pour d'autres résultats de convergence concernant les itérations asynchrones avec mémoire, on pourra se référer à [ET82] dans un contexte de contraction classique, ainsi qu'à [ET84] dans un contexte d'ordre partiel.

Nous présentons au paragraphe suivant quelques-uns des résultats les plus importants dans un contexte d'ordre partiel.

3.2.2 Ordre partiel

Théorème 3.13 : *(cf. [Mie75b]) Si chaque espace de Banach E_i est muni d'un ordre partiel (cf. Définition A.2 de l'Annexe A), E étant muni de l'ordre partiel noté $\leq$, soient y^0 et x^0 deux éléments de E tels que $x^0 \leq y^0$, définissant le segment d'ordre $< x^0, y^0 >$ (cf. Définition A.3 de l'Annexe A), si l'application de point fixe F de E 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 (2.2) et initialisé par y^0 ou x^0 converge de manière monotone vers un point fixe de F.*

Remarque 3.9 *Dans [Mie75b] ce résultat a été donné pour les applications F de $< x^0, y^0 > \times < x^0, y^0 >$ à valeurs dans $< x^0, y^0 >$ qui sont croissantes sur $< x^0, y^0 > \times < x^0, y^0 >$. On définit dans ce cas des itérations asynchrones doubles du type $(F, x(0), \mathcal{S}, \mathcal{I}^1, \mathcal{I}^2)$.*

Les aspects contraction et ordre partiel peuvent être combinés, on a alors le résultat suivant.

Théorème 3.14 : *(cf. [Jac77]) Sous les hypothèses du Théorème 3.13 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 :*

$$x \leq y \Rightarrow F(y) - F(x) \leq T(y - x), \forall x, y \in E, \tag{3.16}$$

alors F est continue et l'algorithme itératif asynchrone défini par (2.2) 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.36 de l'Annexe A), alors le point fixe de F est unique.

3.2.3 Le théorème de convergence asynchrone de Bertsekas

Le théorème de convergence asynchrone de Bertsekas (cf. [Ber83] et [BT89]) est un résultat de convergence original et très général, c'est aussi un outil très puissant pour prouver la convergence d'algorithmes itératifs asynchrones pour des applications très variées. Le théorème de convergence asynchrone de Bertsekas donne un ensemble de conditions suffisantes qui assurent la convergence d'algorithmes asynchrones pour des problèmes de point fixe. Contrairement aux résultats présentés précédemment qui sont basés sur l'étude d'une suite de vecteurs, ce résultat est basé sur l'étude d'une suite d'ensembles auxquels les éléments de la suite de vecteurs appartiennent. Cette approche tire son origine de la théorie de la stabilité de Lyapunov ; son avantage est de fournir un cadre plus abstrait pour l'analyse de la convergence des itérations asynchrones et d'englober les aspects contraction et ordre partiel. Les hypothèses du théorème de convergence sont présentées ci-dessous.

Hypothèse 3.1 : *(cf. [BT89]) Il existe une suite de sous-ensembles non vides $\{X(j)\}$ de l'ensemble X telle que :*

$$...X(j+1) \subset X(j) \subset ... \subset X,$$

satisfaisant les conditions suivantes :

Condition de convergence des itérations synchrones :

On a :

$$F(x) \subset X(j+1), \forall j \text{ et } x \in X(j).$$

De plus, si $\{x(j)\}$ est une suite de vecteurs telle que $x(j) \in X(j)$ pour tout j alors $\{x(j)\}$ tend vers un point fixe de l'application F.

Condition topologique :

Pour tout j, il existe des ensembles $X_i(j) \subset X_i$ tels que

$$X(j) = X_1(j) \times X_2(j) \times ... \times X_n(j).$$

où $\times$ représente le produit Cartésien.

Théorème 3.15 : Théorème Général de Convergence Asynchrone *(cf. [Ber83] et [BT89]) Si la suite d'ensembles $\{X(j)\}$ satisfait les conditions énoncées ci-dessus et si le vecteur initial $x(0)$ appartient à $X(0)$, alors toute itération totalement asynchrone définie suivant le modèle (2.5) converge vers un point fixe de F.*

Remarque 3.10 *Ce résultat a été montré de manière récursive en prouvant que pour tout j il existe un $t(j)$ tel que pour $t \geq t(j)$, les vecteurs $x(t)$ appartiennent à $X(j)$ et les vecteurs $(x_1(\tau_1^i(t), ..., x_n(\tau_n^i(t)))$ utilisés dans les calculs appartiennent à $X(j)$.*

On constate que l'approche développée par Bertsekas est particulièrement originale car elle fait le lien avec la convergence des algorithmes itératifs synchrones. Cependant si ce résultat fournit un cadre abstrait pour faciliter l'étude de la convergence des algorithmes itératifs asynchrones, il ne s'applique pas de manière directe, l'obtention d'un résultat particulier de convergence demandant notamment de définir les ensembles $X(j)$. Nous donnons ci-dessous quelques exemples d'application du théorème de convergence asynchrone de Bertsekas. Les exemples attestent la généralité de ce résultat.

Exemple 1 : Cas des applications contractantes par rapport à une norme pondérée

Nous considérons une norme pondérée du type :

$$\| x \|_{\infty}^{w} = \max_{i=\{1,\ldots,n\}} \frac{| x_i |}{w_i},$$

sur R^n, où $w \in R^n$ est un vecteur dont les composantes sont positives. Notons que la sphère de rayon unité de la norme présentée ci-dessus satisfait la condition topologique énoncée précédemment. Si $F : R^n \to R^n$ est une application contractante par rapport à la norme présentée ci-dessus et si $X_i = R$ pour $i = 1, \ldots, n$ et $X = R^n$, alors on peut définir les ensembles :

$$X(j) = \{x \in R^n \mid \ \| x - x^* \|_{\infty}^{w} \leq \nu^j \| x(0) - x^* \|_{\infty}^{w}\}, j = 1, 2, \ldots \tag{3.17}$$

où x^* est l'unique point fixe de F et $\nu < 1$ est le facteur de contraction. Il est clair que l'Hypothèse 3.1 est alors satisfaite.

Exemple 2 : Cas des applications monotones

Si $F : R^n \to R^n$ est une application monotone, X_i est un sous-ensemble de $[-\infty, +\infty]$ pour $i = 1, \ldots, n$, x^* est l'unique point fixe de F, et il existe deux vecteurs de X notés $x(0)$ et $y(0)$ qui sont respectivement sous-solution et sur solution :

$$x(0) \leq F(x(0)) \leq F(y(0)) \leq y(0),$$

et tels que :

$$\lim_{j \to \infty} F^j(x(0)) = \lim_{j \to \infty} F^j(y(0)) = x^*.$$

où $F^j(.)$ résulte de j compositions de F avec lui-même. Il résulte de la monotonie de F que

$$F^j(x(0)) \leq F^{j+1}(x(0)) \leq x^* \leq F^{j+1}(y(0)) \leq F^j(y(0)), \forall j.$$

Alors on peut définir les ensembles :

$$X(j) = \{x \mid \ F^j(x(0)) \leq x \leq F^j(y(0))\},$$

qui satisfont l'Hypothèse 3.1.

Remarque 3.11 *Nous avons fait usage du Thèorème de Convergence Asynchrone de Bertsekas notamment dans [BEB87] et [EB96a] 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. Nous avons aussi utilisé ce résultat dans [EB96b] afin de construire une méthode originale de terminaison pour les algorithmes itératifs asynchrones (cf. paragraphe 4.5).*

Remarque 3.12 *Bertsekas et ses collègues et étudiants au MIT : Athans, Castanon, Eckstein, Tseng, et Tsitsiklis ont utilisé le théorème de convergence asynchrone de Bertsekas afin d'obtenir des résultats de convergence pour des algorithmes asynchrones divers appliqués a la résolution de problèmes très variés ; on peut citer notammemt la programmation dynamique (cf. [Ber82]), la recherche de chemins minimaux (cf. [BGM96b]), les problèmes d'optimisation de type assignation (cf. [Ber85]), les problèmes de flot dans les réseaux (cf. [BE87], [TBT90], [BC90], et [BCEZ95]), le routage optimal dans les réseaux de communication (cf. [TB86] et aussi [BC91] et [TBA86])*

Remarque 3.13 *On peut se reporter aux travaux de Miellou, Cortey-Dumont, et Boulbrachêne (cf. [MCDB90]) pour une extension de ces résultats.*

3.3 Autres résultats de convergence

Lubachewski et Mitra (cf. [LM86]) ont aussi établi un résultat suffisant de convergence pour des itérations asynchrones à retards bornés appliquées à la résolution de systèmes singuliers de type Markoviens. Leur modèle d'algorithme itératif asynchrone est très voisin des itérations partiellement asynchrones de Bertsekas (cf. Hypothèse 2.2 du Chapitre 2.3). Dans ce cas, $\mathcal{A}$ est une matrice irréductible, non négative, de rayon spectral unité ; c'est aussi une matrice stochastique (la somme des coefficients de chaque ligne est égale à 1), Lubachewski et Mitra supposent de plus qu'un coefficient diagonal de $\mathcal{A}$ est positif et donc que $\mathcal{A}$ est apériodique. La convergence est obtenue à une constante de proportionnalité près.

Le lecteur pourra aussi se reporter aux travaux de Frommer et Szyld [FS94], [FS97b], [FS97a], [FS98],et [Szy97], Bru [BMPS95], et B. Zhong-Zhi [ZDE95] en ce qui concerne les méthodes de "multisplitting" ou encore de multidécomposition asynchrones.

Pour finir, notons l'analogie entre les schémas itératifs et les systèmes dynamiques discrets et plus particulièrement entre les algorithmes itératifs asynchrones et les systèmes discrets avec des retards qui ne sont pas connus a priori et qui peuvent être variables dans le temps. Dans certains cas, l'étude de la convergence des schémas numériques peut tirer profit de l'étude de la stabilité ou de la stabilité asymptotique du système dynamique discret correspondant. Des résultats fondés sur la théorie de la stabilité de Lyapunov ont été présentés dans ce cadre notamment dans les références suivantes : [Tsi87], [KB90], et [BKM91].

3.4 Conclusion

Dans ce chapitre nous avons présenté plusieurs résultats de convergence pour les itérations asynchrones. Les méthodes considérées pouvant être de type relaxation par blocs, méthode alternée de Schwarz, "multi-splitting" ou multidécomposition, ou Richardson. Ces résultats ont porté tout d'abord sur les propriétés de l'application $\mathcal{A}$ du système $\mathcal{A}x^* = z$ à résoudre ; certains résultats étant originaux notamment ceux concernant les opérateurs diagonaux monotones croissants et hors-diagonaux monotones décroissants, ainsi que ceux relatifs aux itérations asynchrones avec communication flexible. Par la suite nous avons donné les principaux résultats de convergence pour des problèmes formulés directement comme la recherche d'un point fixe d'un opérateur. Pour finir nous avons énoncé le théorème de convergence asynchrone de Bertsekas qui est un outil très puissant pour montrer la convergence d'algorithmes itératifs asynchrones.

Bien que l'analyse de la convergence des itérations asynchrones soit complexe, nous disposons aujourd'hui d'un ensemble important d'outils pour traiter ce problème ; outils qui sont adaptés à de larges classes de méthodes itératives pour des applications très variées comme nous le verrons au Chapitre 6.

Chapitre 4

Terminaison

Dieu est dans le détail
Flaubert

La terminaison des algorithmes itératifs asynchrones relève des Mathématiques Appliquées, puisque la terminaison de l'algorithme itératif doit se produire lorsque le vecteur itéré est suffisamment proche d'une solution du problème ainsi que de l'Informatique, puisqu'une procédure spéciale doit être conçue afin de détecter la terminaison.

Ce problème a de fortes connexions avec celui de la terminaison de processus distribués. Toutefois le nombre d'itérations de l'algorithme peut être infini, les processus de calcul peuvent alors ne jamais être inactifs.

La terminaison présente de nombreuses difficultés particulièrement dans le cas d'architectures de type passage de message puisque dans ce contexte les processeurs possèdent uniquement des informations locales, il n'y a pas d'horloge globale, et les délais de communication peuvent être arbitrairement longs. Aussi, il existe relativement peu de méthodes de terminaison efficaces pour les algorithmes itératifs asynchrones.

De manière plus explicite et en se plaçant dans le contexte le plus général, l'état global relatif à la terminaison d'un algorithme itératif asynchrone, qui nous le rappelons est modélisé par des retards, ne peut être déduit d'un ensemble hétéroclite d'informations locales du type : $\|x_i(j) - x_i(j-1)\|_i \leq \epsilon$, si on considère l'écart entre deux valeurs successives du même sous-vecteur x_i du vecteur itéré ou du type : $\|A_{i1}x_1(I_1(j))+, ..., +A_{i\alpha}x_\alpha(I_\alpha(j)) - b_i\|_i \leq \epsilon$ si on considère des résidus. Il apparaît que les informations locales ne peuvent être assemblées dans un ordre quelconque si l'on veut établir formellement que la terminaison a bien eu lieu.

Dans ce chapitre nous présentons des solutions au problème de la terminaison ; nous présentons notamment notre contribution aux paragraphes 4.3 et 4.5.

4.1 Méthodes empiriques

Les méthodes de terminaison des itérations asynchrones les plus couramment utilisées sont souvent conçues de manière empirique. Une méthode souvent usitée consiste en l'observation par un processeur particulier de conditions de terminaison locales en chaque processeur. L'algorithme est arbitrairement terminé lorsque toutes les conditions locales sont satisfaites. On conçoit aisément que ce type de méthode ne puisse donner des résultats satisfaisants que dans le cas où le degré d'asynchronisme est très faible. En effet lorsque les retards dus aux communications ou au déséquilibre des tâches de calcul sont importants cette méthode simple ne peut qu'entrainer une terminaison prématurée.

Une autre méthode (cf. [BT89]) consiste en l'émission de messages de terminaison et de messages de redémarage par chaque processeur et en l'utilisation d'un processeur spécial qui collecte et centralise ces messages.

Dans une approche différente (cf. [ME89]), le schéma de terminaison échantillonne périodiquement l'état des processeurs et associe à chaque processeur une valeur Booléenne suivant la satisfaction du critère de terminaison local. Cette valeur locale est ensuite communiquée aux autres processeurs. L'état global est déduit en calculant le point fixe d'un opérateur Booléen au moyen d'un algorithme itératif asynchrone. Toutefois, cette approche nécessite que chaque processeur dispose d'une estimation du temps de commencement et du temps de terminaison de l'algorithme asynchrone de recherche du point fixe de l'opérateur Booléen.

Une autre méthode de terminaison (cf. [CZ91]) utilise des messages de terminaison et des acquittements de messages de terminaison. Dans ce schéma de terminaison, un processeur termine ses calculs si sa condition de terminaison locale est satisfaite et s'il a reçu des messages de terminaison ainsi que des acquittements de tous ses messages de terminaison provenant de tous les processeurs.

Il n'existe aucune preuve formelle de validité pour les méthodes de terminaison citées ci-dessus dans le cas le plus général. En effet, comme nous l'avons mentionné plus haut, pour une architecture informatique de type passage de message, tous les processeurs possèdent des informations locales, en particulier il n'y a pas d'horloge globale, de plus les messages peuvent être sujets à des retards arbitrairement grands.

4.2 Méthode de Bertsekas et Tsitsiklis

Une des solutions les plus intéressantes au problème de la terminaison des algorithmes itératifs asynchrones a été proposée par Bertsekas et Tsitsiklis au paragraphe 8.1 de [BT89] ainsi que dans [BT91]. Bertsekas et Tsitsiklis se placent dans le contexte suivant :

Hypothèse 4.1 *Chaque donnée communiquée sur un lien est correctement reçue avec un retard fini qui est cependant non spécifié.*

La solution apportée au problème de la terminaison des algorithmes itératifs asynchrones est basée sur la décomposition du problème en deux parties. Premièrement, l'algorithme itératif asynchrone est modifié de telle sorte qu'il se termine en temps fini et converge vers un point fixe suffisamment proche de la solution du problème. Deuxièmement, une procédure de détection de la terminaison relevant de l'informatique est appliquée.

Bertsekas et Tsitsiklis ont proposé de modifier l'algorithme itératif asynchrone de la manière suivante. Si la réactualisation d'une composante du vecteur itéré n'altère pas sa valeur de manière significative alors la valeur de la composante n'est pas modifiée et n'est pas communiquée aux autres processeurs. La terminaison de l'algorithme itératif asynchrone modifié intervient lorsqu'une réactualisation ne modifie pas la valeur des composantes du vecteur itéré quel que soit le processeur (c'est à dire lorsque toutes les conditions de terminaison locales sont satisfaites) et qu'aucun message n'est en transit dans le réseau de communication.

Plusieurs procédures peuvent être utilisées afin de détecter la terminaison de l'algorithme itératif asynchrone modifié. On peut citer par exemple la procédure de Dijkstra et Scholten (cf. [DS80] et [BT89]) qui est basée sur l'acquittement de tous les messages et la génération d'un graphe d'activité. Le modèle de l'algorithme de détection de la terminaison est donné par la machine à états finis de la figure 4.1.

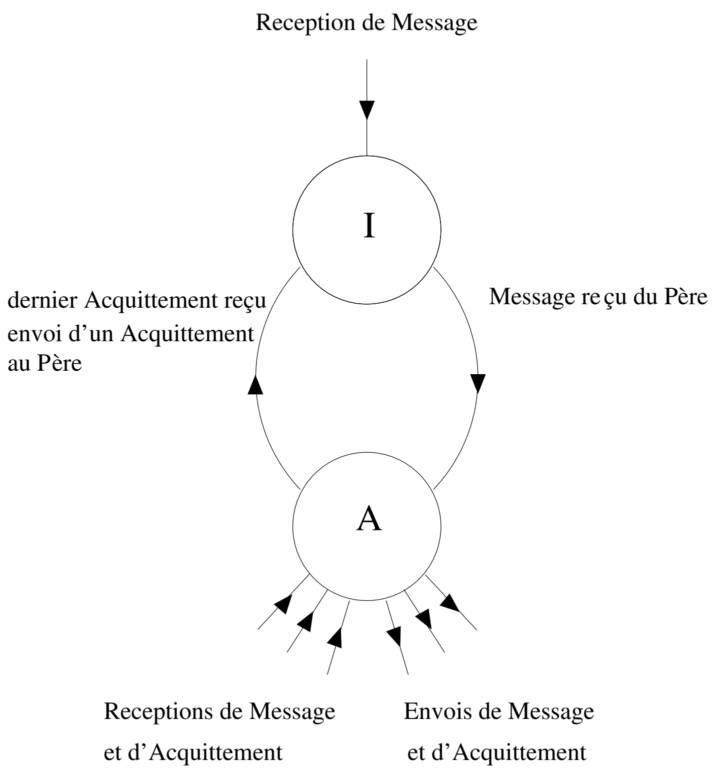


FIG. 4.1 – *Modèle d'un processeur pour la méthode de terminaison avec acquittement*

Les processeurs communiquent deux types d'information :
- la nouvelle valeur des composantes du vecteur itéré,
- l'acquittement des messages contenant une nouvelle valeur.

Définition 4.1 : Etat I. *Dans l'état inactif, un processeur ne calcule pas et ne transmet ni message ni acquittement.*

Définition 4.2 : Transition Tia. *Un processeur inactif P, change d'état dès la reception d'un message contenant la nouvelle valeur d'un sous-ensemble de composantes du vecteur itéré provenant d'un autre processeur qui est alors appelé : le père de P. Le message qui a* *activé P, qui a un rôle particulièrement important jusqu'à la prochaine phase d'inactivité, est appelé : message critique.*

Définition 4.3 : Etat A. *Dans l'état actif, un processeur réactualise les composantes du vecteur itéré qui lui sont affectées, transmet les valeurs résultant des réactualisations aux autres processeurs, et envoie systématiquement un acquittement pour tout message reçu à l'exclusion du message critique qui reçoit un traitement spécial.*

Définition 4.4 : Transition Tai. *Un processeur actif P, change d'état lorsque sa condition de terminaison locale est satisfaite, des acquittements ont été envoyés pour tous les messages reçus, à l'exception du message critique, et un acquittement a été reçu pour tous les messages envoyés. Lorsque la transition est tirée, le processeur P transmet à son père l'acquittement du message critique.*

Initialement, un seul processeur est actif. Ce processeur est appelé processeur racine et noté R. 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. Un exemple d'évolution du graphe d'activité est montré sur la figure 4.2, où nous avons représenté uniquement les processeurs actifs.

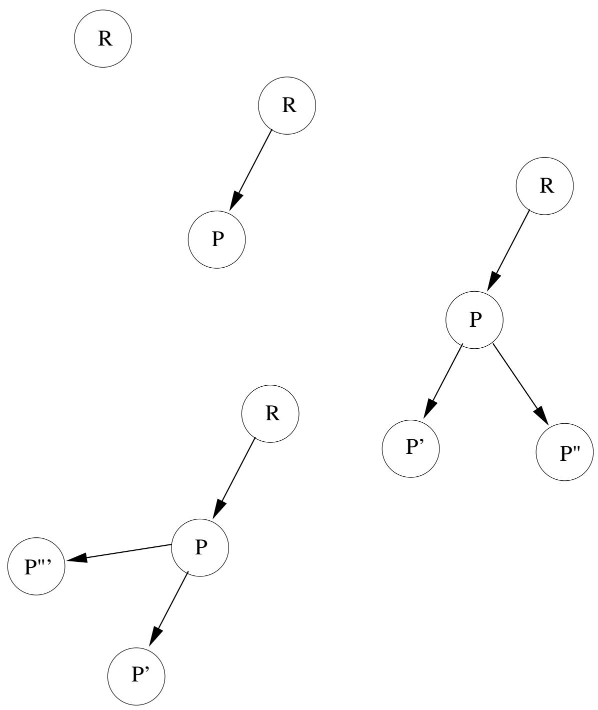


FIG. 4.2 – *Evolution du graphe d'activité*

Remarque 4.1 : *Le modèle des algorithmes itératifs asynchrones modifiés est différent de celui présenté au paragraphe 2.3.4. En effet, contrairement aux algorithmes itératifs asynchrones pour lesquels les valeurs des composantes du vecteur itéré peuvent changer à chaque itération, l'algorithme asynchrone modifié est tel que la valeur d'une ou de plusieurs composantes du vecteur itéré devient fixe après un temps fini* δ. *Bertsekas et Tsitsiklis ont suggéré d'introduire une nouvelle application de point fixe. Soit l'application de point fixe* F *et le paramètre* ϵ, *on définit une nouvelle application de point fixe* $G : X \to X$ *satisfaisant pour tout* $i \in \{1, ..., n\}$ *:*

$$G_i(x) = \begin{cases} F_i(x) \text{ si } \|F_i(x) - x_i\| \geq \epsilon, \\ = x_i, \text{ sinon}, \end{cases} \tag{4.1}$$

où $\|.\|$ *est une norme appropriée. On suppose donc que le nouveau problème de point fixe est :*

$$x = G(x). \tag{4.2}$$

Par conséquent, la convergence de l'algorithme asynchrone modifié ne découle pas directement du Théorème de Convergence Asynchrone de Bertsekas. Afin d'illustrer ce phénomène, Bertsekas et Tstsiklis ont donné dans [BT91] l'exemple suivant :
Soit ϵ *une constante positive et* $F : R^2 \to R^2$, *une application telle que*

$$F_1(x) = \begin{cases} -x_1, \text{ si } x_2 \geq \frac{\epsilon}{2}, \\ = 0, \text{ si } x_2 < \frac{\epsilon}{2}, \end{cases} \tag{4.3}$$

$$F_2(x) = \frac{x_2}{2}. \tag{4.4}$$

Tout algorithme asynchrone converge vers

$$x^* = (0, 0). \tag{4.5}$$

Toutefois si l'algorithme asynchrone modifié est initialisé avec

$$x_2 \in [\frac{\epsilon}{2}, 2.\epsilon[, \tag{4.6}$$

alors la valeur de x_2 *ne change pas et le processeur qui réactualise* x_1 *effectue de manière répétitive :*

$$x_1 = -x_1. \tag{4.7}$$

Il n'y a donc pas convergence.

Remarque 4.2 : *Bertsekas et Tsitsiklis ont montré dans [BT91] que parmi toutes les classes de méthodes pour lesquelles les conditions du Théorème de Convergence Asynchrone de Bertsekas sont satisfaites on peut identifier deux grandes sous-classes pour lesquelles l'algorithme itératif asynchrone modifié converge en temps fini :*

1) Le cas où l'algorithme asynchrone est associé à une application de point fixe F *contractante en norme scalaire du type maximum pondérée :*

$$\|F(x) - F(x')\|_\infty^w \leq \nu \|x - x'\|_\infty^w, \tag{4.8}$$

où $\nu \in [0, 1\}$ *et*

$$\| x \|_\infty^w = \max_i \frac{| x_i |}{w_i}.$$

2) Certains cas où les algorithmes asynchrones sont associés à une application de point fixe F *monotone croissante.*

La méthode de Bertsekas et Tsitsiklis est une des rares méthodes de la littérature pour laquelle il existe une preuve formelle de validité. Toutefois cette méthode présente des inconvénients. Elle requiert tout d'abord l'utilisation d'un protocole complexe ainsi que deux fois plus de communications qu'un algorithme itératif asynchrone classique. De plus des conditions plus restrictives que les conditions du Théorème de convergence asynchrone de Bertsekas doivent être satisfaites afin de garantir la convergence de l'algorithme asynchrone modifié.

Pour terminer ce paragraphe, notons que Bertsekas et Tsitsiklis ont suggéré dans [BT91] d'utiliser 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 des liens et des processeurs lorsque les messages de marquage sont délivrés. Notons que les états enregistrés dans un cliché ne correspondent pas nécessairement au véritable état global du système à un instant donné. Toutefois l'information contenue dans un cliché est suffisante pour détecter certaines propriétés de l'etat global du système et en particulier la terminaison.

4.3 Une variante de la méthode de Bertsekas et Tsitsiklis

Dans ce paragraphe nous présentons une variante de la méthode de Bertsekas et Tsitsiklis (cf. paragraphe précédent ainsi que : [BT89] et [BT91]). Nous avons proposé cette version afin de diminuer le nombre de messages échangés. De manière plus précise, la méthode que nous avons développée ne nécessite aucun acquittement systématique contrairement à celle de Bertsekas et Tsitsiklis.

Tout comme dans le cas de la méthode de Bertsekas et Tsitsiklis, les algorithmes itératifs asynchrones sont modifiés afin de converger vers une solution en un temps fini. Notre méthode de terminaison repose sur l'hypothèse suivante concernant le temps de transmission des messages :

Hypothèse 4.2 : *Le temps de transmission* t_t *des messages entre toute paire de processeurs adjacents est borné par une constante positive* δ. *On a :*

$$t_t < \delta.$$

De plus les messages entre toute paire de processeurs adjacents sont reçus suivant leur ordre d'émission.

Remarque 4.3 : *L'hypothèse 4.2 est légèrement plus restrictive que l'hypothèse 4.1 ; elle ne réduit pas la portée de la méthode que nous proposons. En effet l'hypothèse 4.2 ne borne que les retards dus aux communications. Le comportement général des algorithmes itératifs* *peut toujours être asynchrone puisque les deux sources d'asynchronisme dans le modèle mathématiques étudié sont les retards dus aux communications et la différence de charge entre processeurs comme cela a été présenté au chapitre 2.*

Le fonctionnement des processeurs mettant en œuvre l'algorithme itératif asynchrone modifié est représenté par la machine à états finis de la figure 4.3 pour laquelle chaque processeur peut avoir trois états : actif (A), inactif (I), et terminal (T).

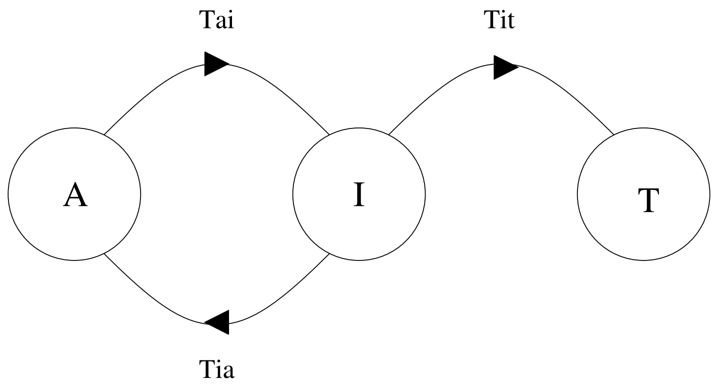


FIG. 4.3 – *Modèle d'un processeur mettant en œuvre la méthode de terminaison sans acquittement*

Initialement, un seul processeur est actif. Ce processeur est appelé la racine et est noté R. Tous les autre processeurs sont inactifs.

On notera par la suite T, un arbre de racine R recouvrant l'ensemble des processeurs.

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,
- les messages de terminaison.

Les données suivantes sont rangées dans chaque processeur P :
- les valeurs des itérés.
- l'identité du processeur qui a activé P (qui est aussi appelé : le père de P),
- la liste des processeurs activés par P (qui sont aussi appelés : les fils de P).

Définition 4.5 : Etat A. *Dans l'état actif, 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'.*

Remarque 4.4 : *Le test de terminaison local peut être du type proposé par Bertsekas et Tsitsiklis dans l'équation (4.1) ; il peut être aussi basé sur un calcul de résidu du type :*

$$\left|\sum_{l=1}^{\alpha} A_{il}.x_l(\tau_l^i(t)) - b_i\right| \leq \epsilon, \tag{4.9}$$

où ϵ est une constante positive.

Définition 4.6 : Etat I. *Dans l'état d'inactivité, un processeur est en attente de message.*

Définition 4.7 : Etat T. *Dans l'état terminal, un processeur ne fait plus rien.*

Définition 4.8 : Transition Tia. *Un processeur inactif devient actif lorsqu'il reçoit la nouvelle valeur d'un sous-ensemble de composantes du vecteur itéré. Lorsque la transition Tia est tirée, le processeur envoie un message d'activité au processeur qui lui a envoyé cette nouvelle valeur et qui devient alors son père, l'identité du père étant mémorisée par le nouveau processeur actif.*

Définition 4.9 : Transition Tai *Un processeur actif devient inactif 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.*

Définition 4.10 : Transition Tit. *Le processeur racine R passe immédiatement de l'état inactif à l'état terminal. 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.*

Remarque 4.5 : *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.*

Remarque 4.6 : *De manière claire, 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 échangés entre processeurs et donc au nombre de messages d'acquittement de l'algorithme de Bertsekas et Tsitsiklis.*

Remarque 4.7 : *Comme dans le cas de l'algorithme de terminaison proposé par Bertsekas et Tsitsiklis au paragraphe 4.2, l'algorithme asynchrone modifié présenté dans ce paragraphe est tel que certaines composantes du vecteur itéré deviennent constantes en un temps fini. Par conséquent la convergence de l'algorithme itératif asynchrone modifié ne découle pas directement du Thèorème de Convergence Asynchrone de Bertsekas. Toutefois la remarque 4.2 du paragraphe 4.2 s'applique toujours.*

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

4.4 Méthode de Savari et Bertsekas

Une autre méthode de terminaison particulièrement intéressante a été proposée par Savari et Bertsekas dans [SB96]. Dans cette méthode, les itérations asynchrones sont peu modifiées : le résultat de chaque nouvelle réactualisation d'une composante du vecteur itéré est pris en compte et communiqué aux autres processeurs s'il est différent de la dernière valeur de la composante. De plus, des requêtes sont envoyées à tous les processeurs du système chaque fois qu'une condition de terminaison locale n'est pas satisfaite. Un processeur effectue les calculs et transmet messages et requêtes aux autres processeurs aussi longtemps que sa condition de terminaison locale n'est pas satisfaite ou qu'il reçoit des requêtes provenant d'autres processeurs. La terminaison intervient lorsque tous les processeurs ont satisfait leur condition locale de terminaison et qu'aucun message relatif à une requête ou au résultat d'une réactualisation n'est en transit dans le système.

La terminaison est détectée en utilisant un protocole standard (cf. [DS80] et [CL85]).

Savari et Betsekas ont donné une preuve formelle de validité de cet algorithme de terminaison. Le principal avantage de cette méthode est qu'elle peut être appliquée avec succès à une plus large classe d'algorithmes itératifs que la méthode de Bertsekas et Tsitsiklis. Son principal défaut est de nécessiter un très grand nombre de communication de type requête.

4.5 Méthode des ensembles de niveau

Dans ce paragraphe, nous proposons une approche différente de celles présentées précédemment. Elle est basée sur l'utilisation du Théorème de Convergence Asynchrone de Bertsekas (cf. [BT89] et chapitre 2. Plus précisément, cette méthode repose sur l'utilisation de la suite d'ensemble de niveaux $\{X(j)\}$ que nous avons présentée au paragraphe 3.2.3. Le principe de cette méthode consiste à terminer l'algorithme itératif asynchrone lorsque le vecteur itéré pénètre dans l'ensemble de niveau $X(\hat{q})$, où $\hat{q}$ est un entier naturel fixé a priori (cf. [EB96c]) et aucun message ne transite dans le réseau. L'algorithme itératif asynchrone est peu modifié.

Une procédure de calcul très simple relative aux ensembles de niveau est ajoutée. La définition de l'algorithme asynchrone modifié est donnée ci-dessous.

Définition 4.11 : *Chaque processeur mettant en œuvre l'algorithme itératif asynchrone modifié calcule $x_i(t+1)$ pour tout $t \in T^i$ suivant le modèle (2.5). A chaque réactualisation, le processeur calcule aussi le numéro du plus faible sous-ensemble de niveau auquel appartient la nouvelle valeur de la composante du vecteur itéré $x_i(t+1)$; ce numéro est communiqué ainsi que la valeur réactualisée de la composante à tous les processeurs qui en ont besoin pour leurs calculs. Si ce numéro est supérieur ou égal à $\hat{q}$, alors la composante n'est plus réactualisée. Pour tout $i \in \{1, ..., n\}$ et $t \in T^i$, le numéro $q_i(t+1)$ du sous-ensemble de niveau est calculé suivant :*

$$q_i(0) = 0, q_i(t+1) = \min_{l \in N^i} q_l(\tau_l^i(t)) + 1,$$

où $q_l(\tau_l^i(t))$ est le numéro du sous-ensemble de niveau auquel appartient la valeur disponible de la l-ème composante du vecteur itéré : $x_l(\tau_l^i(t))$ et $N^i = \{l \in \{1, ..., n\} \mid F_i$ est une fonction de la composante $x_l\}$.

Remarque 4.8 : *Le calcul des numéros des sous-ensembles de niveau consomme très peu de temps, De plus, les communications additionnelles dues à cette méthode ne sont pas pénalisantes étant donné que l'entier correspondant au numéro du sous-ensemble est codé avec peu de bits et peut être accolé au message contenant la valeur réactualisée de la composante.*

Remarque 4.9 : *Le test de terminaison:*

$$x(k+1) \in X(\hat{q}),$$

est particulièrement intéressant dans le cas où F est une P-contraction (cf. Annexe A). L'erreur initiale satisfait alors :

$$\|x(0) - x^*\| \leq (I - P)^{-1} \|x(0) - F(x(0))\|,$$

où $\|.\|$ est une norme vectorielle sur R^n et x^* représente la solution du problème. L' erreur est alors au moins réduite par $P^{\hat{q}}$ lorsque la terminaison intervient.

Remarque 4.10 *La détection de la terminaison peut être faite à l'aide d'un algorithme de détection classique (cf [DS80] et [CL85]). De plus, cette méthode peut être étendue aux itérations asynchrones par bloc. Dans [EB96c] nous avons donné une preuve formelle de validité de cette méthode.*

4.6 Autres méthodes de terminaison

Il existe d'autres méthodes de terminaison : Savari et Bertsekas ont notamment proposé dans [SB96] divers schémas de terminaison supervisée.

Miellou propose dans [Mie75a] et [MCDB90] une méthode basée sur l'utilisation d'un algorithme secondaire ou de contrôle d'erreur qui est dérivé de l'algorithme de F. Robert et G. Schroeder (cf. [Rob69] et [RCM75]). Cet algorithme secondaire consomme moins de temps de calcul que l'algorithme asynchrone initial encore appelé algorithme principal, toutefois les suites $\mathcal{S}$ et $\mathcal{I}$ définies au paragraphe 2.3.1 doivent être nécessairement identiques pour l'algorithme principal et l'algorithme secondaire.

Nous n'oublierons pas de mentionner qu'une analyse fine de l'application considerée peut aussi fournir des éléments pour décider de la terminaison d'un algorithme itératif asynchrone. En effet, dans certains cas on peut déduire de la valeur de certaines variables une information sur l'état global du réseau. Un exemple de ce type d'analyse peut être trouvé dans [EBSMG96] ainsi qu'au paragraphe 6.1.4, où nous traitons des problèmes d'optimisation de type flot dans les réseaux. Dans ce cas, le déficit ou résidu à un sommet destination du réseau permet de déduire la somme des valeurs absolues des déficits en chaque sommet du réseau ce qui fournit donc une information globale concernant la terminaison de l'algorithme.

4.7 Conclusion

Le problème de la terminaison des itérations asynchrones est certainement un des problèmes les plus complexes qui soient posés par ce type d'algorithme pour lequel chaque processeur va à son propre rythme, éventuellement suivant sa propre horloge, et pour lequel il n'y a pas de phase de synchronisation des processeurs. Dans ce chapitre, nous avons présenté plusieurs solutions dont il existe une preuve formelle de validité. Nous avons proposé aussi deux méthodes originales pour la terminaison des itérations asynchrones. La première méthode est dérivée de la méthode de Bertsekas et Tsitsiklis et requiert moins de communications que la méthode proposée par ces deux auteurs, la seconde est basée sur l'utilisation du Théorème de Convergence Asynchrone de Bertsekas et présente un intérêt particulier dans un contexte d'applications P-contractantes.

De manière générale, les méthodes qui modifient les algorithmes asynchrones de façon importante comme la méthode de Bertsekas et Tsitsiklis s'appliquent à un sous-ensemble de la classe des itérations asynchrones car la convergence ne découle pas directement des résultats standards du fait des modifications introduites. A contrario les méthodes qui modifient peu l'algorithme asynchrone comme la méthode de Savary et Bertsekas ont un vaste champ d'application car les résultats standards de convergence s'appliquent, toutefois, le nombre de communications peut devenir alors très important. Les méthodes que nous avons proposées se situent entre ces deux extrêmes; elles tendent à s'appliquer à une large classe d'algorithmes asynchrones tout en limitant le nombre de communications.

Chapitre 5

Mise en œuvre

> Quand on veut créer sans avoir observé, on ne trouve que des rêveries et des absurdités.
>
> *J. Garat*

Ce chapitre traite de la mise en œuvre des itérations asynchrones et des itérations asynchrones avec communication flexible sur diverses architectures parallèles. Pour chaque machine, nous mentionnons brièvement comment nous avons mis en œuvre les schémas itératifs synchrones. Nous considérons successivement des machines à mémoire distribuée telles que le Tnode de Telmat (qui est une des premières machines parallèles que nous avons utilisée) et le supercalculateur Cray T3E de l'Idris ; une architecture à mémoire partagée de type "symmetric multiprocessor" qui est une Sun Sparcstation 20 quadriprocesseur, et une architecture de type réseau de stations de travail.

Nous nous sommes placés dans un contexte d'allocation statique des tâches de calcul aux processeurs qui convient bien aux différentes applications considérées : problèmes d'optimisation, résolution de systèmes Markoviens et de systèmes d'équations aux dérivées partielles discrétisés (cf. chapitre 6). Les algorithmes sont programmés suivant le modèle SPMD "Single Program Multiple Data". Nous distinguerons par la suite deux types de composantes du vecteur itéré.

Définition 5.1 *Toute composante du vecteur itéré qui doit être communiquée par un processeur à un autre processeur est appelée composante frontière ; les autres composantes étant appelées composantes internes.*

5.1 Mise en œuvre sur un multiprocesseur à mémoire distribuée Tnode

Dans ce paragraphe, nous présentons diverses mises en œuvre avec ou sans communication flexible sur le Tnode de Telmat. La machine utilisée consistait en un réseau de 16 à 32 transputers T800 possédant de la mémoire locale. Nous rappelons brièvement que le transputer T800 était un microprocesseur qui intégrait sur un même composant : un processeur, une unité de calcul en virgule flottante, 1 Moctet de mémoire rapide, et quatre liens de communication bidirectionnels. Le processeur, l'unité de calcul en virgule flottante, et la mémoire permettaient d'utiliser ce microprocesseur comme composant de base d'une architecture de calcul. Les liens de communication permettaient de connecter plusieurs transputers afin de créer une machine parallèle. Le T800 présentait à la fin des années 80 l'avantage d'avoir d'excellentes caractéristiques tant en ce qui concerne la puissance des processeurs : 10 MIPS, 2 Mflops, que les communications interprocesseurs : temps de latence très bref : 7 μs et bande passante de l'ordre de 2 Moctets/s. Plusieurs topologies de réseau pouvaient être programmées au moyen du circuit "crossbar" C004 de Inmos, à savoir : un "pipeline", un anneau, une grille, un cube... Nous nous sommes essentiellement intéressés au cas de la topologie "pipeline" qui était bien adaptée aux différents problèmes que nous avons traités : systèmes Markoviens etc. (cf. chapitre 6).

Nous décrivons tout d'abord très brièvement comment nous avons mis en œuvre des schémas de calcul synchrones sur ce type d'architecture afin de mieux situer les problèmes posés et les techniques utilisées dans le cas de la mise en œuvre asynchrone.

5.1.1 Mise en œuvre synchrone

Dans le cas d'une mise en œuvre synchrone, la réactualisation et l'échange de données sont effectués de manière séquentielle. Les processeurs communiquent la valeur des composantes frontières à la fin de chaque réactualisation. Les processeurs se synchronisent par échange de message, puisque le seul type de communication disponible sur les architectures à base de transputers programmées suivant le modèle CSP, "Communicating Sequential Processes" de Hoare (cf. [Hoa85] et [MT84]) est synchrone. Le mode de communication exploité par les transputers repose sur le concept de rendez-vous avec émission et réception bloquante des messages. Les communications entre transputers sont donc synchrones et ne font pas l'objet d'une "bufferisation" ou rangement dans une mémoire tampon.

5.1.2 Une première mise en œuvre asynchrone

Comme on l'a vu au paragraphe précédent, le seul type de communication disponible sur des architectures à base de transputers est le mode synchrone. Nous avons choisi la solution suivante afin de mettre en œuvre des schémas de calcul asynchrones (cf. [EB89b] et [EB93a]). Deux processus concurrents sont mis en œuvre sur chaque processeur (cf figure 5.1 et algorithme ci-après).

PAR $i = 1$ FOR α
    PRI PAR
        MEMOIRE
        CALCUL(i)

Un processus de calcul réactualise toutes les composantes du vecteur itéré qui lui sont affectées. Le processus de calcul envoie la valeur des composantes frontières aux processeurs qui les utilisent dans leurs calculs grâce aux cannaux de communication du transputer. Un processus mémoire encore appelé processus "buffer" reçoit et range en mémoire la valeur des composantes frontières envoyées par les autres processeurs. L'utilisation du processus

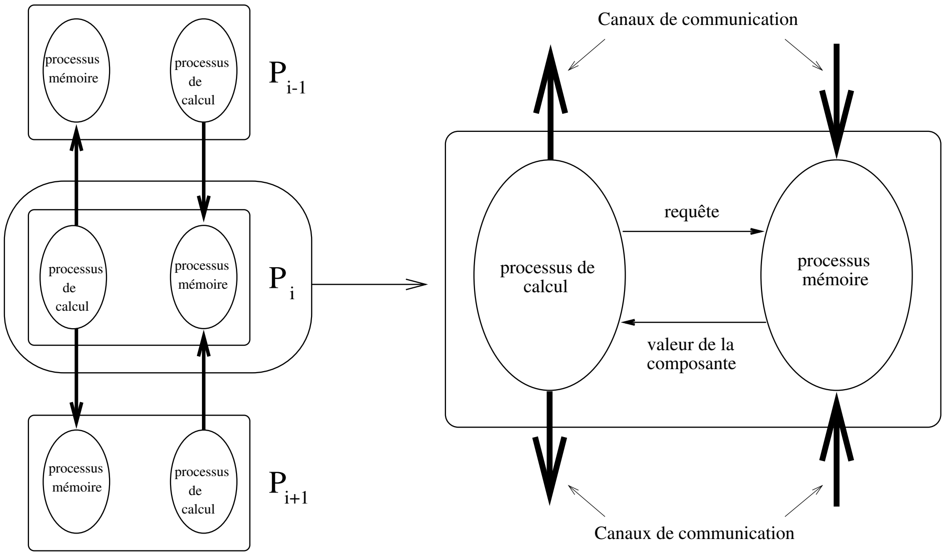


FIG. 5.1 – *Mise en œuvre des itérations asynchrones avec une mémoire*

mémoire permet de mettre en œuvre des communications asynchrones et de manière plus générale des algorithmes itératifs asynchrones. Le processus mémoire qui possède des processus élémentaires très rapides a un niveau de priorité plus élevé que le processus de calcul qui consomme beaucoup de temps. Ainsi la réception de données n'est pas retardée. Le processus mémoire est inactif lorsqu'il ne reçoit pas de message. Tout le temps cpu est alors alloué au processus de calcul car l'ordonnanceur du T800 est conçu de manière à ne pas allouer de temps cpu aux processus inactifs. Les processus calcul et mémoire communiquent par échange de message suivant le modèle CSP. Plus précisément, le processus de calcul émet une requête vers le processus mémoire afin d'acquérir la valeur des composante frontières. Le processus mémoire effectue la réception des différents messages : requêtes ou données et exécute des services dès la réception de ces messages : réponse aux requêtes, rangement en mémoire de la valeur des composantes frontières ; toutes ces tâches sont regroupées au sein d'un processus qui effectue un test conditionnel sur les événements relatifs aux messages disponibles grâce au constructeur Occam : ALT. On obtient ainsi une mise en œuvre très efficace des communications asynchrones. A notre connaissance nous avons été le premier auteur à proposer ce type de mise en œuvre pour des machines à base de transputers (cf. [EB89b] et aussi [EB93a]).

Remarque 5.1 *Conforti, Grandinetti, Musmanno, Cannataro, Spezzano, et Talia ont proposé en 1992 dans [CGM+92] une mise en œuvre de l'asynchronisme assez semblable à la notre car elle utilise des processus calcul et "buffer". Toutefois, les auteurs de cet article ne définissent aucun niveau de priorité en ce qui concerne les processus calcul et "buffer" ; de plus plusieurs processus "buffer" peuvent être mis en œuvre sur un même processeur ce qui alourdit la mise en œuvre. Un processus "buffer" assure toujours la transmission de la dernière valeur d'une composante frontière.*

Remarque 5.2 *Giraud, Spitéri, et Berger ont proposé dans [GSB89] et [GS89] (cf. aussi [Gir91]) différentes mises en œuvre des itérations asynchrones sur machine multi-transputers* *basées sur l'utilisation de processus de communication et de processus de calcul. A la différence de notre approche, la solution qu'ils proposent consiste à allouer seulement un processus par processeur et donc à définir des processeurs ayant uniquement des fonctions de communication. Giraud, Spitéri, et Berger ont proposé différentes topologies de réseau pour la mise en œuvre d'itérations asynchrones: soit en anneau double, où tous les processeurs associés aux processus de communication sont connectés en anneau, les processeurs associés aux processus de calcul étant uniquement connectés à un seul processeur de communication, soit en anneau simple où un processeur de communication s'intercale entre deux processeurs de calcul.*

5.1.3 Une seconde mise en œuvre asynchrone

Dans le cadre de la thèse de Didier Gazen (cf. [Gaz98] et [GEB95]), nous avons été amenés à proposer une seconde mise en œuvre qui déroge au règles d'Occam mais qui est légèrement plus efficace. Cette mise en œuvre conserve les principes de base de la mise en œuvre présentée au paragraphe précédent; toutefois, le processus de calcul et le processus mémoire ne communiquent plus par passage de message mais au moyen de variables partagées grâce à une des options du compilateur Occam (cf. figure 5.2). Il résulte de ceci que le processus de calcul n'a plus à envoyer de requête au processus mémoire afin que ce dernier lui retoune les valeurs des composantes frontières qu'il a reçues. Le processus mémoire est alors entièrement consacré à la réception et au rangement des valeurs des composantes frontières et le processus de calcul accède sans retard à la valeur des composantes frontières. Notons que les valeurs rangées dans les variables partagées correspondent uniquement à des composantes frontières et que le processus de calcul accède toujours à la dernière valeur disponible de chaque composante frontière car les anciennes valeurs sont écrasées au fur et à mesure que des valeurs plus récentes sont rangées dans les variables partagées.

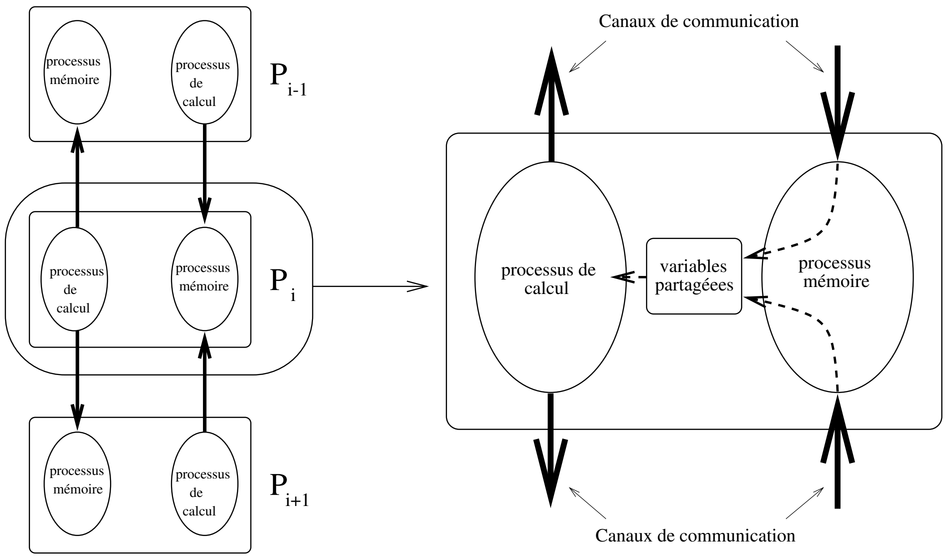


FIG. 5.2 – *Mise en œuvre des itérations asynchrones avec l'utilisation de variables partagées*

5.1.4 Une première mise en œuvre asynchrone avec communication flexible

Une première mise en œuvre simple des itérations asynchrones avec communication flexible consiste à modifier uniquement le processus de calcul afin d'envoyer des itérés partiels en supplément du résultat de chaque itération. Dans ce cas, la mise en œuvre est essentiellement identique à celle des figures 5.1 ou 5.2. Quand la réactualisation de chaque composante ou de chaque bloc de composantes est effectuée grâce à un processus itératif, les itéré partiels peuvent être envoyés tous les q pas de calcul intermédiaires, où q est un nombre fixé a priori, ou lorsqu'un degré de précision supplémentaire est atteint (cf. [EBSMG96]).

5.1.5 Une seconde mise en œuvre asynchrone avec communication flexible

Dans ce paragraphe, nous présentons une autre méthode permettant de mettre en œuvre les algorithmes itératifs asynchrones avec communication flexible sur un multiprocesseur MIMD à mémoire distribuée suivant le modèle SPMD. Cette méthode a été développée dans le cadre de la thèse de Didier Gazen afin de minimiser le nombre de communications entre processeurs (cf. Annexe C.2, [Gaz98], et [EBGMS96]) ; elle a été mise en œuvre sur la machine Tnode de Telmat.

Nous supposons que les données sont partitionnées en α sous-ensembles, où α correspond au nombre de processeurs utilisés. Chaque sous-ensemble est attribué de manière statique à un processeur qui est alors chargé de réactualiser un sous-ensemble de composantes du vecteur itéré. La technique employée repose sur l'émission de requêtes par les processeurs qui désirent connaître la valeur de composantes frontières réactualisées par d'autres processeurs et l'envoi de la valeur des composantes frontières en réponse à ces requêtes.

Principe de la méthode

La mise à jour de la valeur des différentes composantes affectées à un même processeur s'effectue séquentiellement. La procédure de mise à jour ne présente aucune difficulté majeure lorsqu'elle ne fait pas appel à la valeur d'une composante frontière d'un autre processeur. Nous détaillons par la suite la procédure de calcul des valeurs des composantes dans le cas où la connaissance de la valeur d'au moins une composante frontière est requise. Nous décrivons plus précisément la technique adoptée pour les échanges de données. Dans ce cas, la procédure itérative de calcul de la valeur d'une composante x_i commence après requête des valeurs des composantes frontières x_k mais sans attente de réponse à ces requêtes. Si une réponse survient immédiatement après une requête, elle est prise en compte au cours du processus de calcul de x_i. Si cette réponse survient plus tard, le calcul s'effectue avec la valeur de x_k présente au moment de l'envoi de la requête.

Pour réduire le nombre de communications (en particulier lorsque la charge des processeurs est inégale) nous avons défini la politique suivante d'échange de message.

- un processeur P_1 n'envoie une requête relative à une composante frontière x_k que s'il a reçu une réponse à la requête précédente concernant x_k.
- un processeur P_2 se doit de répondre à une requête de P_1 relative à une composante frontière x_k :
  - soit immédiatement si la composante x_k a évolué depuis le précédent envoi de x_k

à P_1,
– soit dès que la composante x_k est réactualisée.

La figure 5.3 illustre par un diagramme temporel le principe de l'envoi et de la prise en compte des requêtes. La composante x_k est une composante frontière du processeur P_2, la composante x_i quant à elle est réctualisée par le processeur P_1. Nous avons représenté sur la figure 5.3 :

- les périodes de mise à jour de x_i et x_k,
- les requêtes issues de P_1 concernant la composante x_k,
- les envois de la valeur de la composante x_k par P_2.

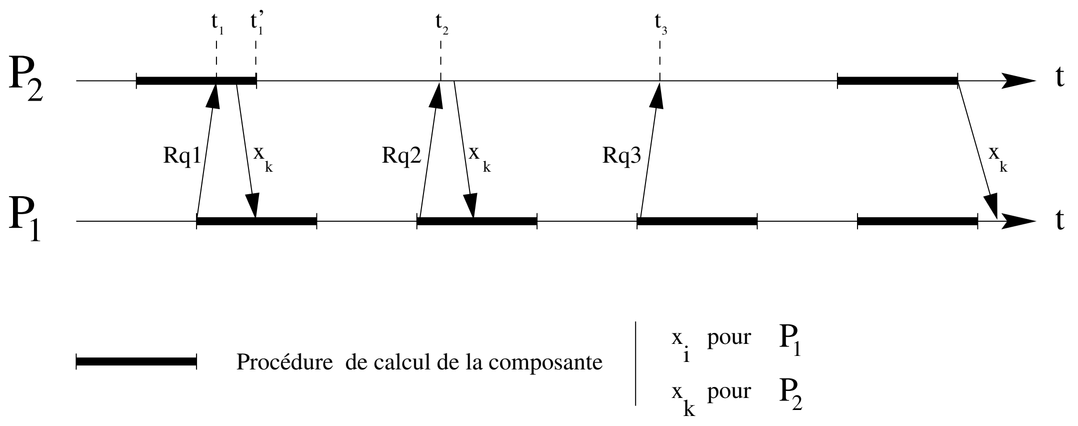


FIG. 5.3 – *Illustration de l'algorithme avec requêtes*

Nous pouvons vérifier que les réponses aux requêtes 1 et 2 sont quasi-immédiates car x_k est en cours de réactualisation à l'instant t_1 et $x_k(t_2) = x_k(t_1') \neq x_k(t_1)$. De plus, on remarque que la valeur de la composante $x_k(t_1)$ qui est prise en compte durant la mise à jour de x_i, est un itéré partiel : ce type de fonctionnement caractérise les itérations asynchrones avec communication flexible. Enfin, on constate que la requête 3 n'obtient pas de réponse immédiate puisque $x_k(t_3) = x_k(t_2)$, il n'y aura donc pas d'autres requêtes pour x_k tant que le processeur P_1 n'aura pas reçu de réponse à la requête 3.

Mise en œuvre des itérations asynchrones avec communication flexible

L'algorithme a été programmé en langage 3LC sur la machine TNode. Dans cette étude nous avons considéré un réseau de processeurs organisé en *pipeline* avec liens bidirectionnels. On suppose par simplicité que tout routage d'information est inutile.

La mise en œuvre de la méthode décrite au sous-paragraphe précédent met en jeu plusieurs processus au sein d'un même processeur :

- d'une part, un processus de *Calcul*, noté C, permettant de réactualiser un sous-ensemble de composantes,

– d'autre part, plusieurs processus de communication :

  – un processus R de *Réception* de requête ou de la valeur d'une composante frontière émise par un autre processeur,
  – deux processus E_R d'*Envoi immédiat* de la valeur d'une composante frontière activés par le processus de Réception,
  – deux processus E_C d'*Envoi différé* de la valeur d'une composante frontière ou d'*Envoi* de requête, activés par le processus de Calcul.

On notera que les processeurs situés à l'extrémité du pipeline ne dialoguent qu'avec un seul processeur et ne disposent donc que d'une copie des processus E_R et E_C.

Tous les processus de communication sont des processus de priorité *haute* alors que le processus de calcul est l'unique processus de priorité *basse*. Ainsi, les processus de communication agissent comme des interruptions sur le processus de calcul. Ils s'exécutent un à un suivant leur ordre d'activation et de manière ininterrompue jusqu'à une attente de communication. Ces interruptions, bien que nombreuses, consomment peu de temps CPU et en l'absence de toute communication, le temps CPU est entièrement dédié au processus de calcul.

Les différents processus ainsi que leurs liaisons d'activation sont representés sur la figure 5.4. Le processus de calcul C met à jour séquentiellement la valeur des composantes du vecteur itéré qui lui ont été affectées. Lors de la mise à jour des composantes, C peut avoir besoin de prendre connaissance de la valeur de composantes frontières calculées par d'autres processeurs ou d'envoyer la valeur d'une composante frontière en fin de réactualisation. Il active alors le processus E_C par émission d'un message sur le canal unidirectionnel L_i.

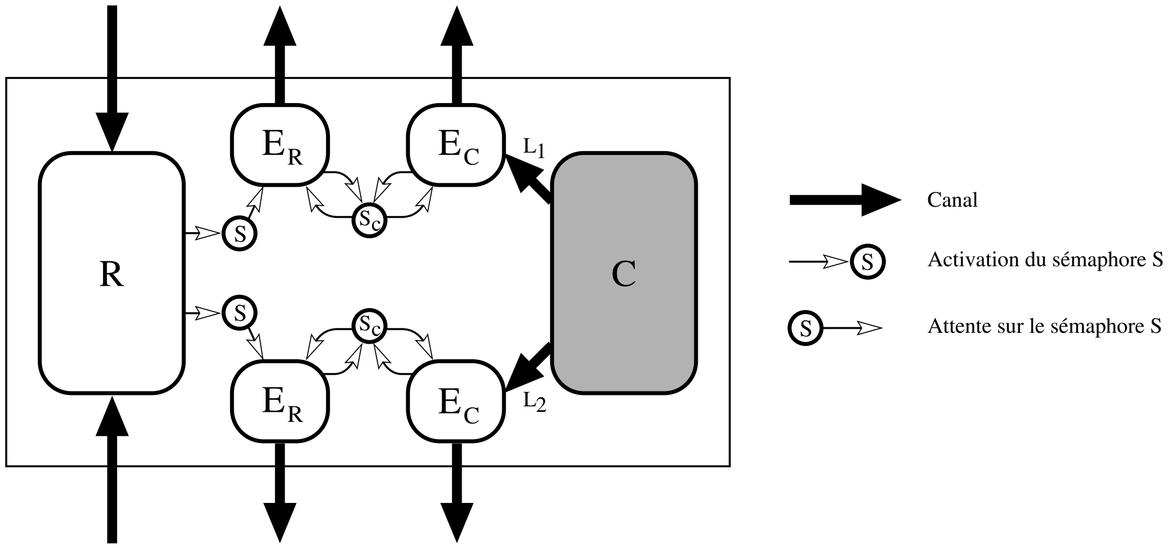


FIG. 5.4 – *Les processus et leurs intéractions*

Le processus E_C de priorité haute interrompt immédiatement C et transmet une requête ou une valeur de composante frontière au processus de réception d'un des processeurs voisins. Si le processus de réception du processeur voisin n'est pas en état de recevoir (dans le cas où un autre processus de priorité haute est en train de s'exécuter, par exemple), le processus E_C est désordonnancé et C poursuit son exécution. Dès qu'il a émis vers le processeur voisin, E_C se replace en attente de communication sur le canal L_i.

Le processus R, quant à lui, est en attente de réception sur les deux canaux physiques qui le relient aux processus d'envoi des deux processeurs voisins. Il passe dans l'état actif dès qu'un message arrive sur l'un des canaux[1]. Au cours de son exécution, R réagit en fonction du type de message émis par un processeur voisin :

- si le message correspond à la valeur d'une composante, il range cette valeur dans la zone mémoire relative aux composantes frontières,

- si le message est une requête relative à une composante frontière et que la valeur de la composante n'a pas évolué depuis le précédent envoi, R mémorise la requête. Si la valeur de la composante a évolué, R active le processus E_R par l'envoi d'un signal[2] sur le sémaphore S.

Le processus E_R attend le signal[3] de R pour transmettre la valeur de la composante frontière au processus de réception du processeur voisin concerné. L'utilisation d'un sémaphore plutôt que d'un canal entre les processus R et E_R qui ont le même niveau de priorité permet d'éviter les interblocages qui se seraient produits si le processus R avait fonctionné à la fois en processus récepteur et émetteur. L'activation du sémaphore par R est en effet une opération non bloquante.

Remarque 5.3 *Les processus R et E_R ayant la même priorité (haute), E_R ne s'exécute pas à l'instant où R l'active, mais uniquement lorsque R se retrouve bloqué en attente de réception de messages. La durée écoulée entre l'instant d'activation par R et l'instant d'exécution de E_R est indéterminée alors que ces deux instants se confondent pour E_C processus de priorité haute activé par C processus de priorité basse.*

Remarque 5.4 *Les processus E_C et E_R partagent le même canal physique pour accéder au processus de réception d'un processeur voisin. Les deux processus étant de priorité identique, le mécanisme de partage de la ressource canal d'émission est réalisée à l'aide d'un sémaphore (S_c).*

Remarque 5.5 *La mise en œuvre de l'algorithme nécessite de mémoriser le fait qu'une composante a été mise à jour, ou qu'une requête a été envoyée ou reçue au moyen de variables booléennes que se partagent les différents processus.*

1. R ne prend en compte qu'une communication à la fois - attente sur l'instruction 3LC alt_wait -.
2. instruction 3LC : sema_signal.
3. instruction 3LC : sema_wait.

5.2 Mise en œuvre sur Cray T3E

Les itérations asynchrones ont été mises en œuvre sur le supercalculateur Cray T3E de l'Idris dans le cadre du D.E.A. et de la thèse de notre nouvel étudiant M. Jarraya (cf. [Jar97] et [EBGJ+98]). Cette architecture parallèle à mémoire distribuée est constituée par un ensemble de 256 processeurs Digital α EV5 ayant chacun une puissance de crête de 600 Mflops (l'horloge centrale a une fréquence de 300 MHz) connectés par un tore tridimensionnel. Le temps de latence d'une communication est très faible : 1 μs et la bande passante du réseau est égale à 480 Moctets/s. Le Cray T3E peut avoir des configurations massivement parallèles comprenant jusqu'à 2048 processeurs. Pour notre part, le nombre maximum de processeurs utilisés lors des mises en œuvre n'a pas dépassé 64 processeurs, essentiellement en raison des contraintes d'exploitation de la machine. Divers environnements sont disponibles tels que M.P.I. "Message Passing Interface" qui permet d'utiliser un réseau de processeurs comme une ressource unique de calcul ou SHMEM qui est un logiciel mis au point par Cray afin de permettre les communications entre processeurs et d'utiliser notamment l'ensemble de la mémoire de la machine comme s'il s'agissait d'une mémoire partagée, ceci afin de faciliter les échanges de données.

5.2.1 Mise en œuvre parallèle synchrone

Nous avons choisi de mettre en œuvre simplement les schémas itératifs synchrones en utilisant la bibliothèque M.P.I. Chaque processeur réactualise un sous-ensemble de composantes, envoie aux autres processeurs la valeur des composantes frontières qui seront utilisées dans les calculs à l'itération suivante, et se met en attente des valeurs des composantes frontières qui sont réactualisées lors de la même itération par les autres processeurs et qui seront utilisées à l'itération suivante. L'envoi et la reception des données se font respectivement à l'aide des fonctions MPI_Isend() et MPI_Recv() de la bibliothèque MPI qui ont pour argument l'adresse des données à envoyer ainsi que leur taille. Les valeurs de toutes les composantes frontières peuvent être envoyées en une seule opération en définissant un type dérivé créé dynamiquement[4] (cf. [Jar97]). Les fonctions MPI_Isend() et MPI_Recv() établissent un protocole de communication entre le processeur source et le processeur destination. La fonction non bloquante MPI_Isend() permet un retour immédiat et évite le blocage des processeurs lors de l'envoi simultané de messages. La synchronisation est garantie par la fonction bloquante MPI_Recv() qui ne peut se terminer que lorsqu'un message a été effectivement reçu.

5.2.2 Mise en œuvre parallèle asynchrone

Dans le cas asynchrone, chaque processeur itère à son propre rythme sans attendre la valeur des composantes frontières provenant des autres processeurs. Dans le cadre du travail de thèse de Mohamed Jarraya, une mise en œuvre des itérations asynchrones a été réalisée en utilisant les fonctions non bloquantes MPI_Isend() et MPI_Irecv() de la bibliothèque MPI. Dans [Jar97], une autre politique a été mise en œuvre afin d'échanger les données entre processeurs ; elle consiste à diffuser après chaque réactualisation la nouvelle valeur des composantes frontières à tous les processeurs qui doivent l'utiliser lors de leurs calculs. Cette politique utilise la fonction put() de la bibliothèque SHMEM de CRAY qui a pour argument l'adresse de la donnée sur

4. fonction MPI-type-hindexed()

le processeur cible, l'adresse de la composante frontière sur le processeur source, la longueur de la donnée à copier, et l'identité du processeur cible. Plus précisément, la fonction put() effectue une copie de la mémoire du processeur source vers la mémoire du processeur cible. La fonction put() est non blocante et son éxécution est relativement rapide, le temps de latence de cette fonction est de l'ordre de 1,5 μs. Nous rappelons que l'un des grands avantages de la bibliothèque SHMEM est de permettre d'accéder aux différentes données de l'architecture parallèle comme si elles appartenaient à une mémoire partagée.

5.2.3 Mise en œuvre asynchrone avec communication flexible

Nous proposons deux manières de mettre en œuvre les communications flexibles : soit en permettant aux différents processeurs d'envoyer une valeur d'itéré partiel aux autres processeurs après un certain nombre de pas de calcul en utilisant la fonction put() (dans ce cas nous noterons S le sous-ensemble des numéros de pas intermédiaires auxquels les itérés partiels sont communiqués) ; soit en permettant aux processeurs de lire la valeur courante des composantes frontières réactualisées par les autres processeurs en utilisant la fonction get(), cette valeur n'étant pas forcément labellée par un numero d'itération. Les algorithmes suivants correspondent aux deux manières de réactualiser une composante x_i à l'itération j, NProc étant le nombre de processeurs cibles et P_l le numéro du processeur considéré. Afin de simplifier la présentation, les seuls arguments représentés pour les fonctions put() et get() sont respectivement la valeur de la composante frontière, l'identité du processeur cible, et l'identité du processeur source.

$q \leftarrow 0$
$x_i^q \leftarrow x_i(j-1)$
TANTQUE *résidu* $> \epsilon'$
  $q \leftarrow q+1$
  calcul de x_i^q
  SI $q \in S$ ALORS
    FAIRE $l = 1$ POUR NProc
      put(x_i^q,P_l)
    FINFAIRE
  FINSI
FINTANTQUE
$x_i(j) \leftarrow x_i^q$
FAIRE $l = 1$ POUR NProc
  put($x_i(j)$,P_l)
FINFAIRE

mise en œuvre avec la fonction put()

$q \leftarrow 0$
$x_i^q \leftarrow x_i(j-1)$
FAIRE $l = 1$ POUR NProc
  get(P_l)
FINFAIRE
TANTQUE *résidu* $> \epsilon'$
  $q \leftarrow q+1$
  calcul de x_i^q
FINTANTQUE
$x_i(j) \leftarrow x_i^q$

mise en œuvre avec la fonction get()

Il est à noter que d'autres mises en œuvre des itérations asynchrones avec communication flexible ont été effectuées en utilisant les fonctions non bloquantes MPI_Isend() et MPI_Irecv() dans le cadre du travail de thèse de Mohamed Jarraya.

5.3 Mise en œuvre sur SMP

Dans le cadre de la thèse de Didier Gazen (cf. [Gaz98] et aussi [EBGJ+98]) nous avons mis en œuvre les itérations asynchrones sur une SUN Sparcstation 20 quadriprocesseur qui est une machine à mémoire partagée de type SMP, “Symmetric MultiProcessor” où multiprocesseur symétrique pour laquelle tous les processeurs voient les ressources systèmes de la même manière et qui est composée de quatre processeurs HyperSparc de fréquence 100 MHz disposant de mémoire cache interne de 8 Koctets et accédant à une mémoire partagée de 128 Moctets par un bus à une fréquence de 50 MHz. Notons l'existence de mémoire cache externe de 256 Koctets.

La mise en œuvre repose sur l'utilisation de processus “multithreadés” qui permettent d'exploiter efficacement le parallélisme de l'architecture. Rappelons qu'un processus multithreadé est composé de plusieurs “threads”, ou flots d'instructions, exécutant simultanément des portions de code sur différents processeurs. Les “threads”, ou fils, sont des sortes de processus légers qui consomment peu de ressources (ils possèdent leurs propres registres, compteur ordinal, et petite pile d'exécution), ils peuvent donc être créés et coexister en grand nombre. Les processus légers partagent un espace d'adressage ainsi que d'autres ressources et présentent des temps de commutation de contexte plus brefs que ceux des processus lourds supportés par le noyau du système; ils peuvent notamment partager un nœud de calcul de manière parallèle comme dans le cas d'un multiprocesseur symétrique ou un monoprocesseur en temps partagé.

Les processus légers ont été introduits en particulier dans les systèmes d'exploitation afin de permettre à un serveur d'un système réparti de répondre efficacement à des requêtes simultanées. De manière plus précise, lorsqu'une requête est bloquée sur une entrée ou une sortie, une autre requête peut être exécutée en maintenant ainsi une activité soutenue du serveur. Dans le domaine des systèmes répartis ils offrent donc une alternative particulièrement attractive aux processus “lourds” de type Unix qui consomment beaucoup de ressources système.

Dans le domaine de l'informatique du parallélisme ils sont une solution attractive pour aborder le parallélisme massif. Dans ce cas, la mise en œuvre repose sur l'utilisation de processus “multithreadés” qui permettent d'exploiter efficacement le parallélisme de l'architecture. En supplément de la souplesse offerte à l'utilisateur, l'emploi d'une bibliothèque de threads accroît l'efficacité et présente l'avantage de la portabilité grâce à des standards comme POSIX, “Portable Operating System Interface eXchange” disponible sous Solaris 2.

Nous avons adopté un modèle simple de programmation de type SPMD où l'on crée autant de threads qu'il y a de processeurs. Chaque thread met donc en œuvre la même fonction sur des données et des processeurs différents. Les échanges de données se font par accès direct à la mémoire partagée.

5.3.1 Mise en œuvre parallèle synchrone

Nous rappelons que dans le cas d'une mise en œuvre synchrone, la réactualisation d'une composante à l'itération j ne peut commencer que si la réactualisation des composantes à l'itération $j-1$ est terminée. Les données sont structurées de la manière suivante : on définit deux copies des valeurs des composantes du vecteur itéré qui sont accessibles par tous les threads. La première copie contient l'ancienne valeur des composantes du vecteur itéré et n'est accédée qu'en lecture afin de construire le nouvel itéré. Dans la seconde copie, accessible en écriture, les threads rangent le résultat de la phase de réactualisation. Cette structure est introduite afin d'assurer un fonctionnement synchrone et d'éviter les problèmes d'accès concurrents à la mémoire. La synchronisation des threads est réalisée au moyen d'une barrière de synchronisation développée en utilisant les deux primitives de base offertes par les threads POSIX qui sont : le verrou d'exclusion mutuelle et la variable de condition. La barrière de synchronisation utilise un compteur d'appel ; initialement ce compteur est égal à zéro. Chaque appel à la barrière provient d'un thread différent. L'accès au compteur est protégé par le verrou d'exclusion mutuelle et le blocage des threads est réalisé par attente sur variable de condition. La condition est satisfaite lorsque le dernier thread fait appel à la barrière. Le dernier thread qui n'est donc pas bloqué lance alors une fonction qui permute le nom des anciennes et nouvelles copies du vecteur itéré. Il en résulte que l'ancienne copie contient alors le résultat de la dernière réactualisation. Puis le dernier thread débloque les autres threads en signalant la variable condition, ce signal a pour effet d'annuler l'attente des threads sur la variable de condition.

5.3.2 Mise en œuvre asynchrone avec communication flexible

Les algorithmes itératifs asynchrones avec communication flexible sont les seuls schémas de calcul asynchrones a avoir été mis en œuvre sur SMP étant donné qu'ils s'adaptent particulièrement bien et de manière très naturelle à ce type d'architecture. En effet, l'introduction de variables locales nécessaire à la mise en œuvre d'algorithmes itératifs asynchrones standards serait dans ce cas artificielle.

Une seule copie du vecteur itéré est suffisante pour mettre en œuvre les itérations asynchrones avec communication flexible. Cette copie est accessible en lecture et en écriture par tous les threads. La bibliothèque de threads POSIX n'est alors utilisée que pour la création des différents threads qui peuvent être ainsi affectés aux différents processeurs. Notons cependant qu'il est difficile de savoir si un thread donné a été ordonnancé sur le même processeur tout au long d'une application.

Pour une mise en œuvre sur la machine à mémoire partagée IBM 3090-600 on pourra se reporter notamment à [MEBS98] et [SMEB95].

5.4 Mise en œuvre sur un réseau de stations de travail

Dans le cadre du travail de thèse de Jarraya (cf. [JEBG98]), nous avons aussi mis en œuvre les algorithmes asynchrones sur un réseau de 8 stations de travail Sun qui est vu comme une machine parallèle virtuelle. Les stations utilisées sont des Sparcstations ayant une fréquence d'horloge de 110 MHz et une taille mémoire de 32 ou 64 Moctets. Elles sont interconnectées via un réseau Ethernet dont la latence (environ 1 ms) et la bande passante (10 Moctets/s) sont pénalisants.

Chaque instance du programme, nommée tâche de l'application, est exécutée sur une station de travail suivant le modèle SPMD. La communication entre les tâches de l'application est gérée par LANDA, "Local Area Network for Distributed Applications", système multi-serveur développé au LAAS par Jean Marie Garcia et ses collègues et commercialisé par la société Delta Partners S.A. qui permet à des applications réparties de communiquer tout en surveillant leur bon déroulement (cf. [LAN96]). LANDA ne supporte pas de primitives d'échange de données entre machines permettant la copie de mémoire à mémoire. Cet échange 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 pour une gestion par pile, soit après pour une gestion par file[5], soit en le recouvrant pour une gestion par écrasement et mise à jour[6].

5.4.1 Mise en œuvre synchrone

La mise en œuvre synchrone sur un réseau de stations géré par LANDA se fait suivant le même principe que sur le CRAY T3E avec MPI. Chaque tâche réactualise une partie des composantes du vecteur itéré, envoie les composantes frontières aux autres stations par le biais de la fonction non bloquante Lsend(), puis effectue une lecture bloquante des composantes envoyées par les autres processeurs à l'aide de la fonction Lreadb() avant de commencer l'itération suivante.

5.4.2 Mise en œuvre asynchrone

Les itérations asynchrones ont été mises en œuvre en utilisant le mécanisme d'envoi avec recouvrement. Cette stratégie permet d'avoir une boîte aux lettres contenant un seul prototype de chaque message qui correspond au dernier envoi. Dans le cas d'une convergence monotone, l'utilisation de cette stratégie permet de disposer de la valeur la plus proche de la solution et offre une plus grande souplesse au niveau de la mise en œuvre. La composante frontière est envoyée aux autres stations par le biais de la fonction Lsende(). Chaque machine lit le contenu de sa boîte aux lettres avant de faire une nouvelle réactualisation en utilisant la fonction de lecture non bloquante Lread().

5.4.3 Mise en œuvre asynchrone avec communication flexible

Dans le cas de la mise en œuvre asynchrone avec communication flexible, l'envoi et la réception des itérés partiels se font d'une manière similaire à celle utilisée pour la mise en œuvre asynchrone standard et ceci en appelant les fonctions Lsende() et Lread() de la bibliothèque LANDA. De plus, en fixant une politique de communication flexible, chaque processeur envoie les itérés partiels après avoir fait un certain nombre de pas intermédiaires de calcul.

5.5 Autres mises en œuvre

Parmi les toutes premières mises en œuvre d'itérations asynchrones sur machine parallèle on peut citer celle effectuée à la fin des années 70 par Baudet sur 6 processeurs du C.mmp, Carnegie Mellon Multiprocessor, qui était une machine à mémoire partagée (cf. [Bau78]).

La première mise en œuvre des itérations asynchrones que nous avons effectuée a été réalisée sur un réseau de 6 micro-ordinateurs à base de processeurs M 6809 de Motorola

5. fonction LANDA Lsend()
6. fonction LANDA Lsende()

(cf. [EB84], [GBEB82], et [BACEB84]). Les échanges de messages étaient effectués à l'aide de P.I.A. "Parallel Interface Adapter" ou interface d'entrée-sortie parallèle. La réception des messages se faisait sous interruption. Le protocole de communication employé était la classique poignée de main.

Les algorithmes itératifs asynchrones ont été mis en œuvre sur la machine à mémoire partagée Allient FX/8 par Chajakis et Zenios (cf. [CZ91]) et Giraud et Spitéri (cf. [Gir91] et [GS92]). Ces deux derniers auteurs ont aussi mis en œuvre les itérations asynchrones sur la machine à mémoire partagée IBM 3090-400 (cf. [Gir91] et [GS92]). Pour l'étude de diverses mises en œuvre synchrones d'algorithmes itératifs sur des supercalculateurs on pourra se reporter à [ZM86] et [ZL88].

Guivarch [Gui97] a mis en œuvre les itérations asynchrones sur IBM SP2 grâce aux environnements P.V.M. "Parallel Virtual Machine" et MPI. La mise en œuvre a été effectuée suivant un modèle maître/esclaves, le maître ayant des fonctions de lancement et de terminaison de l'application en supplément des fonctions de calcul qui sont communes aussi aux esclaves. L'asynchronisme est essentiellement réalisé à l'aide de fonctions de communication non bloquantes. Un traitement est ajouté afin de récuperer les informations les plus récentes qui ont été rangées au fond du "buffer".

Pour d'autres mises en œuvre asynchrones sur des réseaux de stations ou divers supercalculateurs on pourra se réferer à [HP94] et [HWR92]. A l'occasion de la présentation des diverses applications des itérations asynchrones nous évoquerons aussi au chapitre 6 d'autres mises œuvre.

5.6 Conclusion

Dans le cas d'architectures à mémoire distribuée, la mise en œuvre d'algorithmes itératifs asynchrones repose essentiellement sur la possibilité de mettre en œuvre des communications asynchrones.

Dans un premier temps, nous avons considéré des architectures parallèles constituées de transputers pour lesquelles les communications synchrones sont les seules qui soient disponibles. Nous avons vu que le problème de la mise en œuvre de communications asynchrones qui est apparemment complexe peut être résolu simplement et efficacement grâce aux larges possibilités du langage Occam. Nous avons proposé plusieurs solutions pour mettre en œuvre l'asynchronisme sur ce type de machine.

Puis nous avons considéré un supercalculateur Cray T3E et nous avons vu comment on peut tirer profit d'environnements sophistiqués tels que la bibliothèque SHMEM de Cray (qui permet d'accéder à la mémoire distribuée de cette machine aussi aisément que s'il s'agissait d'une mémoire partagée) afin de mettre en oeuvre simplement les itérations asynchrones.

Nous avons aussi considéré le cas d'un réseau de stations de travail qui par certains aspects est très similaire au cas des architectures à mémoire distribuée. Pour ce type d'architecture les deux points critiques sont essentiellement le temps de latence des communications et la bande passante.

Par ailleurs, la mise en œuvre efficace d'algorithmes itératifs asynchrones repose aussi en grande partie sur la capacité d'utiliser la valeur la plus récente des composantes. L'environnement de travail doit donc fournir des outils efficaces pour accéder aux dernières valeurs reçues, soit par l'utilisation de piles "LIFO", de type dernier arrivé premier sorti, soit en écrasant la valeur précédente des différentes composantes.

Dans le cas des architectures à mémoire distribuée, la mise en œuvre des itérations asynchrones avec communication flexible requiert plus d'échanges de messages entre processeurs ou bien des protocoles de communication plus complexes que dans le cas des itérations asynchrones standards. Toutefois, l'utilisation de bibliothèques telles que SHMEM permet de mettre en œuvre simplement les itérations asynchrones avec communication flexible.

A contrario, les itérations asynchrones avec communication flexible peuvent être mises en œuvre plus simplement que les itérations asynchrones standards sur une machine à mémoire partagée car elles se prêtent naturellement à ce type d'architecture pour laquelle la mémoire est naturellement partagée entre tous les processeurs et la notion de mémoire locale n'a pas grand sens.

Enfin, en ce qui concerne les schémas itératifs synchrones, leur mise en œuvre est relativement complexe sur des machines à mémoire partagée, elle requiert notamment la mise en place de sections critiques qui peuvent induire un surcoût non négligeable. Ce problème est moins important sur des machines à mémoire distribuée ou sur des réseaux grâce à l'utilisation de fonctions de reception blocantes qui peuvent induire néanmoins des temps d'attente importants lorsque les charges de calcul sont déséquilibrées.

Chapitre 6

Applications

La philosophie est écrite dans cet immense livre
qui se tient toujours ouvert devans nos yeux,
je veux dire l'Univers, mais on ne peut le comprendre
si l'on ne s'applique d'abord à en comprendre la langue
et à connaître les caractères avec lesquels il est écrit.
Il est écrit dans la langue des mathématiques et
ses caractères sont des figures géométriques.
Galilée
L'essayeur

Les algorithmes itératifs asynchrones ont été appliqués à de nombreux problèmes. On peut citer notamment : la résolution de systèmes d'équations (cf. [Ros69], [BE82], [LM86], [LG92], et [Bah98]), d'équations différentielles ordinaires (cf. [Mit87], [Bah91], [Bah96], [BGM96a], [BM97], et [RKBM98]), ou d'équations aux dérivées partielles (cf. [Bau78], [Spi83], [MS85b], [Aut87], [BEN88], [Lao88], [SBA89], [SGLM91], [ED91], [GS91], [GS92], [FS94], [ZDE95], [BMPS95], [DZE97], [FS97b], [FS97a], et [Gui97]), de problèmes non linéaires complémentaires (cf. [DLM88]), ou de théorie des jeux (cf. [LB87]), de problèmes de commande optimale (cf. [LM79], [LMS86], [SL88], [LLMS79], et [Com76]), ou de recherche de chemins minimaux (cf. [AR82], [Ber82], [BGM96b], et [BT89]), d'optimisation (cf. [CZ91], [TB86], [BE87], [Tsa89], [BCEZ95], [BC90], [Ber85], et [BF97]), ainsi que divers problèmes discrets (cf. [UD90] et [UD89] ; voir aussi [Rob89], [Rob86], et [Rob95]), ou de recherche de racines d'un polynome (cf. [CF94]). Pour un aperçu général on pourra se référer au livre de Bertsekas et Tsitsiklis (cf. [BT89]).

Nous présentons dans ce chapitre quelques unes des applications que nous avons étudiées. Pour d'autres applications telles que la recherche de chemins minimaux dans les graphes on pourra se reporter à [EBA83] et [EB84].

6.1 Les problèmes de flot convexes dans les réseaux

Les problèmes de flot convexes dans les réseaux interviennent dans de nombreux domaines : distribution d'eau ou de gaz, modèles financiers, réseaux de communication et réseaux de transport. Ce type de problème requiert des calculs intensifs (cf. [ZM86]). Aussi la résolution de ces problèmes de manière parallèle semble t'elle particulièrement intéressante (cf. [BCEZ95], [TB86], et [ZL88]).

6.1.1 Formulation du Problème

Soit $G = (N, A)$, un graphe connexe orienté. N est l'ensemble des sommets, $A \subset N \times N$ est l'ensemble des arcs. Le cardinal de N est noté n. Soit $c_{il} : R \to (-\infty, +\infty]$, la fonction de coût associée à l'arc $(i, l) \in A$, c_{il} est une fonction du flux f_{il} dans l'arc (i, l). Soit b_i l'entrée ou la sortie de produit au sommet $i \in N$ (on a $\sum_{i \in N} b_i = 0$). Le problème consiste à minimiser le coût total sous des constraintes de conservation de flux en chaque sommet du graphe :

$$min \sum_{(i,l)\in A} c_{il}(f_{il}), \text{ sous } \sum_{(i,l)\in A} f_{il} - \sum_{(m,i)\in A} f_{mi} = b_i, \forall i \in N. \tag{6.1}$$

Nous supposons que le problème (6.1) a une solution admissible et nous considérons les hypothèses suivantes sur les fonctions de coût c_{il}.

Hypothèse 6.1 : *La fonction de coût c_{il} est strictement convexe.*

Hypothèse 6.2 : *La fonction de coût c_{il} est semi-continue inférieurement.*

Hypothèse 6.3 : *La fonction convexe conjuguée de c_{il}, définie par*

$$c^*_{il}(t_{il}) = \sup_{f_{il}} \{t_{il}.f_{il} - c_{il}(f_{il})\}, \tag{6.2}$$

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

L'Hypothèse 6.3 implique que $\lim_{|f_{il}|\to\infty} c_{il}(f_{il}) = +\infty$. Par conséquent les ensembles circonscrits par les courbes de niveau de la fonction critère du problème (6.1) sont tous bornés (cf. [Roc70], chapitre 8). Il s'en suit que le problème (6.1) possède une solution optimale qui est unique en raison de l'Hypothèse 6.1. Il résulte aussi de l'Hypothèse 6.1 que les fonctions convexes conjuguées c^*_{il} sont continûment différentiables et que leur gradient noté $\nabla c^*_{il}(t_{il})$, est l'unique f_{il} qui atteint le maximum dans l'équation (6.2) (cf. [Roc70] p. 218 et p. 253 et [BEB87]). Notons enfin que ∇c^*_{il}, qui est le gradient d'une fonction convexe différentiable est monotone croissant.

6.1.2 Le problème dual

Nous considérons plus particulièrement le dual du problème (6.1) :

$$\min_{x \in R^n} q(x), \text{ sans constrainte sur le vecteur } x = \{x_i | i \in N\}, \tag{6.3}$$

où q est la fonction duale définie par :

$$q(x) = \sum_{(i,l) \in A} c^*_{il}(x_i - x_l) - \sum_{i \in N} b_i.x_i. \tag{6.4}$$

Le vecteur x est appelé vecteur prix. Le i-ème prix x_i, est un multiplicateur de Lagrange associé à la i-ème contrainte de conservation de flux. La condition nécessaire et suffisante d'optimalité d'une paire (f^*, x^*) est donnée dans [Roc70] (cf aussi [Ber95]) : un vecteur de flot admissible $f^* = \{f^*_{il} | (i,l) \in A\}$ est optimal pour (6.1) et un vecteur prix $x^* = \{x^*_i | i \in N\}$ est optimal pour (6.3) si et seulement si pour tout $(i,l) \in A$, $x^*_i - x^*_l$ est un sous-gradient de c_{il} en f^*_{il}. Une condition équivalente est : $f^*_{il} = \nabla c^*_{il}(x^*_i - x^*_l)$, pour tout $(i,l) \in A$.

6.1.3 L'ensemble des solutions duales optimales et le problème dual réduit

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 [BEB87]), c'est à dire s'il existe un vecteur de flot admissible $f = \{f_{il} \mid (i,l) \in A\}$ vérifiant : $c'_{il-}(f_{il}) \leq +\infty$ et $c'_{il+}(f_{il}) \geq -\infty$ pour tout $(i,l) \in A$, où c'_{il-} et c'_{il+} représentent respectivement la dérivée à gauche et la dérivée à droite de c_{il}. D'autre part la solution optimale du problème dual n'est pas unique puisque la fonction duale reste inchangée si l'on ajoute la même constante à toutes les composantes du vecteur x. On peut supprimer ce degré de liberté en contraignant le prix d'un sommet. Par exemple, le prix d'un sommet destination noté d peut être fixé à zéro. Considerons l'ensemble $X = \{x \in R^n | x_d = 0\}$ et le problème dual réduit :

$$min_{x \in X} q(x). \tag{6.5}$$

L'ensemble des solutions duales optimales réduit X^* est défini par : $X^* = \{x^* \in X | q(x^*) = \min_x q(x)\}$.

Hypothèse 6.4 : *L'ensemble des solutions duales optimales réduit X^* est non vide et compact.*

Remarque 6.1 : *L'Hypothèse 6.4 n'est pas très restrictive ; par exemple soit $\{f^*_{il} \mid (i,l) \in A\}$ la solution optimale unique du problème primal (6.1) et considerons l'ensemble d'arcs $\bar{A} = \{(i,l) \in A \mid f^*_{il}$ est à l' intérieur de l'ensemble $\{f_{il} \mid c_{il}(f_{il}) < \infty\}\}$; alors l'Hypothèse 6.4 est satisfaite si le sous-graphe $(N, \bar{A})$ est connexe. On pourra se reporter à [BEB87] pour d'autres exemples.*

Théorème 6.1 : *(cf. [BEB87]) Sous les Hypothèses 6.1 à 6.4 le problème dual réduit admet une solution minimale et une solution maximale, c'est à dire qu'il existe $\underline{x}, \overline{x} \in X^*$ tel que $\underline{x} \leq x^* \leq \overline{x}$, pour tout $x^* \in X^*$, où $\underline{x} \leq x$ représente l'ordre partiel naturel dans R^n (cf. Définition A.2).*

Nous notons $g(x)$ le gradient de la fonction duale. Il résulte de (6.4) que les composantes $g_i(x)$ de $g(x)$ sont données par :

$$g_i(x) = \frac{\partial q(x)}{\partial x_i} = \sum_{(i,l)\in A} \nabla c^*_{il}(x_i - x_l) - \sum_{(m,i)\in A} \nabla c^*_{mi}(x_m - x_i) - b_i, i \in N. \tag{6.6}$$

Remarque 6.2 : *(cf. [EB91]) Le problème dual réduit est un problème convexe continûment différentiable, il est donc équivalent au système d'équations suivant :*

$$g_i(x^*) = 0, \ \forall i \in N - \{d\}. \tag{6.7}$$

Remarque 6.3 : *(cf. [EB91]) $g_i(x)$ est uniquement fonction de valeurs locales à savoir : les prix des sommets adjacents au sommet i. De plus $g_i(x)$ est une fonction continue et monotone croissante de x_i, puisque $g_i(x)$ est la dérivée partielle d'une fonction convexe différentiable (cf. [TB87]) ; $g_i(x)$ est aussi une fonction continue et monotone décroissante de x_l, pour tout $l \in N$ tel que $l \neq i$ et $(l,i) \in A$ ou $(i,l) \in A$.*

Le vecteur de R^n dont la i-ème composante est égale à $\hat{x}_i$ et la l-ème composante est égale à x_l pour tout $l \in N - \{i\}$ sera noté par la suite $(\hat{x}_i; x)$.

6.1.4 Les Méthodes

Dans ce paragraphe, nous étudions diverses méthodes itératives pour la résolution du problème dual réduit.

Méthode de relaxation

Puisque le problème dual réduit est sans constrainte et différentiable, il est naturel d'envisager sa résolution numérique au moyen de méthodes itératives de descente; parmi les méthodes spécifiques à l'optimisation, l'intérêt de la méthode de relaxation dans le cas favorable du dual d'un problème d'optimisation non linéaire séparable de type flot dans les réseaux a été établi notamment par Bertsekas dans [BCEZ95], [Ber95], et [BT89]. Notons aussi que cette méthode est particulièrement intéressante pour le problème traité en raison de la simplicité de sa mise en œuvre. Etant donné un vecteur prix $x \in X$, un sommet $i \in N - \{d\}$ est selectionné et son prix x_i prend une valeur $\hat{x}_i$ qui minimise le coût dual par rapport au i-ème prix, les autres prix étant inchangés (on a : $g_i(\hat{x}_i; x) = 0$). La méthode procède de manière cyclique en relaxant les prix de tous les sommets éléments de $N - \{d\}$ (cf. [BEB87]).

Considérons maintenant l'application multivoque $F_i, i \in N - \{d\}$, qui associe à tout vecteur prix $x \in X$, l'ensemble des prix $\hat{x}_i$ qui minimisent le coût dual par rapport au i-ème prix (c'est à dire : $F_i(x) = \{\hat{x}_i \in R | g_i(\hat{x}_i; x) = 0\}$). 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 6.4 que les ensembles $F_i(x), x \in X, i \in N - \{d\}$, sont non vides et compacts. Il s'en suit que les applications minimales et maximales de relaxation $\underline{F}$ et $\overline{F}$ de composantes respectives :

$$\underline{F}_i(x) = \min_{\hat{x}_i \in F_i(x)} \hat{x}_i \text{ et } \overline{F}_i(x) = \max_{\hat{x}_i \in F_i(x)} \hat{x}_i, i \in N - \{d\}, \tag{6.8}$$

sont bien définies sur l'ensemble X (cf. [BEB87]).

Les méthodes de relaxation présentent l'avantage de bien se prêter à une mise en œuvre parallèle. On remarque en particulier que la propriété de décroissance du critère est conservée lorsqu'on réactualise simultanément un sous-ensemble de composantes relatives à des sommets non adjacents. Cependant on peut aussi choisir de réactualiser simultanément toutes les composantes du vecteur x.

Théorème 6.2 : *(cf. [BEB87]) Sous les Hypothèses 6.1 à 6.4, les applications $\underline{F}$ et $\overline{F}$ sont continues et monotone croissantes sur X.*

Théorème 6.3 : *(cf. [BEB87]) Sous les Hypothèses 6.1 à 6.4, les algorithmes de relaxation asynchrones définis par le modèle (2.5) et associés respectivement à l'application minimale de relaxation $\underline{F}$ et à l'application maximale de relaxation $\overline{F}$ convergent respectivement vers $\underline{x}$ et $\overline{x}$ à partir de points initiaux satisfaisant respectivement $x(0) \leq \underline{x}$ et $y(0) \geq \overline{x}$.*

Remarque 6.4 : *(cf. [BEB87]) Ce résultat a été démontré en utilisant les propriétés de croissance monotone et de continuité des applications $\underline{F}$ et $\overline{F}$ et le théorème de convergence asynchrone de Bertsekas. Les suites d'ensembles de niveau engendrées afin de montrer la convergence sont respectivement définies par : $X(j) = \{x \mid \underline{F}^j(\grave{x}) \leq x \leq \underline{x}\}$ et $X(j) = \{x \mid \overline{x} \leq x \leq \overline{F}^j(\acute{x})\}$, où $\grave{x}$ et $\acute{x}$ sont respectivement une sous-solution et une sursolution qui satisfont : $\grave{x} \leq x(0)$ et $y(0) \leq \acute{x}$ et $\underline{F}^j$, $\overline{F}^j$ représentent respectivement la combinaison de j applications de relaxation $\underline{F}$ et $\overline{F}$.*

Remarque 6.5 : *(cf. [BEB87]) Nous présentons sur la figure 6.1 les régions de convergence dans le cas d'un problème simple à trois sommets de degré deux pour lequel les fonctions de coût sont données respectivement par :*

$$c_{12}(f_{12}) = (f_{12})^2,\ c_{23}(f_{23}) = \mid f_{23} \mid + (f_{23})^2,\ c_{31}(f_{31}) = \mid f_{31} \mid + (f_{31})^2.$$

On suppose qu'il n'y a pas de production de produit et donc pas de consommation. La solution optimale est alors :

$$f_{12} = f_{23} = f_{31} = 0,$$

et l'ensemble des solutions duales optimales réduit X^* est donné par :

$$X^* = \{x^* \mid x_3^* = 0,\ x_1^* = x_2^*,\ \ -1 \leq x_1^* \leq 1,\ \ -1 \leq x_2^* \leq 1\}.$$

On remarque que les deux zones délimitées par des traits en pointillé respectivement en bas à gauche et en haut à droite correspondent aux régions de convergence respectivement vers la solution minimale $\underline{x}$ et la solution maximale $\overline{x}$.

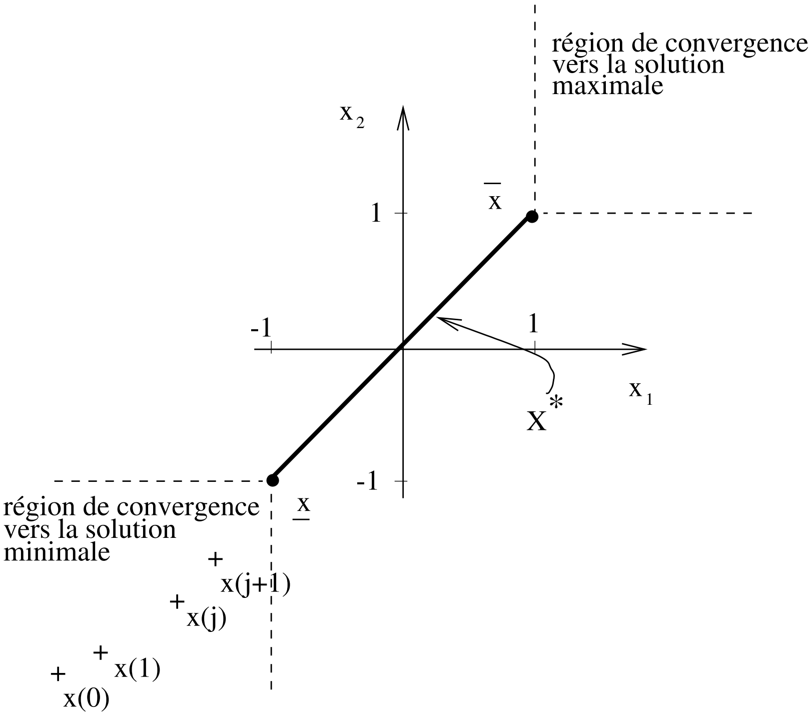


FIG. 6.1 – *Régions de convergence*

Remarque 6.6 : *(cf. [EB91]) Si l'on suppose que c_{il} est continûment differentiable, c_{il}^* est alors strictement convexe, X^* possède un élément unique noté x^*, l'application de relaxation F est dans ce cas univoque, et il résulte de [BEB87] que les algorithmes de relaxation asynchrones définis par le modèle (2.5) et associés à l'application de relaxation F convergent vers x^* à partir de tout point initial. Par ailleurs si l'on considère le système d'équations suivant équivalent au problème dual réduit (6.5) :*

$$g_i(x^*) = 0, \ \forall i \in N - \{d\}, \tag{6.9}$$

on peut caractériser g en terme de M-fonction.

Méthode de gradient

Le problème dual réduit peut être aussi résolu en utilisant une méthode de gradient qui présente l'avantage d'une mise en œuvre simple. Etant donné un vecteur prix $x \in X$, les prix x_i sont modifiés et prennent la valeur $x_i - \frac{1}{\alpha} g_i(x)$ pour tout $i \in N - \{d\}$, où α est une constante positive ; ce processus est répété jusqu'à la convergence. De manière claire la méthode de gradient se prête très bien à une mise en œuvre parallèle pour laquelle toutes les composantes du vecteur itéré sont réactualisées simultanément.

L'application de gradient $F' : X \subset R^{n-1} \rightarrow R^{n-1}$, est définie de la manière suivante :

$$F'_i(x) = x_i - \frac{1}{\alpha} g_i(x), \text{ pour tout } i \in N - \{d\} \text{ et } x \in X. \tag{6.10}$$

Remarque 6.7 : *L'application de gradient F' est continue puisque le gradient de la fonction duale g est continu.*

Hypothèse 6.5 : $0 = arg\ min_{f_{il}} c_{il}(f_{il})$ *et d est l'unique destination de tout le trafic dans le réseau.*

Remarque 6.8 : *L'Hypothèse 6.5 est naturellement satisfaite dans de très nombreuses situations pratiques. Par exemple, les fonctions de coût suivantes satisfont la première partie de l'Hypothèse 6.5 :*

$c_{il}(f_{il}) = (\frac{1}{a_{il}-f_{il}} + b_{il}).f_{il}$, *si* $0 \leq f_{il} < a_{il}$, *et* $c_{il}(f_{il}) = +\infty$, *si* $f_{il} < 0$ *ou* $a_{il} < f_{il}$, *avec* $a_{il} > 0$ *et* $b_{il} \geq 0$ *(ce type de fonction de coût intervient dans les problèmes de communication) ;*

$c_{il}(f_{il}) = a_{il}.|f_{il}| + b_{il}.f_{il}^2$, *avec* $a_{il} \geq 0$ *et* $b_{il} > 0$ *;*

$c_{il}(f_{il}) = a_{il}.max\{f_{il}^2, f_{il}^4\}$, *avec* $a_{il} > 0$.

Théorème 6.4 : *(cf. [EB96a]) Sous l'Hypothèse 6.5, le vecteur $x(0)$ tel que $x_i(0) = 0, \forall i \in N$ est une $\mathcal{A}$-sous-solution.*

Remarque 6.9 : *Il résulte de l'Hypothèse 6.5 et de (6.2) que $\nabla c_{il}^*(0) = 0$ et d'après (6.6) on a : $g_i(x(0)) = -b_i \leq 0, \forall i \in N - \{d\}$.*

Théorème 6.5 : *(cf. [EB96a]) Sous les Hypothèses 6.1 à 6.3 et 6.5, l'intersection I entre l'ensemble des solutions duales optimales réduit et l'orthant non négatif est un polyèdre convexe qui possède un élément minimal noté $\underline{x}'$.*

Hypothèse 6.6 : *c_{il} est fortement convexe de module $\frac{1}{\beta}$.*

Théorème 6.6 : *(cf. [EB96a]) Sous les Hypothèses 6.1 à 6.3 et 6.5, 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 l'application de gradient F' est monotone croissante sur X.*

Remarque 6.10 : *(cf. [EB96a]) On peut montrer également que F' est monotone croissante sur un sous-ensemble de X si c_{il} est fortement convexe sur un sous-domaine associé.*

Théorème 6.7 : *(cf. [EB96a]) Sous les Hypothèses 6.1 à 6.3 et 6.5, les algorithmes de gradient asynchrones définis par le modèle (2.5) et associés à l'application de gradient F' convergent de manière monotone vers $\underline{x}'$ à partir de $x(0)$.*

Remarque 6.11 : *A notre connaissance, ce résultat est le premier à avoir été établi pour des algorithmes de gradient totalement asynchrones. Les seuls résultats de convergence qui avaient été énoncés précédemment portaient sur des algorithmes de gradient partiellement asynchrones (cf. [TBA86] et [BT89]).*

Remarque 6.12 : *Si l'Hypothèse 6.5 est relachée, alors sous les Hypothèses 6.1 à 6.4, les algorithmes de gradient asynchrones définis par le modèle (2.2) et associés à l'application de gradient F' convergent de manière monotone vers $\underline{x}$ à partir d'une $\mathcal{A}$-sous-solution $x(0)$ et vers $\overline{x}$ à partir d'une $\mathcal{A}$-sur solution $y(0)$ (cf. Définitions A.14 et A.13 et Théorème 3.5).*

Relaxation approchée

Lorsque le problème dual réduit présente de fortes non linéarités il peut être intéressant de mettre en œuvre des méthodes de relaxation approchées qui effectuent par exemple des recherches unidirectionnelles de manière inexacte, ceci afin de ne pas consommer trop de ressources de calcul.

Une première classe de méthodes

On peut définir des méthodes de relaxation approchée en utilisant comme opérateur de base l'application de gradient F' définie en (6.10). Ces méthodes procèdent de la manière suivante : étant donné un vecteur prix $x \in X$, les prix x_i sont tour à tour modifiés à la suite d'un nombre de pas de gradient et prennent la valeur $x_i^{q'}$, pour tout $i \in N - \{d\}$ où q' est un entier positif, $x_i^q = F_i'(x_i^{q-1}; x), q = 1, ..., q'$, et $x_i^0 = x_i$. La valeur de q' peut être fixé a priori ou être variable ; dans ce dernier cas q' peut être par exemple le premier entier positif tel que $\left|g_i(x_i^{q'}; x)\right| \leq \epsilon$, ϵ étant une précision fixée. La valeur de q' varie alors suivant le prix x_i considéré et le numéro d'itération. Ce processus est répété de manière cyclique jusqu'à la convergence.

L'application de relaxation approchée $\check{F} : X \subset R^{n-1} \to R^{n-1}$, est définie de la manière suivante :

$$\check{F}_i(x) = x_i^{q'}, \ \forall i \in N - \{d\}, \ \forall x \in X, \tag{6.11}$$

où

$$x_i^q = F_i'(x_i^{q-1}; x), q = 1, ..., q', x_i^0 = x_i. \tag{6.12}$$

Théorème 6.8 : *(cf. [EBSMG96]) L'application de relaxation approchée $\check{F}$ est une sous-application du premier type associée à l'application de relaxation minimale $\underline{F}$ sur $\{x \in X \mid x \leq \underline{x}\}$ (cf. Définitions A.20 et A.25) ; c'est aussi une sur application du premier type associée à l'application de relaxation maximale $\overline{F}$ sur $\{x \in X \mid x \geq \overline{x}\}$ (cf. Définition A.19 et A.23).*

Remarque 6.13 : *(cf. [EBSMG96]) L'application de gradient F' définie en (6.10) est une sous-application continue pour l'ordre sur $\{x \in X \mid x \leq \underline{x}\}$ et une sur application continue pour l'ordre sur $\{x \in X \mid x \geq \overline{x}\}$ (cf. Définition A.21).*

La méthode de relaxation approchée que nous venons de définir se prête bien à une mise en œuvre asynchrone avec communication flexible, pour laquelle toutes les composantes du vecteur itéré sont réactualisées en parallèle par plusieurs processeurs qui peuvent accéder à la valeur courante des différentes composantes du vecteur itéré, c'est à dire éventuellement à des itérés partiels. Nous avons établi le résultat de convergence suivant très voisin du Théorème 3.8.

Théorème 6.9 : *(cf. [EBSMG96]) Toute itération asynchrone par point avec communication flexible définie par le modèle (2.12) et relative à l'application de relaxation approchée $\check{F}$ qui est une sous-application du premier type associée à l'application de relaxation minimale $\underline{F}$ sur $\{x \in X \mid x \leq \underline{x}\}$ et une sur application du premier type associée à l'application de relaxation maximale $\overline{F}$ sur $\{x \in X \mid x \geq \overline{x}\}$ converge de manière monotone respectivement vers $\underline{x}$ à partir d'une sous-solution $x(0)$ et vers $\overline{x}$ à partir d'une sur solution $y(0)$ (cf. Définitions A.16 et A.15).*

Une seconde classe de méthodes

La classe de méthodes associées à l'application de point fixe $\check{F} : X \subset R^{n-1} \to R^{n-1}$, dont les composantes $\check{F}_i$ sont définies de la manière suivante pour tout $x \in X$, et $i \in N - \{d\}$, est aussi de type relaxation inexacte ou approchée. Elle a été étudiée notamment dans [TB87], [BHT87], [BT89], [Tse90], et [CZ91]. On a

$$\check{F}_i(x) = x_i, \text{ si } g_i(x) = 0, \tag{6.13}$$

$$\delta g_i(x) \leq g_i(\check{F}_i(x); x) \leq 0 \text{ et } \check{F}_i(x) \leq \underline{F}_i(x), \text{ si } g_i(x) < 0, \tag{6.14}$$

$$0 \leq g_i(\check{F}_i(x); x) \leq \delta g_i(x) \text{ et } \check{F}_i(x) \geq \overline{F}_i(x), \text{ si } g_i(x) > 0, \tag{6.15}$$

où $\delta \in [0, 1)$.

Théorème 6.10 : *(cf. [EBSMG96]) L'application de relaxation approchée $\check{F}$ est une sous-application associée à l'application de relaxation minimale $\underline{F}$ sur $\{x \in X \mid x \leq \underline{x}\}$; c'est aussi une sur application associée à l'application de relaxation maximale $\overline{F}$ sur $\{x \in X \mid x \geq \overline{x}\}$.*

Remarque 6.14 : *(cf. [EBSMG96]) Dans le cas où c_{il} est continûment différentiable, $\check{F}$ est continue pour l'ordre.*

Dans le cas des problèmes de flot, la terminaison des divers schémas itératifs asynchrones peut être effectuée en se basant sur la valeur de $g_d((x_1(I_1(k)), ..., x_\alpha(I_\alpha(k)))$.

Théorème 6.11 : *(cf. [EBSMG96]) Sous les Hypothèses 6.1 à 6.4, soit $\{x(j)\}$ un algorithme itératif asynchrone avec communication flexible décrit par le modèle (2.12) et associé à une sous-application $\check{F}$ du premier ou du second type qui possède comme point initial une sous-solution $x(0)$, et soit ϵ une constante positive, s'il existe k tel que $g_d((x_1(I_1(k)), ..., x_\alpha(I_\alpha(k))) \leq \epsilon$, alors $\sum_{i \in N-\{d\}} |g_i(x(k'))| \leq \epsilon$, pour tout $k' \geq k$.*

Remarque 6.15 : *(cf. [EBSMG96]) Dans ce cas on a :*

$$g_d(x(j)) = -\sum_{i \in N-\{d\}} g_i(x(j)), \text{ pour tout } j. \tag{6.16}$$

Remarque 6.16 : *(cf. [EBSMG96]) Un résultat similaire peut être montré pour les sur applications.*

Pour les problèmes de flot non linéaires étudiés dans ce paragraphe, la terminaison des itérations asynchrones avec communication flexible peut être donc détectée en faisant un test sur la valeur d'une variable particulière du système à partir de laquelle on peut déduire une information sur l'état global. Un test de terminaison similaire peut être effectué pour les schémas itératifs asynchrones standards correspondant au modèle (2.2) et qui convergent de manière monotone.

6.2 Systèmes Markoviens

Les systèmes Markoviens interviennent dans de nombreux contextes pratiques ; on peut citer par exemple la modélisation de réseaux téléphoniques à commutation de circuit (cf. [BLGA87] et [LG92]) ainsi que la modélisation de systèmes informatiques.

De nombreux auteurs se sont intéressés à la résolution à l'aide de méthodes itératives parallèles de systèmes Markoviens (cf. [LM86], [BLGA87], [BT89], [LG92], et [Bah98]). En particulier, les itérations asynchrones ont été appliquées à la résolution de systèmes Markoviens dans [LM86], [BLGA87], [BT89], et [Bah98]. Nous avons pour notre part étudié des versions asynchrones et aussi synchrones de schémas itératifs parallèles de type relaxation ou Richardson.

6.2.1 Le problème

Nous considérons le système Markovien :

$$Qx^* = 0, \tag{6.17}$$

où x^* est un vecteur de probabilité stationnaire de dimension n (on a $\sum_{i=1}^n x_i = 1$) et Q représente la matrice de transition de dimension (n, n) ; Q est une Z-matrice ($q_{ii} > 0, \forall i; q_{il} \leq 0, \forall i, l$, avec $l \neq i$), irréductible, aperiodique telle que $\sum_{k=1}^n q_{ki} = 0, \forall i$.

Sous les hypothèses précédentes, il existe une solution unique x^* du système (6.17) telle que $\sum_{i=1}^n x_i^* = 1$, et $x^* > 0$ (cf. [BT89]).

Contrairement aux approches développées dans les références [LM86], [BLGA87], [LG92], et [Bah98] pour lesquelles toutes les composantes du vecteur itéré sont calculées par une méthode itérative, nous avons choisi de supprimer une équation du système (6.17) qui possède un degré de liberté et de fixer la valeur d'une composante du vecteur probabilité x. Par exemple, nous avons posé $x_n = 1$ et nous avons supprimé la dernière équation du système (6.17). Nous verrons dans ce qui suit que ce choix a d'importantes conséquences dans la définition de méthodes itératives de résolution ainsi que dans l'étude de leur convergence et que l'on peut toujours retrouver le vecteur probabilité solution du système (6.17) en renormant le vecteur solution obtenu par la méthode itérative utilisée. Nous avons alors le nouveau système :

$$\sum_{l=1}^{n-1} q_{il}.x_l^\star + q_{in} = 0, i = 1, ..., n-1. \tag{6.18}$$

La solution $x^\star$ du système (6.18) et la solution x^* du système (6.17) sont égales à une constante multiplicative près. On a donc $x^\star > 0$. On obtient x^* à partir de $x^\star$ par la relation :

$$x_i^* = \frac{x_i^\star}{\sum_{i=1}^{n} x_i^\star}, i = 1, ..., n. \tag{6.19}$$

6.2.2 Point initial

Nous introduisons maintenant le vecteur de probabilité $x(0)$ qui satisfait :

$$x_1(0) =, ..., = x_{n-1}(0) = 0; x_n(0) = 1. \tag{6.20}$$

On a alors

$$\sum_{l=1}^{n-1} q_{il}.x_l(0) + q_{in} \leq 0, i = 1, ..., n-1, \tag{6.21}$$

puisque

$$q_{in} \leq 0, i = 1, ..., n-1. \tag{6.22}$$

Afin de simplifier la notation, x représentera par la suite le vecteur de R^{n-1} de composantes $x_i, i = 1, ..., n-1$, puisque x_n est une constante. On definit l'ensemble $X = \{x \in R^{n-1} \mid x(0) \leq x \leq x^\star\}$, où la relation d'ordre est de type composante par composante.

6.2.3 Les méthodes

Nous avons considéré la résolution du système (6.18) au moyen de methodes iteratives de relaxation et de Richardson. On définit de la manière suivante l'application de point fixe $F : X \subset R^{n-1} \to R^{n-1}$ de type relaxation :

$$F_i(x) = \hat{x}_i \text{ avec } \sum_{l=1, l\neq i}^{n-1} q_{il}.x_l + q_{ii}.\hat{x}_i + q_{in} = 0, i = 1, ..., n-1. \tag{6.23}$$

L'application de point fixe $F' : X \subset R^{n-1} \to R^{n-1}$ de type Richardson est définie de la manière suivante :

$$F_i'(x) = x_i - \alpha.(\sum_{l=1}^{n-1} q_{il}.x_l + q_{in}), i = 1, ..., n-1, \tag{6.24}$$

où α est une constante positive.

Les méthodes de relaxation et de Richardson se prêtent bien à une mise en œuvre parallèle : les différentes composantes ou les blocs de composantes peuvent être réactualisés concurremment par différents processeurs. On a le résultat suivant.

Théorème 6.12 : *(cf. [EB94b]) Les algorithmes itératifs asynchrones de relaxation ou de Richardson représentés par le modèle (2.2) convergent de manière monotone vers $x^\star$ à partir de $x(0)$.*

Remarque 6.17 : *L'application de point fixe F de type relaxation est continue et monotone croissante sur X puisque Q est une Z-matrice (cf. [EB90] et Remarque 3.4). De plus il résulte de (6.21) que $x(0)$ est une $\mathcal{A}$-sous-solution (cf. Définition A.14).*

Remarque 6.18 : *(cf. [EB94b]) L'application linéaire associée à la matrice Q est continue au sens de Lipschitz : il existe une constante β supérieure ou égale à la norme matricielle $\|Q\|_2$ telle que*

$$\|Q(x-x')\|_2 \leq \beta . \|x-x'\|_2 \, , \forall x, x' \in X. \tag{6.25}$$

En posant $\alpha = \frac{1}{\beta}$ et en utilisant le fait que Q est une Z-matrice, on déduit que F' est continue et monotone croissante sur X (cf. Remarque 3.5).

Remarque 6.19 : *A notre connaissance, aucun résultat de convergence monotone pour les itérations asynchrones n'a été établi dans le cas des systèmes Markoviens. Les seuls résultats connus reposent sur des propriétés de type contraction (cf. [LM86] et [BT89]).*

Par voie de conséquence on a le corollaire suivant.

Théorème 6.13 : *(cf. [EB94b]) Les algorithmes itératifs synchrones de relaxation ou de Richardson convergent de manière monotone vers $x^\star$ à partir de $x(0)$.*

6.3 Equations aux dérivées partielles

Dans ce paragraphe, nous considérons deux problèmes aux limites : le problème de diffusion non linéaire et le problème d'Hamilton-Jacobi-Bellman. Pour d'autres problèmes tels que le problème de l'obstacle et le problème de convection-diffusion on pourra se reporter respectivement à [SMEB95] et [GSMEB96].

6.3.1 Résolution numérique d'un problème de diffusion non linéaire

Soit à résoudre le problème elliptique non linéaire suivant :

Trouver u tel que

$$\begin{cases} -\Delta u + a\ \frac{\partial u}{\partial v} + b\ \frac{\partial u}{\partial w} + e^{cu} = f \quad \text{dans } \Omega, \\ u \mid \partial\Omega = 0, \end{cases} \tag{6.26}$$

où Ω est un domaine ouvert de R^2 (ou R^3), $\partial\Omega$ est la frontière de Ω, les coefficients a, b, c sont des réels, et c est positif. Nous considérons la discrétisation du problème (6.26) par différences finies au moyen du schéma classique à cinq points pour l'opérateur de Laplace et d'un schéma décentré en avant ou en arrière en fonction du signe de a ou b pour les dérivées partielles de u. On obtient alors le système suivant suite à une décomposition par bloc :

$$\sum_{l=1}^{\alpha} A_{il} x_l^* + \varphi_i(x_i^*) = b_i, \ \forall i \in \{1, \ldots, \alpha\}, \tag{6.27}$$

où $x_i^*, b_i \in R^{n_i}$, avec $\sum_{i=1}^{\alpha} n_i = n$, et $\varphi_i(x_i)$ est monotone croissante, convexe, et continûment differentiable, A étant une matrice par bloc. Il résulte de ce type de discrétisation que A est une M-matrice.

Remarque 6.20 : *On peut obtenir un système discretisé présentant les mêmes propriétés en discretisant le problème (6.26) par une méthode d'éléments finis P_1 appropriée.*

Remarque 6.21 : *Nous considérons des M-fonctions $\mathcal{A}$ obtenues par une perturbation monotone diagonale d'une M-matrice A, on a :*

$$\mathcal{A}(x) = Ax - b + \varphi(x), \tag{6.28}$$

où $b \in R^n, \varphi(x) = \text{ diag } \{\ldots, \varphi_i(x_i), \ldots\}$, et les applications univoques φ_i sont monotones croissantes et continues et nous étudions la résolution du système non linéaire suivant :

$$\mathcal{A}(x^*) = 0, \tag{6.29}$$

essentiellement à l'aide de méthode de sous-domaines de type méthode alternée de Schwarz. La méthode alternée de Schwarz peut être décrite simplement de la manière suivante dans le cas de deux sous-domaines discrétisés qui peuvent éventuellement se recouvrir. Soit m_1, m_2, et n trois entiers positifs, n étant le nombre de points de discretisation, m_1 le dernier indice du premier sous-domaine, et m_2 le premier indice du second sous-domaine (on a : $1 < m_2 < m_1 < n$). En introduisant les ensembles :

$$\mathcal{J}_1 = \{1, \ldots, m_1\} \ , \ \mathcal{J}_2 = \{m_2, \ldots, n\}, \ \mathcal{J}' = \mathcal{J}_1 \cap \mathcal{J}_2 = \{m_2, \ldots, m_1\},$$

et en posant $\delta m = m_1 - m_2 + 1$ et $m = n + \delta m$, on peut considérer un vecteur étendu appartenant à $R^{m \cdot}$ On utilise les notations suivantes:

$$\breve{\mathcal{J}}_1 = \{1, \ldots, m_1\}, \ \breve{\mathcal{J}}_2 = \{m_1 + 1, \ldots, m\}, \breve{\mathcal{J}}_1' = \{m_2, \ldots, m_1\}, \ \breve{\mathcal{J}}_2' = \{m_1 + 1, \ldots, m_1 + \delta_m\},$$

où $\breve{\mathcal{J}}_1'$ et $\breve{\mathcal{J}}_2'$ correspondent au recouvrement. Quel que soit $x \in R^n$, les applications $\breve{p}^1, \breve{p}^2$: $R^n \to R^m$ de composantes $\breve{p}_\ell^1(x)$, $\breve{p}_\ell^2(x)$, respectivement, sont définies comme suit.

$$\breve{p}_\ell^1(x) = \begin{cases} x_\ell, \forall \ \ell \in \breve{\mathcal{J}}_1, \\ 0, \ \forall \ \ell \in \breve{\mathcal{J}}_2', \\ x_{\ell - \delta m}, \ \forall \ \ell \in \breve{\mathcal{J}}_2 \ominus \breve{\mathcal{J}}_2', \end{cases}$$

$$\breve{p}^2_\ell(x) = \begin{cases} x_\ell, \ \forall\ \ell \in \breve{\mathcal{J}}_1 \ominus \breve{\mathcal{J}}'_1, \\ 0, \ \forall\ \ell \in \breve{\mathcal{J}}'_1, \\ x_{\ell-\delta_m}, \ \forall\ \ell \in \breve{\mathcal{J}}_2. \end{cases}$$

Quel que soit $\breve{x} \in R^m$, les applications $r^1, r^2 : R^m \to R^n$ de composantes $r^1_\ell(\breve{x}), r^2_\ell(\breve{x})$, respectivement, sont définies de la manière suivante :

$$r^1_\ell(\breve{x}) = \begin{cases} \breve{x}_\ell, \forall\ \ell \in \mathcal{J}_1, \\ \breve{x}_{\ell+\delta m}, \forall\ \ell \in \mathcal{J}_2 \ominus \mathcal{J}', \end{cases}$$

$$r^2_\ell(\breve{x}) = \begin{cases} \breve{x}_\ell, \forall\ \ell \in \mathcal{J}_1 \ominus \mathcal{J}', \\ \breve{x}_{\ell+\delta m}, \forall\ \ell \in \mathcal{J}_2. \end{cases}$$

Quel que soit $i \in \{1,2\}$ et $x \in R^n$, on a alors $r^i(\breve{p}^i(x)) = x$. On considère le système d'équations :

$$\breve{\mathcal{A}}(\breve{x}^*) = 0, \tag{6.30}$$

où l'application étendue $\breve{\mathcal{A}} : R^m \to R^m$ de composantes $\breve{\mathcal{A}}_i(\breve{x}),\ i = 1, \ldots, m$, est définie de la manière suivante :

$$\begin{cases} \breve{\mathcal{A}}_i(\breve{x}) = \mathcal{A}_i(r^1(\breve{x})), \ \forall\ i \in \breve{\mathcal{J}}_1, \\ \breve{\mathcal{A}}_i(\breve{x}) = \mathcal{A}_{i-\delta m}(r^2(\breve{x})), \ \forall\ i \in \breve{\mathcal{J}}_2. \end{cases}$$

D'après [ED91], si $A \in \mathcal{L}(R^n\,;\,R^n)$ est une M-matrice, alors la matrice $\breve{A} \in \mathcal{L}(R^m\,;\,R^m)$ qui résulte du processus d'extension décrit ci-dessus est aussi une M-matrice, l'opérateur non linéaire diagonal $\varphi : R^n \to R^n$ peut être étendu de manière analogue ; de plus, l'application $\breve{\mathcal{A}}$ est une M-fonction surjective. Le cas avec α sous-domaines, $\alpha > 2$, où aucune composante n'appartient à plus de deux sous-domaines peut être réduit au cas où $\alpha = 2$ en considérant une décomposition par sous-domaines en bande avec une numérotation rouge-noir. Cette remarque peut être étendue au cas où $\varphi(x)$ est un opérateur diagonal monotone maximal multivoque (cf. [BM93]) ; on doit alors résoudre le problème non linéaire algébrique

$$b - Ax \in \varphi(x). \tag{6.31}$$

Soit à résoudre le système $\breve{\mathcal{A}}(\breve{x}^*) = 0$ ou $\mathcal{A}(x^*) = 0$. L'application $\mathcal{A}_i(\hat{x}_i; x)$ de R^{n_i} dans R^{n_i} définie par : $\mathcal{A}_i(\hat{x}_i; x) = A_{ii}\hat{x}_i + \varphi_i(\hat{x}_i) + \sum_{l \neq i} A_{il}x_l - b_i$, est une M-fonction surjective ; comme dans la Remarque 3.3 on peut définir implicitement une application de point fixe F de R^n dans R^n associée au problème (6.27).

Nous introduisons la matrice bloc-diagonale $C(x)$, derivée de la méthode de Newton, dont les blocs diagonaux $C_i(x_i)$ sont définis par

$$C_i(x_i) = A_{ii} + \varphi'_i(x_i). \tag{6.32}$$

La matrice $C_i(x_i)$ est une M-matrice car la fonction φ_i est convexe et la matrice A_{ii} est une M-matrice.

Considérons l'application $\hat{F}$ de composantes $\hat{F}_i$ définies par

$$\hat{F}_i(x) = x^q_i,\ \forall i \in \{1, \ldots, \alpha\}, \tag{6.33}$$

où q est un entier positif, $x_i^0 = x_i$, et x_i^q est le q-ème iteré produit par l'algorithme :

$$x_i^q = x_i^{q-1} - C_i^{-1}(x_i^{\rho'(q-1)}).\mathcal{A}_i(x_i^{q-1};x), q = 1, 2, \ldots \tag{6.34}$$

où $0 \leq \rho'(q-1) \leq q-1$.

Théorème 6.14 : *(cf. [MEBS98]) L'application $\hat{F}$ de composantes $\hat{F}_i$ définies par*

$$\hat{F}_i(x) = x_i^1,\ \forall i \in \{1, \ldots, \alpha\}, \tag{6.35}$$

est une $\mathcal{A}$-sur application continue pour l'ordre associée à F.

Théorème 6.15 : *(cf. [MEBS98]) L'application $\hat{F}$ de composantes $\hat{F}_i$ définies par*

$$\hat{F}_i(x) = x_i^q,\ q > 1,\ \forall i \in \{1, \ldots, \alpha\}, \tag{6.36}$$

est une $\mathcal{A}$-sur application du premier type associée à F.

Remarque 6.22 : *(cf. [MEBS98]) La propriété de continuité de l'application $\hat{F}_i$ dans le cas où $q = 1$ résulte du fait que $C(x)$ est une matrice bloc-diagonale dont les blocs satisfont : $C_i(x) = A_{ii} + \varphi_i'(x)$ et que l'application $\mathcal{A}(x)$ est continue. De plus on a :*

$$(A_{ii} + \varphi_i'(x_i))^{-1} \leq A_{ii}^{-1},$$

puisque A est une M-matrice et φ' est positive (cf. [OR70], p. 55). Il s'en suit que le rayon spectral de $C^{-1}(x)$ est inférieur ou égal au rayon spectral de l'inverse de la matrice bloc-diagonale de blocs A_{ii} qui est une M-matrice. Par conséquent, l'application linéaire associée à la matrice $C^{-1}(x)$ est continue au sens de Lipschitz et $C^{-1}(x)$ est uniformément continue au sens de Lipschitz.

Remarque 6.23 : *(cf. [MEBS98]) Les applications $\hat{F}$ sont aussi des $\mathcal{A}$-sur applications du second type associées à F en raison des propriétés de convergence de la méthode de Newton (cf. [OR70] paragraphe 13.3.4).*

Remarque 6.24 : *(cf. [MEBS98]) Les applications $\hat{F}$ sont aussi des $\mathcal{A}$-sous-applications du premier et du second type associées à F.*

Problèmes pseudolinéaires et techniques de multidécomposition

Dans le cas des problèmes pseudolineaires du type :

$$Ax^* + \varphi(x^*) = 0, \tag{6.37}$$

où A est une M-matrice de dimension (n, n), $\varphi : R^n \to R^n$ est un opérateur diagonal monotone maximal continu, on peut considérer les "splittings" ou décompositions régulières suivantes de la matrice A :

$$A = M^l - N^l, \; l = 1, ..., m, \tag{6.38}$$

où $(M^l)^{-1} \geq 0$ et $N^l \geq 0$. Soit $F^l : R^n \to R^n$, $l = 1, ..., m$, les applications de point fixe associées au problème (6.37) et définies par :

$$F^l(x) = \hat{x} \text{ tel que } M^l\hat{x} = N^l x - \varphi(x). \tag{6.39}$$

Définition 6.1 : *(cf. [BMR97]) Une multidécomposition formelle associée au problème (6.37) correspond à l'ensemble de problèmes de point fixe suivant :*

$$x^* = F^l(x^*) = 0, \; l = 1, ..., m. \tag{6.40}$$

Soit $E = (R^n)^m$, nous considérons la décomposition par blocs suivante de E:

$$E = \prod_{l=1}^{m} E_l, \tag{6.41}$$

où $E_l = R^n$. Chaque sous-espace E_l est muni de l'ordre partiel naturel dans R^n. Soit $\breve{x} \in E$, on a la décomposition par blocs suivante de $\breve{x}$:

$$\breve{x} = \{\breve{x}_1, \ldots, \breve{x}_l, \ldots, \breve{x}_m\} \in \prod_{l=1}^{m} E_l. \tag{6.42}$$

Définition 6.2 : *(cf. [BMR97]) L'application de point fixe étendue $T : E \to E$, de composantes T_l associée à la multidécomposition formelle est définie de la manière suivante :*

$$T_l(\breve{x}) = F^l(z_l) \text{ avec } z_l = \sum_{k=1}^{m} W_{lk}\breve{x}_k, l = 1, ..., m, \tag{6.43}$$

où W_{lk} sont des matrices de pondération diagonales, non negatives, qui satisfont pour tout $l \in \{1, \ldots, m\}$:

$$\sum_{k=1}^{m} W_{lk} = I, \tag{6.44}$$

I étant la matrice identité dans $L(E_l)$.

Soit la décomposition par blocs suivante de l'application T :

$$T(\breve{x}) = \{T_1(\breve{x}), \ldots, T_l(\breve{x}), \ldots, T_m(\breve{x})\} \in \prod_{l=1}^{m} E_l. \tag{6.45}$$

L'application de point fixe étendue T est associée au problème non linéaire étendu suivant :

$$\breve{\mathcal{A}}(\breve{x}^*) = 0, \tag{6.46}$$

où l'application $\breve{\mathcal{A}} : E \to E$ est définie par :

$$\breve{\mathcal{A}}(\breve{x}) = \breve{A}\breve{x} + \breve{\varphi}(\breve{x}), \tag{6.47}$$

la fonction $\breve{\varphi} : E \to E$ étant un operateur monotone étendu et pour tout $l \in \{1, \ldots, m\}$:

$$\breve{A}_l\breve{x} = M^l\breve{x}_l - N^l \sum_{k=1}^{m} W_{lk}\breve{x}_k. \tag{6.48}$$

Par la suite $\breve{\mathcal{A}}_l(\breve{x}_1, \ldots, \breve{x}_{l-1}, y_l, \breve{x}_{l+1}, \ldots, \breve{x}_m)$ sera aussi noté $\breve{\mathcal{A}}_l(y_l\,;\,\breve{x})$.

Théorème 6.16 : *([EBSM97]) Sous les hypothèses précédentes, L'application $\breve{\mathcal{A}}$ est une M-fonction continue surjective.*

Il résulte du Théorème 6.16 que pour tout $l \in \{1, \ldots, m\}$ et tout $\breve{x} \in E$, l'application $y_l \to \breve{\mathcal{A}}_l(y_l, \breve{x})$, est une M-fonction continue surjective de E_l dans E_l (cf. [Rhe70], théoreme 3.5). De plus il résulte du Théorème 6.16 que pour tout $l \in \{1, \ldots, m\}$ et $\breve{x} \in E$, le problème $\mathcal{A}_l(y_l\,;\,\breve{x}) = 0$, a une solution unique y_l.

Il résulte des hypothèses ci-dessus que T est monotone croissante sur E (cf. [Mie86]).

Les méthodes de multidécomposition peuvent donc être combinées avec des itérations asynchrones avec communication flexible.

6.3.2 Résolution numérique d'un problème d'Hamilton-Jacobi-Bellman discrétisé et linéarisé

Nous considérons le problème suivant :

$$\begin{cases} \text{trouver } u \text{ tel que} \\ \max\,\{\mathbf{A}^1u - f^1, \mathbf{A}^2u - f^2\} = 0, \text{ partout dans } \Omega, \\ u \mid \partial\Omega = 0, \end{cases} \tag{6.49}$$

où $\mathbf{A}^1$ et $\mathbf{A}^2$ sont deux opérateurs elliptiques du second ordre satisfaisant le Principe du Maximum et f_1, f_2 appartiennent à $L^2(\Omega)$.

En effectuant une discrétisation appropriée du problème (6.49) par différences finies et en supposant en particulier que les matrices d'incidence B^1 et B^2 associées aux matrices de discrétisation sont identiques, on obtient le problème discret suivant :

$$\begin{cases} \text{trouver } x^* \text{ solution de} \\ \max\ (A^1.\, x^* - b^1, A^2.\, x^* - b^2) = 0, \end{cases} \tag{6.50}$$

où b^1, $b^2 \in R^n$ et A^1, A^2 sont des matrices de dimension (n, n) de composantes a^1_{il}, a^2_{il}, respectivement, qui satisfont les conditions suivantes :

$$a^r_{ii} > 0, a^r_{il} \leq 0, i = 1, ..., n, l = 1, ..., n, l \neq i, r = 1, 2, \tag{6.51}$$

$$\sum_l a^r_{il} \geq 0, i = 1, ..., n, r = 1, 2, \tag{6.52}$$

$$\text{il existe au moins un } i \text{ tel que } \sum_l a^1_{il} > 0 \text{ et } \sum_l a^2_{il} > 0, \tag{6.53}$$

$$\text{les matrices } A^1 \text{ et } A^2 \text{ sont irréductibles.} \tag{6.54}$$

Remarque 6.25 : *(cf. [MEBS98]) La matrice d'incidence B^1, respectivement B^2, de dimension (n, n) est engendrée à partir de la matrice A^1, respectivement A^2, en replaçant les composantes a^1_{il}, respectivement a^2_{il}, par 1 si $a^1_{il} \neq 0$, respectivement $a^2_{il} \neq 0$.*

Remarque 6.26 : *(cf. [MEBS98]) Il résulte de (6.51) à (6.54) que les matrices A^1 et A^2 sont des M-matrices (cf. [Var62]).*

Remarque 6.27 : *(cf. [MEBS98]) Le problème (6.50) peut être linéarisé de la manière suivante :*

$$\mathcal{A}(x^*) = C(x^*).\, x^* - b(x^*) = 0, \tag{6.55}$$

où $b(x^*) \in R^n$ et $C(x^*)$ est une matrice de dimension (n, n). Si $(A_1 x - b_1)_i$ est supérieur à $(A_2 x - b_2)_i$, alors la i-ème ligne de la matrice $C(x)$ est identique à la i-ème ligne de la matrice A_1, sinon elle est identique à la i-ème ligne de la matrice A_2. Le vecteur $b(x)$ est défini de manière analogue. Il résulte des hypothèses précédentes que la matrice $C(x)$ est une matrice irréductible diagonale dominante, $C(x)$ est une M-matrice (cf. [Var62]) et donc $\mathcal{A}$ est une M-fonction. L'application $\mathcal{A}_i(\hat{x}_i; x)$ de R^{n_i} dans R^{n_i} définie par :

$$\mathcal{A}_i(\hat{x}_i; x) = \max\ (A^1_{ii}\hat{x}_i - (b^1_i - \sum_{l \neq i} A^1_{il} x_l), A^2_{ii}\hat{x}_i - (b^2_i - \sum_{l \neq i} A^2_{il} x_l)), \tag{6.56}$$

est une M-fonction continue surjective.

Suivant la Remarque 3.3 on peut définir une application de point fixe F de R^n dans R^n associée au problème (6.55). On definit aussi l'application $\check{F}$ de R^n dans R^n de composantes $\check{F}_i$ de la manière suivante :

$$\check{F}_i(x) = x^q_i,\ \forall i \in \{1, ..., \alpha\}, \tag{6.57}$$

où q est un entier positif et x^q est engendré par l'algorithme général suivant :

$$x^q = x^{q-1} - C^{-1}(x^{\rho(q-1)})\mathcal{A}(x^{q-1}), q = 1, 2, ... \tag{6.58}$$

qui est derivé de l'équation de point fixe :

$$x^* = x^* - C^{-1}(x^*)\mathcal{A}(x^*) \tag{6.59}$$

et qui est appliquée ici à la résolution du problème :

$$\mathcal{A}_i(\hat{x}_i; x) = 0. \tag{6.60}$$

Théorème 6.17 : *(cf. [MEBS98]) L'application $\check{F}$ de composantes $\check{F}_i$ définies par*

$$\check{F}_i(x) = x_i^1, \ \forall i \in \{1, \ldots, \alpha\}, \tag{6.61}$$

est une $\mathcal{A}$-sur application continue pour l'ordre associée à F.

Remarque 6.28 : *La continuité de l'application $\check{F}$ résulte de la continuité de l'application $\mathcal{A}$ puisque l'enveloppe convexe de fonctions continues dans R^n est continue.*

Théorème 6.18 : *(cf. [MEBS98]) L'application $\check{F}$ de composantes $\check{F}_i$ definies par*

$$\check{F}_i(x) = x_i^q, \ q > 1, \ \forall i \in \{1, \ldots, \alpha\}, \tag{6.62}$$

est une $\mathcal{A}$-sur application du premier type associée à F.

6.4 Conclusion

Dans ce chapitre nous avons mis en évidence que les schémas de calcul asynchrones peuvent être combinés à un grand nombre de méthodes itératives de type : relaxation, relaxation approchée, Richardson ou gradient, Schwarz, et multisplitting afin d'être appliquées à des problèmes très variés comme des problèmes d'optimisation de type flot dans les réseaux, des systèmes Markoviens, et des équations aux dérivées partielles. Nous avons montré chaque fois que le contexte dans lequel on se plaçait garantissait la convergence. En particulier, nous avons fait le lien entre certains résultats de convergence établis dans des contextes applicatifs spécifiques comme les problèmes de flot de coût minimum et les systèmes Markoviens et les résultats généraux de convergence présentés au chapitre 3.

Nous avons aussi montré que certaines applications comme les problèmes de flot dans les réseaux peuvent fournir la possibilité de concevoir des méthodes de terminaison spécifiques, simples, et originales découlant directement de l'analyse du problème et qui peuvent être fondées sur l'observation de certaines variables contenant une information sur l'état global du système. Cette approche doit être différenciée de l'approche développée au chapitre 4 et qui a pour objectif de concevoir des méthodes de terminaison générales.

Chapitre 7

Etude de performance

> S'il en allait du raisonnement sur un problème difficile comme du transport des charges, sachant qu'une multitude de chevaux transporteront plus de sac de blé qu'un seul cheval, j'accorderais qu'une multitude de raisonnements puissent être plus efficaces qu'un seul ; mais le raisonnement est discours, course plutôt que transport et un seul cheval arabe courra mieux que cent chevaux frisons.
>
> *Galilée*
> *Saggiatore*

L'analyse de performance présentée dans ce chapitre repose sur des expérimentations numériques portant sur des problèmes de flot de coût minimum (cf. paragraphe 6.1), des systèmes Markoviens (cf. paragraphe 6.2), et des problèmes aux limites (cf. paragraphe 6.3). Toutefois, il ne nous est pas possible de présenter une synthèse de tous les résultats expérimentaux en raison des contraintes portant sur le nombre de pages de ce mémoire. Aussi nous avons choisi de privilégier essentiellement les résultats qui ont été obtenus pour des problèmes de flot de coût minimum pour lesquels nous avons utilisé des architectures parallèles très variées. Une synthèse de toutes les expérimentations numériques serait de toute façon hors de propos dans le contexte d'un mémoire d'habilitation, de plus nous ne souhaitons pas donner une place prédominante dans ce document au chapitre portant sur les performances, ni déflorer les études menées actuellement par Mohamed Jarraya dans le cadre de son travail de thèse, enfin, nous n'avons pas la prétention d'avoir mené une étude exhaustive. Il reste en effet à accomplir un important travail d'expérimentation avant de tirer des conclusions définitives.

7.1 Première classe de méthodes de relaxation approchée et méthode du gradient conjugué

Dans le cadre de l'étude des problèmes de flot dans les réseaux, nous montrons tout d'abord l'intérêt des méthodes de relaxation approchée pour lesquelles les applications de point fixe associées sont des sous-applications. Nous rappelons que ce concept est de première importance pour l'étude des itérations parallèles asynchrones avec communication flexible. Nous nous intéressons plus particulièrement à la première classe de méthodes de relaxation approchée présentée au paragraphe 6.1.4 qui permet d'engendrer très simplement des algorithmes asynchrones avec communication flexible ; nous comparons ce type de méthode de relaxation approchée à la méthode du gradient conjugué bien connue pour son efficacité.

Nous avons considéré divers problèmes de flot dans des réseaux : des problèmes de flot à coût quadratique qui ont un très large éventail d'applications, des problèmes de distribution, et des problèmes de communication. Les fonctions de coût sont respectivement :

$$c_{il}(f_{il}) = f_{il}^2, \tag{7.1}$$

dans le cas d'un coût quadratique, on a alors : $\nabla c_{il}^*(x_i - x_l) = \frac{1}{2}(x_i - x_l)$, on remarque que cette fonction de coût satisfaît les Hypothèses 6.1 à 6.3, 6.5, et 6.6 du chapitre 6 ;

$$c_{il}(f_{il}) = |f_{il}|^{\frac{1+b}{b}}, \tag{7.2}$$

avec $b = 1.85$, dans le cas correspondant à des flots turbulents dans des canalisations d'un réseau de distribution (cf. [BD65], [Por69], et [Rhe70]), on a alors : $\nabla c_{il}^*(x_i - x_l) = sign(x_i - x_l)\,|x_i - x_l|^{1.85}$, cette fonction de coût satisfaît les Hypothèses 6.1 à 6.3, 6.5, ainsi que l'Hypothèse 6.6 sur un sous-domaine borné ;

$$c_{il}(f_{il}) = (\frac{1}{a_{il} - f_{il}} + b_{il}).f_{il}, \tag{7.3}$$

si $0 \leq f_{il} < a_{il}$ et $c_{il}(f_{il}) = +\infty$ si $f_{il} < 0$ ou $a_{il} \leq f_{il}$ (nous avons pris : $a_{il} = 1$ et $b_{il} = 0,5$), ce type de fonction de coût est souvent rencontré dans des problèmes de communication comme le routage des paquets dans un réseau informatique, on a alors : $\nabla c_{il}^*(x_i - x_l) = 1 - (\frac{1}{x_i - x_l - \frac{1}{2}})^{\frac{1}{2}}$, si $x_i - x_l \geq 1,5$ et $\nabla c_{il}^*(x_i - x_l) = 0$ si $x_i - x_l \leq 1,5$; cette fonction de coût satisfaît les Hypothèses 6.1 à 6.3, 6.5, et 6.6.

Nous avons considéré divers réseaux maillés, le nombre de sommets et d'arcs variant respectivement de 48 à 144 et de 77 à 237. Pour chaque problème, le degré des différents sommets est compris entre de 2 et 4. Nous avons étudié le cas d'une seule entrée et d'une seule sortie de produit égales à 1.

La méthode de relaxation approchée appartenant à la première classe et désignée encore par relaxation approchée I est utilisée avec une précision $\epsilon = 10^{-3}$ pour le test relatif à la recherche unidirectionnelle : $\left|g_i(x_i^{q'}; x)\right| \leq \epsilon$. Nous avons choisi $\beta = 0.5$ pour le problème de flot quadratique et le problème de communication et $\beta = 0.74$ pour le problème de distribution. Nous rappelons que le pas du gradient intervenant dans la méthode de relaxation approchée I est déduit en prenant : $\alpha = \beta . \max_{i \in N} a_i$, où a_i est le degré du sommet $i \in N$. Le point initial de la méthode de relaxation approchée I et de la méthode du gradient conjugué est la sous-solution $\{x_i = 0 \mid i \in N\}$. La terminaison intervient lorsque $g_d(x(j)) \leq 10^{-1}$.

Les résultats expérimentaux obtenus sur un processeur du Cray T3E et présentés dans le Tableau 7.1 montrent que la méthode de relaxation approchée I présente d'excellents résultats comparativement à la méthode du gradient conjugué. Le seul cas où la méthode du gradient conjugué est légèrement plus rapide étant celui du problème de communication. Les bonnes performances de la méthode de relaxation approchée I pour ce type de problème s'expliquent par la propriété de séparabilité du critère primal. Il est à noter que dans le cas des problèmes quadratiques, la méthode de relaxation approchée I est équivalente à une méthode de relaxation exacte, l'optimum suivant une direction pouvant alors être obtenu de manière analytique. Pour une étude de la valeur de la fonction duale, du nombre d'itérations, et du temps de résolution en fonction de la précision du test de terminaison on pourra se référer à [EB96a]. On notera par ailleurs que la méthode du gradient conjugué se prête moins bien à une mise en œuvre parallèle que la méthode de relaxation approchée I.

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

TAB. 7.1 – *Problèmes de flot : comparaison de méthodes itératives dans le cas séquentiel*

7.2 Méthodes de relaxation approchée

Dans ce paragraphe, nous comparons les performances de diverses méthodes de relaxation approchée appartenant aux deux classes présentées au paragraphe 6.1.4 et qui sont associées à des sous-applications. Les problèmes traités sont des problèmes de flot dans des réseaux. Les fonctions de coût qui vérifient (7.2) correspondent au cas de flots turbulents dans des canalisations. Nous avons considéré les mêmes réseaux qu'au paragraphe 7.1. Toutefois, chaque problème présente trois entrées et trois sorties de produit non nulles. Pour la première classe de méthodes de relaxation approchée, nous considérons l'algorithme général, qui est aussi appelé relaxation approchée I, avec recherche unidirectionnelle inexacte et une précision $\epsilon = 10^{-2}$ pour le test $\left|g_i(x_i^{q'}; x)\right| \leq \epsilon$, ainsi que la méthode de gradient qui apparaît comme un cas particulier de cette classe de méthodes itératives. En ce qui concerne la deuxième classe de méthodes de relaxation approchée présentée au paragraphe 6.1.4, nous avons considéré plus précisément la méthode de descente duale mise en œuvre dans [CZ91], qui est aussi appelée relaxation approchée II.

dimension du vecteur prix	48	72	96	144
relaxation approchée I	31.53	94.70	210.77	674.89
gradient	41.73	135.11	312.65	1027.81
relaxation approchée II	58.92	194.71	456.02	1518.07

TAB. 7.2 – *Problèmes de flots turbulents : temps des différentes méthodes de relaxation approchée*

Les résultats expérimentaux obtenus sur un processeur du Tnode et présentés dans le Tableau 7.2, indiquent clairement que la méthode générale de relaxation approchée appartenant à la première classe et dont l'application de point fixe est définie par (6.11) avec $\epsilon = 10^{-2}$ a été plus rapide que la méthode du gradient (pour laquelle $q' = 1$) et que la méthode de relaxation approchée appartenant à la deuxième classe. Notons par ailleurs la différence importante des temps de calcul sur le Cray T3E et sur le T800 du Tnode pour des problèmes qui ne diffèrent que légèrement.

7.3 Mesures de performance et paramètres importants

Les principales mesures de performance des algorithmes itératifs parallèles sont le temps de la méthode parallèle en utilisant p processeurs qui est noté t_p, l'accélération avec p processeurs qui est notée $a(p)$ et définie comme suit :

$$a(p) = \frac{t_s}{t_p}, \tag{7.4}$$

où t_s est le temps de la méthode séquentielle, et enfin l'efficacité $e(p)$ qui est définie de la manière suivante :

$$e(p) = \frac{t_s}{t_p.p}, \tag{7.5}$$

cette dernière mesure étant celle qui est la plus souvent utilisée.

Les résultats expérimentaux obtenus pour les différents problèmes abordés au chapitre 6 et pour les diverses mises en œuvre présentées au chapitre 4 (cf. [EB96a], [EBSMG96], [EBGJ$^+$98], [EBGMS96], [JEBG98], [EB96b], [MEBS98], et [GSMEB96]) ont permis d'établir que 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 est notée g et définie de la manière suivante :

$$g = \frac{t_{cal}}{t_{com}}, \tag{7.6}$$

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 (cf. [BPF89] et [KW84]).

Dans les paragraphes suivants nous détaillons l'influence des facteurs cités ci-dessus sur les performances des algorithmes itératifs parallèles en nous basant sur les multiples expérimentations effectuées sur diverses architectures parallèles de type mémoire distribuée ou mémoire partagée.

7.4 Performances de diverses mises en œuvre parallèles

Les performances des différentes méthodes de relaxation approchée considérées au paragraphe 7.2 ont été comparées pour diverses mises en œuvre parallèles synchrones et asynchrones sur la machine Tnode, le nombre de processeurs étant égal à 2, 4, 8 ou 16.

Nous avons traité des problèmes identiques à ceux du paragraphe 7.2. Pour tous les problèmes et méthodes considérés, nous avons équilibré le nombre de sommets sur les différents processeurs. De plus, à nombre de processeurs égal, le partitionnement des données est identique pour les différentes méthodes considérées.

Les méthodes générales appartenant à la première et à la seconde classe de méthodes de de relaxation approchée sont notées respectivement I et II, la méthode du gradient est notée G. Les mises en œuvre synchrone, asynchrone, et asynchrone avec communication flexible qui sont notées respectivement s, a, et acf, correspondent aux descriptions des paragraphes 5.1.1, 5.1.3, et 5.1.5. Il est à noter que seule la méthode I a été mise en œuvre de manière asynchrone avec communication flexible, les méthodes G et II étant des méthodes à un pas de calcul. Les Tableaux 7.3 à 7.6 donnent le temps et l'efficacité des différentes mises en œuvre.

	2 processeurs		*4 processeurs*		*8 processeurs*	
méthode	temps	efficacité	temps	efficacité	temps	efficacité
Is	17.91	0.880	10.20	0.773	5.85	0.674
Ia	17.24	0.914	9.40	0.839	5.74	0.687
Iacf	17.16	0.919	9.19	0.858	5.42	0.727
Gs	21.29	0.980	10.87	0.960	5.52	0.945
Ga	21.00	0.994	10.64	0.981	5.56	0.938
IIs	30.67	0.961	16.93	0.870	9.44	0.780
IIa	29.98	0.983	15.99	0.921	9.54	0.772

TAB. 7.3 – *Temps et efficacité des mises en œuvre parallèles des méthodes de relaxation approchée sur le Tnode pour un problème de flot de dimension 48*

	2 processeurs		*4 processeurs*		*8 processeurs*	
méthode	temps	efficacité	temps	efficacité	temps	efficacité
Is	53.81	0.880	29.32	0.808	16.13	0.734
Ia	52.09	0.909	27.21	0.870	15.15	0.781
Iacf	51.59	0.918	26.93	0.879	14.56	0.813
Gs	68.55	0.985	34.74	0.972	17.56	0.962
Ga	67.84	0.996	34.19	0.988	17.54	0.963
IIs	99.99	0.974	53.44	0.911	28.82	0.845
IIa	98.30	0.990	50.56	0.963	27.83	0.875

TAB. 7.4 – *Temps et efficacité des mises en œuvre parallèles des méthodes de relaxation approchée sur le Tnode pour un problème de flot de dimension 72*

Dans le cas des méthodes parallèles de gradient (G), les charges de calcul sont équilibrées de manière déterministe puisque la réactualisation d'un prix consiste essentiellement en un calcul de gradient et les sommets ont été répartis équitablement entre les différents processeurs. Les Tableaux 7.3 à 7.6 montrent qu'une mise en œuvre asynchrone accélère de manière très efficace la méthode de gradient. Les méthodes de gradient asynchrones ont été généralement plus rapides que les méthodes de gradient synchrones pour une granularité suffisamment grande. Les très bonnes performances des méthodes de gradient synchrones résultent de l'équilibre des charges de calcul, en conséquence les temps d'inactivité dûs aux synchronisations sont très réduits. Notons enfin que les très bonnes efficacités des méthodes de gradient synchrones et asynchrones n'ont pas été suffisantes pour rendre ces méthodes les plus rapides.

Dans le cas des mises en œuvre parallèles des méthodes de relaxation approchée appartenant à la deuxième classe (II), les charges de calcul sont équilibrées de manière déterministe pour des raisons identiques à celles invoquées dans le cas de la méthode du gradient. Les Tableaux 7.3 à 7.6 indiquent qu'une mise en œuvre asynchrone a accéléré de manière très efficace la méthode II. Les mises en œuvre asynchrones ont été plus efficaces que les mises en œuvre synchrones pour une granularité suffisante.

Les méthodes parallèles de relaxation approchée appartenant à la première classe (I) sont en général caractérisées par un déséquilibre indéterministe des charges de calcul puisque la ré-

	2 processeurs		*4 processeurs*		*8 processeurs*		*16 processeurs*	
méthode	temps	efficacité	temps	efficacité	temps	efficacité	temps	efficacité
Is	117.75	0.895	62.92	0.838	34.00	0.775	19.46	0.677
Ia	115.19	0.915	59.61	0.884	31.92	0.825	19.21	0.686
Iacf	114.13	0.923	59.12	0.891	31.09	0.847	18.23	0.723
Gs	158.17	0.988	79.89	0.978	40.28	0.970	20.44	0.956
Ga	156.92	0.996	78.95	0.990	40.08	0.975	20.79	0.940
IIs	232.54	0.981	122.35	0.932	64.78	0.880	35.98	0.792
IIa	229.49	0.994	116.67	0.977	62.11	0.918	36.63	0.778

TAB. 7.5 – *Temps et efficacité des mises en œuvre parallèles des méthodes de relaxation approchée sur le Tnode pour un problème de flot de dimension 96*

	2 processeurs		*4 processeurs*		*8 processeurs*		*16 processeurs*	
méthode	temps	efficacité	temps	efficacité	temps	efficacité	temps	efficacité
Is	368.79	0.915	192.82	0.875	101.64	0.830	55.41	0.761
Ia	364.73	0.925	185.18	0.911	95.73	0.881	52.73	0.800
Iacf	360.07	0.937	183.83	0.918	95.06	0.887	52.01	0.811
Gs	518.49	0.991	261.02	0.984	131.30	0.978	66.35	0.968
Ga	515.70	0.997	258.81	0.993	130.55	0.984	66.67	0.964
IIs	769.45	0.986	398.24	0.953	207.04	0.917	111.33	0.852
IIa	762.23	0.996	384.69	0.987	197.68	0.960	107.76	0.881

TAB. 7.6 – *Temps et efficacité des mises en œuvre parallèles des méthodes de relaxation approchée sur le Tnode pour un problème de flot de dimension 144*

actualisation d'une composante du vecteur itéré résulte d'un processus itératif dont le nombre d'itérations n'est pas connu a priori et peut être variable. Les Tableaux 7.3 à 7.6 mettent en évidence que la mise en œuvre asynchrone a accéléré de manière très efficace la méthode I. La mise en œuvre asynchrone a été plus performante que la mise en œuvre synchrone ; les temps d'inactivité dûs aux synchronisations ont été importants dans ce dernier cas en raison du déséquilibre indéterministe des charges de calcul. De plus, Iacf a été meilleure que les autres méthodes parallèles. Il semble donc qu'une meilleure interaction entre communication et calcul permette d'accroître l'efficacité des algorithmes parallèles. Par ailleurs, la première mise en œuvre asynchrone présentée au paragraphe 5.1.2 donne des résultats légèrement moins bons que ceux présentés ici et qui nous le rappelons correspondent à la deuxième mise en œuvre du paragraphe 5.1.3. La première mise en œuvre asynchrone avec communication flexible présentée au paragraphe 5.1.4, avec envoi tous les trois pas de calcul, donne des résultats très semblables à ceux présentés ici pour la deuxième mise en œuvre asynchrone avec communication flexible du paragraphe 5.1.5.

7.5 Effets des différentes architectures et bibliothèques

Dans ce paragraphe, nous présentons les résultats expérimentaux obtenus pour les différentes mises en œuvre parallèles de la méthode de relaxation approchée I, avec une précision $\epsilon = 10^{-4}$ pour le test $\left|g_i(x_i^{q'}; x)\right| \leq \epsilon$, sur un supercalculateur Cray T3E (pour lequel on a utilisé les bibliothèques de communication SHMEM de Cray et MPI), une machine Sparc 20 SMP quadriprocesseur, et un réseau de stations de travail en utilisant le logiciel Landa. Nous avons considéré un problème de flots turbulents avec des fonctions de coût satisfaisant (7.2).

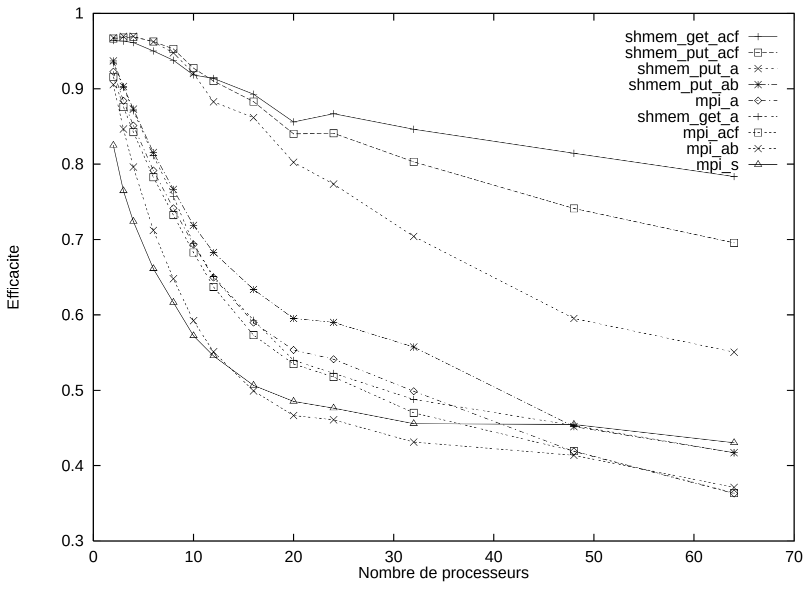


FIG. 7.1 – *Efficacité des mises en œuvre parallèles de l'algorithme de relaxation approchée I sur le Cray T3E pour les différentes fonctions des bibliothèques SHMEM et MPI et pour un problème de flot de dimension 192 avec des sommets de degré élevé*

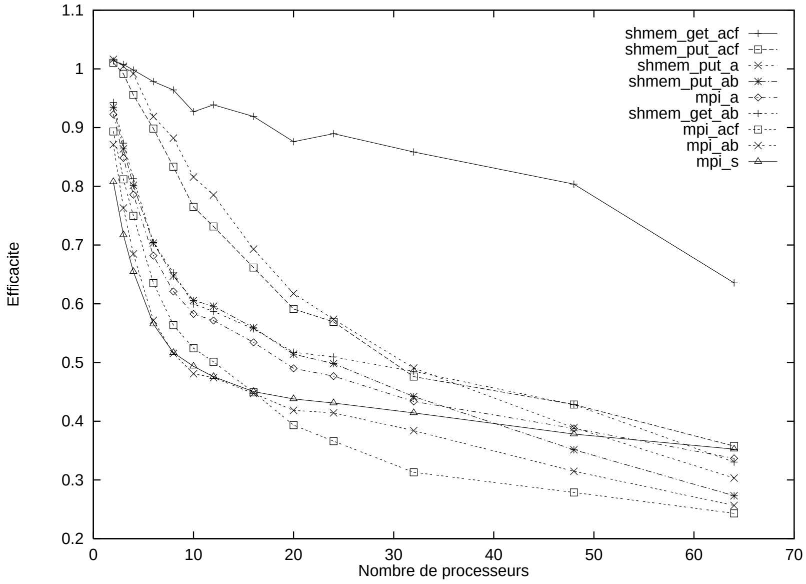


Fig. 7.2 – *Efficacité des mises en œuvre parallèles de l'algorithme de relaxation approchée I sur le Cray T3E pour les différentes fonctions des bibliothèques SHMEM et MPI et pour un problème de flot de dimension 96 avec des sommets de degré élevé*

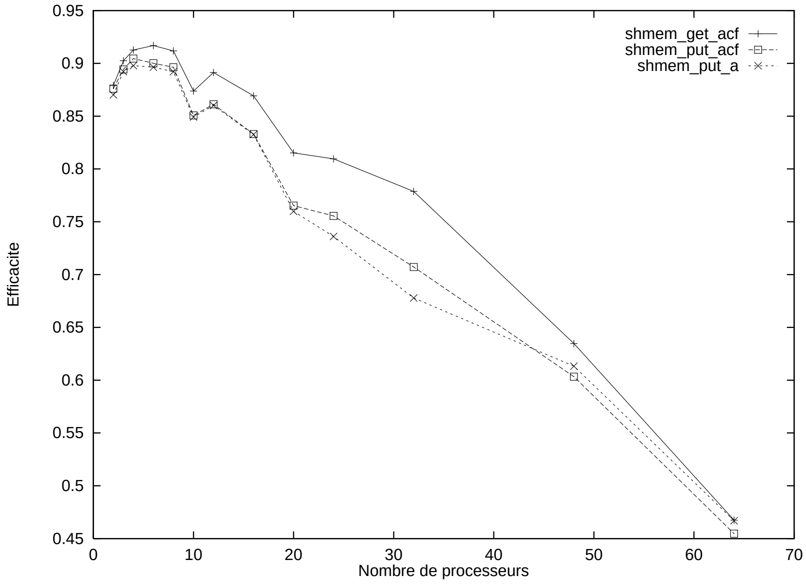


Fig. 7.3 – *Efficacité des mises en œuvre parallèles de l'algorithme de relaxation approchée I sur le Cray T3E pour les différentes fonctions de la bibliothèque SHMEM et pour un problème de flot de dimension 96 avec des sommets de degré faible*

Nous présentons tout d'abord sur les Figures 7.1 à 7.3, l'efficacité des différentes mises en œuvre de la méthode de relaxation approchée I sur le Cray T3E en fonction du nombre de processeurs pour trois problèmes différents. Les deux premiers problèmes possèdent respectivement 192 et 96 sommets et le degré maximum de chaque sommet est égal à 32. Le dernier problème possède 96 sommets et le degré maximum de chaque sommet est égal à 4. Nous avons précisé pour chaque courbe : la bibliothèque de communication utilisée, éventuellement la fonction de communication, et le type de mise en œuvre : synchrone (s), asynchrone (a), ou asynchrone avec communication flexible (acf) ; les mises en œuvre asynchrones pour lesquelles les échanges de données portent sur des blocs de composantes sont notées ab.

Les Figures 7.1 et 7.2 montrent que les performances des mises en œuvre utilisant la bibliothèque SHMEM de Cray sont généralement meilleures que celles qui utilisent la bibliothèque MPI. Notons que les mises en œuvre asynchrones avec communication flexible qui ont été présentées au paragraphe 5.2.3 et qui utilisent la bibliothèque SHMEM de Cray sont plus performantes que celles qui utilisent la bibliothèque MPI ; elles sont aussi sont plus performantes que les mises en œuvre asynchrones présentées au paragraphe 5.2.2 qui sont elles mêmes meilleures que les mises en œuvre synchrones présentées au paragraphe 5.2.1 et qui utilisent la bibliothèque MPI. Dans le cas asynchrone, on notera l'intérêt de communiquer les composantes une à une au lieu de transférer des blocs de composantes. Dans le cas asynchrone avec communication flexible, on remarque qu'il est plus intéressant d'accéder à des composantes en utilisant la fonction get de SHMEM que de communiquer régulièrement des itérés partiels avec la fonction put de SHMEM (la communication tous les 10 pas d'itérés partiels avec la fonction put a donné les meilleures performances). Pour conclure sur cette série d'expérimentations avec le Cray T3E, on note que la mise en œuvre asynchrone avec communication flexible est très performante, un gain de 25 à 40 % d'efficacité pouvant être obtenu par rapport à la meilleure mise en œuvre asynchrone ainsi que de 30 à 40 % par rapport à la mise en œuvre synchrone. Toutefois ces performances ne sont obtenues que pour une granularité non négligeable ; en effet dans le cas où le degré des sommet est plus petit, l'efficacité des mises en œuvre asynchrone avec communication flexible est plus faible et l'écart avec les mises en œuvre asynchrones utilisant la fonction put de la bibliothèque SHMEM tend à se réduire parfois de manière très importante. Ce type de problème n'a pas été rencontré sur le Tnode en raison de la faible puissance du processeur T800 (la granularité était toujours non négligeable pour les problèmes traités).

La Figure 7.4 présente l'efficacité des différentes mises en œuvre de la méthode de relaxation approchée I en fonction du nombre de processeurs sur la Sparc 20 SMP pour un problème avec 96 sommets et dont le degré maximum de chaque sommet est égal à 32 (les conditions d'expérimentation sont identiques à celles de la Figure 7.2). Nous avons précisé pour chaque courbe le type de mise en œuvre : synchrone (s), ou asynchrone avec communication flexible (acf).

On note que les très bonnes performances des mises en œuvre asynchrones avec communication flexible sont confirmées (cf. Figure 7.4). De manière générale, l'efficacité des mises en œuvre asynchrones avec communication flexible est plus grande que celle des mises en œuvre synchrone lorsque l'accès aux données est très rapide, c'est notamment le cas des machines de type mémoire partagée. C'est aussi le cas des supercalculateurs de type mémoire distribuée pour lesquels les temps de latence sont très faibles et la bande passante est élevée (cf. Figure 7.2). Globalement, les performances des diverses mises en œuvre sont excellentes pour la Sparc 20 SMP. La seule situation défavorable qui a été constatée est celle où la granularité

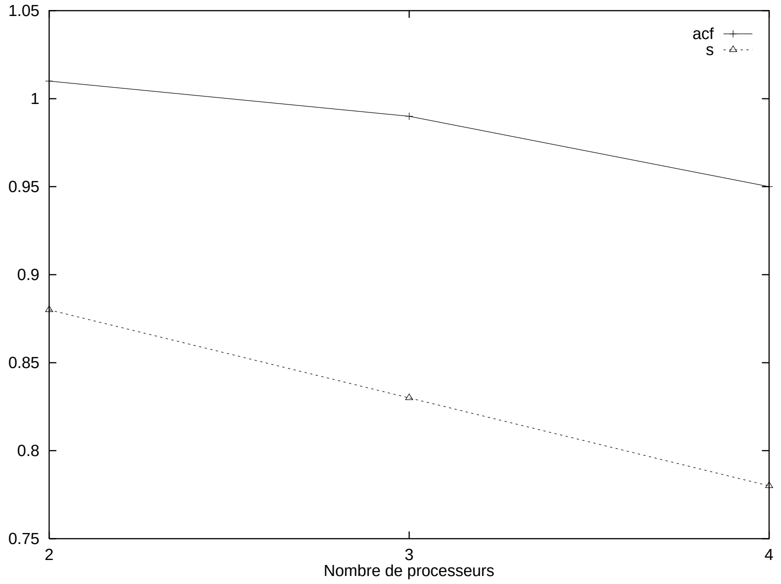


FIG. 7.4 – *Efficacité des mises en œuvre de I sur SMP*

est si fine que les temps de calcul sont de l'ordre des temps d'accès mémoire et de résolution des défauts de cache (cf. [Gaz98]) mais ce cas est vraiment un cas limite car les problèmes généralement traités ont une dimension relativement importante et la granularité est donc non négligeable.

La Figure 7.5 présente l'efficacité des différentes mises en œuvre de la méthode de relaxation approchée I en fonction du nombre de processeurs sur un réseau de stations de travail en utilisant le logiciel Landa. Nous avons considéré un problème de flots turbulents pour lequel le nombre de sommets est égal à 1600, des dimensions trop faibles ne permettant pas d'obtenir un résultat significatif. Cette remarque amène à considérer les performances médiocres des différentes mises en œuvre parallèles sur le réseau de stations qui sont essentiellement dues au temps de latence important des communications imposé par les différentes couches de logiciel ainsi que les caractéristiques du réseau. Dans ce cas, le temps de latence est si grand que l'on ne peut espérer une bonne efficacité que pour des problèmes de dimension considérable et qui présenteront une granularié non négligeable, ce que ne permettent pas toutes les applications. On note en particulier les très mauvaises performances de la mise en œuvre synchrone ainsi que les médiocres performances des mises en œuvre asynchrones. Dans le cas des réseaux de stations pour lequel le temps de latence est de l'ordre de la milliseconde, une mise en œuvre asynchrone avec communication flexible présente peu d'intérêt (cf. Figure 7.5).

De manière plus générale, il serait vain de comparer pour un même problème les performances des différentes mises en œuvre d'une méthode sur les diverses machines considérées ici ; en effet les caractéristiques des machines sont tellement différentes que les expérimentations n'auraient pas grand sens. Par exemple, dans le cas du Tnode la fréquence d'horloge du T800 et le temps de latence d'une communication sont égaux respectivement à 20 MHz et 7 μs, ces caractéristiques sont égales à 110 MHz et 1 ms pour un réseau de stations de travail ; en tenant compte des options offertes par les nouveaux compilateurs, la granularité

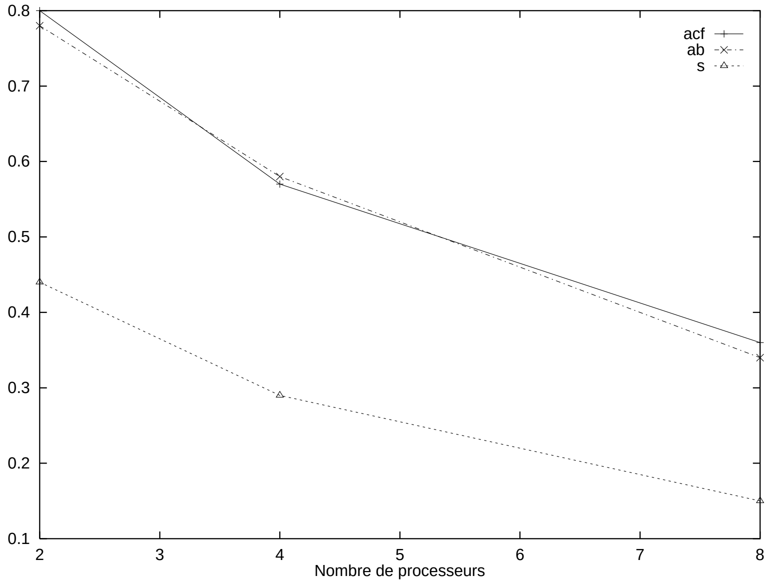


FIG. 7.5 – *Efficacité des mises en œuvre de I sur réseau de stations*

sur le Tnode est 1000 fois supérieure à la granularité sur le réseau de stations pour un même problème résolu par la même méthode parallèle sur ces deux architectures en supposant que le coût du temps de latence est prépondérant dans les communications.

7.6 Granularité et Equilibrage des tâches

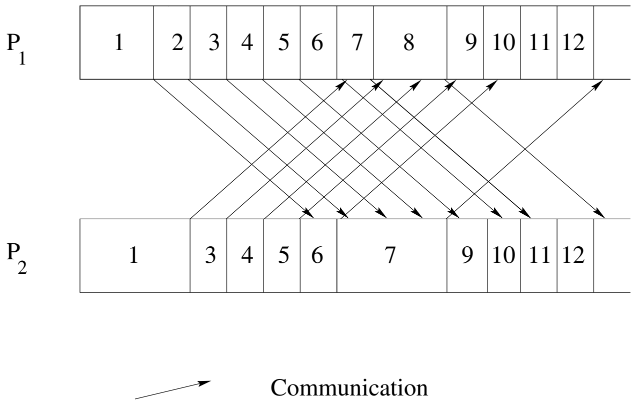
Pour un problème de dimension donnée, lorsque le nombre de processeurs augmente, le nombre d'échanges de données s'accroît et la granularité des tâches de calcul diminue. Toutefois en se basant sur les nombreuses expériences menées (cf. [EB96a], [EBSMG96], [EBGJ+98], [EBGMS96], [JEBG98], [EB96b], [MEBS98], et [GSMEB96]) et en particulier sur les résultats des Tableaux 7.3 à 7.6 on remarque que pour des expériences effectuées sur un nombre de processeurs différent mais avec une granularité identique, les efficacités sont très proches. Par exemple, le problème de dimension 48 résolu par 4 processeurs (cf. Tableau 7.3) présente des resultats en efficacité très voisins de ceux obtenus pour le problème de dimension 96 résolu par 8 processeurs (cf. Tableau 7.5). La granularité est donc un facteur prépondérant en matière de performance des algorithmes parallèles.

Les Tableaux 7.3 à 7.6, ainsi que les Figures 7.1 à 7.5 et les nombreuses expérimentations numériques citées au commencement de ce paragraphe permettent de tirer quelques conclusions générales.

On peut constater notamment que l'efficacité des mises en œuvre parallèles augmente avec la granularité ; elle peut même avoisiner 100 % pour des algorithmes asynchrones même dans des cas déséquilibrés. Les itérations asynchrones avec communication flexible ne peuvent être réellement efficaces que lorsque les communications sont très rapides, c'est à dire lorsque les temps de latence sont très brefs et la bande passante très grande ; dans ce cas la granularité est généralement élevée.

	16 processeurs	
méthode	temps	efficacité
Is	41.30	0.603
Ia	34.39	0.724
Iacf	32.67	0.762
Gs	46.62	0.806
Ga	40.95	0.918
IIs	80.03	0.690
IIa	67.94	0.813

TAB. 7.7 – *Temps et efficacité des mises en œuvre parallèles des méthodes de relaxation approchée sur le Tnode pour un problème de flot déséquilibré*



Les itérations 2,3,4,5,6,9,10,11,12 dupliquent inutilement les calculs et multiplient les communications. Seules les itérations 1,7 et 8 modifient les valeurs des composantes du vecteur itéré.

FIG. 7.6 – *Itérations asynchrones, cas d'une très faible granularité*

Dans le cas de charges de calcul déséquilibrées, l'efficacité des méthodes asynchrones est bien meilleure que celle des méthodes synchrones, de plus la première décroit bien moins rapidement que la seconde lorsque la granularité diminue. Pour des charges de calcul déséquilibrées de manière déterministe, le caractère pénalisant des temps d'attente dus aux synchronisations ainsi que les bonnes performances des mises en œuvre asynchrones apparaît nettement : dans le Tableau 7.7 nous donnons le temps de calcul et l'efficacité des méthodes étudiées au paragraphe 7.4 dans des conditions analogues d'expérimentation mais pour une répartition de manière non équilibrée de 120 sommets sur 16 processeurs : 18 sommets sont alloués aux processeurs impairs et 27 sommets sont alloués aux processeurs pairs. On remarque notamment que les mises en œuvre asynchrones sont de 10 à 15 % plus performantes que les mises en œuvre synchrones, On constate aussi que les itérations asynchrones avec communication flexible sont plus performantes que les algorithmes itératifs asynchrones.

Dans le cas de charges de calcul équilibrées, pour une granularité moyenne ou importante l'efficacité des algorithmes asynchrones est meilleure que celle des algorithmes synchrones. Lorsque la granularité diminue, l'efficacité des méthodes synchrones décroit généralement moins rapidement que l'efficacité de méthodes asynchrones, car l'asynchronisme peut entrainer des duplications de calcul.

Pour une granularité très faible, c'est à dire pour un temps de réactualisation très petit par rapport au temps d'une communication, les algorithmes itératifs asynchones présentent de piètres performances car les phases de calcul et les communications sont dupliquées inutilement ou du moins avec peu d'effet en raison de l'absence d'information fraiche (cf. Figure 7.6 où nous avons repris l'exemple simple considéré tout au long des paragraphes 2.1 et 2.3).

7.7 Spécificités des différents problèmes

Il ne faut pas perdre de vue que chaque application et par voie de conséquence chaque méthode de résolution a ses caractéristiques propres qui conditionnent la granularité, la fréquence des communications, ainsi que la quantité d'information transmise a l'issue de chaque réactualisation. Comme nous l'avons vu dans les paragraphes précédents, ces facteurs influencent grandement les performances des différentes mises en œuvre.

mise en œuvre	*problème de diffusion non linéaire*		*problème d'Hamilton-Jacobi-Bellman*	
	3 processeurs	*6 processeurs*	*3 processeurs*	*6 processeurs*
s	0.87	0.65	0.86	0.85
acf	0.99	0.81	0.90	0.90

TAB. 7.8 – *Problèmes aux limites discrétisés : efficacité des mises en œuvre parallèles sur IBM 3090-600*

Dans ce paragraphe nous présentons brièvement des résultats expérimentaux sur une machine à mémoire partagée IBM 3090-600 possédant 6 processeurs pour les deux problèmes aux limites du paragraphe 6.3. Nous avons considéré le problème de diffusion nonlinéaire discrétisé et les équations d'Hamilton-Jacobi-Bellman discrétisées et linéarisées. La dimension des problèmes traités correspond à 25000 points de discrétisation. La méthode alternée de Schwarz avec recouvrement a été utilisée ; le problème de diffusion nonlinéaire discrétisé a été résolu grâce à la méthode de Newton et le problème d'Hamilton-Jacobi-Bellman discrétisé et linéarisé a été résolu grâce à la méthode d'Howard-Mosco-Scarpini. L'efficacité des mises en œuvre synchrones et asynchrones avec communication flexible des différentes méthodes iteratives est présentée dans le Tableau 7.8.

On peut voir que les mises en œuvre asynchrones avec communication flexible sont plus efficaces que les mises en œuvre synchrones. De plus les performances des différentes mises en œuvre décroissent avec la granularité, la perte d'efficacité étant plus grande dans le cas synchrone. Pour plus de détails on pourra se reporter à [MEBS98] ainsi qu'à [SMEB95].

7.8 Conclusion

Dans ce chapitre nous avons étudié les performances des mises en œuvre des schémas itératifs synchrones, asynchrones et asynchrones avec communication flexible présentées au chapitre 5 sur diverses machines telles que le Cray T3E, le Tnode, la Sparc 20 SMP, et un réseau de stations de travail pour des applications très variées : problèmes d'optimisation et équations aux dérivées partielles discrétisées.

Le paramètre qui influe de manière prépondérante sur les performances des mises en œuvre parallèles semble être la granularité en effet il synthétise à lui seul des phénomènes très divers liés à des caractéristiques de l'architecture parallèle (on peut citer par exemple : la puissance des processeurs, le temps de latence des communications, et la bande passante) éventuellement à la bibliothèque de communication (comme par exemple le temps d'exécution d'une fonction spécifique d'envoi de message), mais aussi à l'application considérée et la méthode de résolution utilisée (le nombre d'opérations ou d'appels à des fonctions de calcul lors d'une phase élémentaire de réactualisation, ainsi que le nombre de pas de calcul si chaque phase de réactualisation nécessite de mettre en œuvre un processus itératif).

Il ressort des différentes expérimentations que pour une granularité moyenne ou grande les mises en œuvre sans synchronisation sont plus performantes que les mises en œuvre synchrones. De plus les performances diminuent en fonction de la granularité. Par contre pour des problèmes qui présentent certaines similarités et pour un nombre différent de processeurs on retrouve des performances très semblables pour des granularités identiques. Les performances des mises en œuvres asynchrones avec communication flexible sont aussi fonction de la granularité. En dessous d'une certaine granularité, les communications flexibles ne présentent guère d'intérêt, au delà elles peuvent entraîner un gain de performance très important dans le cas d'un grand nombre de processeurs.

L'équilibre ou le déséquilibre des charges de calcul est aussi un facteur qui conditionne les performances des différentes mises en œuvre parallèles. Cependant, si le déséquilibre des charges a de l'influence sur les mises en œuvre synchrones pour lesquelles il entraîne des temps d'attente importants et donc une perte d'efficacité notable, il a beaucoup moins d'impact sur les mises en œuvre asynchrones qui conservent de bonnes performances dans le cas d'un déséquilibre indéterministe ou déterministe des charges de calcul ; de plus les performances des mises en œuvre asynchrones décroissent moins rapidement en fonction de la granularité que celles des mises en œuvre synchrones. Le cas de charges de calcul déséquilibrées est celui où se manifeste le mieux l'intérêt d'une mise en œuvre asynchrone. Pour des charges de calcul équilibrées, les performances des mises en œuvre asynchrones qui nous le rappelons sont toujours meilleures que celles des mises en œuvre synchrones pour une granularité moyenne ou grande décroissent toutefois plus rapidement en fonction de la granularité que celles des mises en œuvre synchrones.

Chapitre 8

Conclusion Générale et Bilan Synthétique

> Nous retournons tous dans nos mains un vieux pneu vide grâce auquel nous voudrions parvenir au sens ultime que les paroles n'atteignent pas
> Italo Calvino
> Palomar

Depuis 1981, nous avons développé une recherche originale sur de nouvelles méthodes numériques : les méthodes itératives parallèles alors que les architectures de calcul parallèles ou distribuées étaient à leur balbutiements. Nous avons étudié en particulier des schémas de calcul typiquement parallèles sans équivalent dans le cas séquentiel : les itérations asynchrones. Nous nous sommes attachés à l'étude de ces algorithmes tant sur le plan théorique : étude de la modélisation, de la convergence, et de la terminaison, qu'expérimental : application à des problèmes très variés d'optimisation, de resolution de systèmes d'équations aux dérivées partielles, de systèmes Markoviens et plus récemment de commande optimale ; nous avons considéré la mise en œuvre de ces méthodes parallèles sur divers types d'architectures : machines à mémoire distribuée, Cray T3E, IBM SP2, Tnode, machines à mémoire partagée, Sun quadriprocesseur, et réseaux de stations de travail. Pour les problèmes ainsi que les machines citées ci-dessus, nous avons étudié les performances des algorithmes asynchrones et nous les avons comparées avec celles des schémas itératifs synchrones.

En ce qui concerne les aspects théoriques, nous avons établi la convergence de nombreux algorithmes itératifs asynchrones dans des contextes variés. Pour des systèmes d'équations non linéaires $Fx = z$, nous avons montré que lorsque F est continue, hors diagonale monotone décroissante et diagonale monotone strictement croissante, les algorithmes de relaxation asynchrone par point convergent de manière monotone vers une solution à partir d'une sur-solution ou d'une sous-solution. Ces résultats ont été par la suite étendus au cas où F est continue, hors diagonale monotone décroissante et diagonale monotone croissante. Nous avons aussi montré la convergence monotone d'algorithmes de type Richardson lorsque F présente la particularité d'être continue au sens de Lipschitz. Dans le cas particulier d'une classe importante de problèmes d'optimisation : les problèmes de flot convexes dans les réseaux (cette classe de problème est actuellement l'objet d'un intérêt particulier, notamment pour des applications au télécommunications) nous avons établi un résultat de convergence pour les algorithmes de relaxation asynchrone ainsi que pour les algorithmes de gradient asynchrones. Nous avons établi la convergence d'une nouvelle classe d'algorithmes itératifs totalement asynchrones permettant des communications flexibles entre processeurs par la prise en compte lors des calculs d'itérés partiels dans un contexte M-fonctions. En particulier nous avons considéré la résolution de systèmes d'équations non linéaires issus de la discrétisation d'équations aux dérivées partielles. Nous avons présenté un résultat de convergence monotone pour des algorithmes asynchrones par bloc avec communication flexible associés à des $\mathcal{A}$-sur applications ou à des $\mathcal{A}$-sous-applications. Ces résultats ont été étendus aux méthodes de multisplitting asynchrones avec communication flexible. La convergence des itérations asynchrones avec communication flexible a été aussi montrée pour des problèmes de flot convexes dans les réseaux, nous nous sommes plaçés dans un cadre plus général que celui des M-fonctions et nous avons considéré des applications de point fixe qui sont des sur applications ou des sous-applications.

Deux méthodes de terminaison pour les algorithmes asynchrones ont été proposées, La première consiste en un raffinement d'une méthode élaborée par Bertsekas et Tsitsiklis qui présente l'avantage de ne pas nécessiter l'acquittement de tous les messages envoyés, la seconde est basées sur l'utilisation du théorème de convergence asynchrone de Bertsekas et des ensembles de niveau.

Pour tous les problèmes traités et les différentes architectures considérées nous avons montré que les algorithmes asynchrones sont généralement plus performants que les versions synchrones. Ce résultat est particulièrement notable lorsque les charges de calcul sont déséquilibrées, que le temps de latence des communications est très court, et le taux de transfert important. Les expérimentations numériques ont mis en évidence que l'introduction de communications flexibles dans les schémas itératifs asynchrones permet d'accélerer la convergence de manière non négligeable lorsque le nombre de processeurs est moyen ou important. Nous avons aussi constaté que les algorithmes asynchrones sont généralement plus simple à mettre en œuvre que les algorithmes synchrones ; ceci est particulièrement vrai lorsque des fonctions de communication asynchrone ou des accès direct mémoire sont disponibles. Les expérimentations sur architectures à base de transputers ont démontré que lorsque des communications synchrones sont seules disponibles ont peut obtenir malgrè tout d'excellentes performances dans le cas asynchrone en utilisant au mieux les priorités et en mettant en œuvre un processus de type "buffer" qui permet de désynchroniser les processus de calcul. Dans ce cas, on rajoute une couche de logiciel qui permet réaliser une communication de type asynchrone.

Cette recherche sur l'asynchronisme, entreprise à notre initiative au LAAS du CNRS et sans contact avec d'autres laboratoires au commencement, nous l'avons menée tant sur le plan théorique que pratique avec persévérance et fidélité jusqu'à ce jour en essayant d'accroître les collaborations avec d'autre chercheurs français et étrangers; on peut citer notamment : les Professeurs Jean Claude Miellou de l'Université de Franche-Comté et Pierre Spiteri de l'ENSEEIHT-IRIT et les Professeurs Dimitri P. Bertsekas du MIT, David J. Evans du Loughborough University of Technology, Royaume Uni, et Andreas Frommer de l'Université de Wuppertal, Allemagne.

A ce jour, et malgré les difficultés pour accéder aux premières machines parallèles, les travaux publiés dans des revues scientifiques, les deux thèses, et trois D.E.A. encadrés, les contacts noués tant au sein de groupes de travail français tels que CAPA (Conception et Analyse d'Algorithmes Parallèles) du Programme de Recherche Coordonnée C3 puis P.R.S. (Parallélisme, Réseaux, et Systèmes Distribués), qu'au niveau de programmes internationaux tels que le projet européen Stimulation avec l'Université de Loughborough sont un encouragement à poursuivre sur ce thème de recherche prometteur. Les résultats sont synthétisés dans les tableaux ci-dessous.

revues internationales	9
revues nationales	3
ouvrages	3
conférences internationales	17

Tab. 8.1 – *Publications*

revues	nombre d'articles	années
SIAM	2	1987, 1990
Mathematics Of Computation	1	1998
Parallel Computing	2	1992, 1993
Computational Optimization and Applications	1	1996
Journal of Parallel and Distributed Computing	1	1996
Parallel Algorithms and Applications	1	1996
R.A.I.R.O. A.P.I.I.	1	1984
Calculateurs Parallèles	3	1989, 1996, 1998

Tab. 8.2 – *Noms des revues, nombre d'articles publiés et années*

conférences	nombre d'articles	années
IEEE CDC	3	1989, 1992, 1994
ECC	1	1991
ParCo	3	1991, 1993, 1997
HPCN	1	1995
ISCA PDCS	1	1996
ESA	1	1986

Tab. 8.3 – *Noms des principales conférences, nombre d'articles publiés et années*

Thèses de Doctorat	2	1991, 1998
D.E.A.	3	1992, 1996, 1997

Tab. 8.4 – *Encadrements*

Européen (Stimulation)	1	1990
Inter PRC (Stratagème)	1	1994
C3 (CAPA)	1	1993
ATP	2	1983, 1986
CNES	1	1984

TAB. 8.5 – *Travaux sur contrats*

Comité de Rédaction	1	Calculateurs Parallèles
Comité de Programme	2	CompEuro 1991, JNB97
Referee : revues	6	IEEE TPDS, Math. Prog. J.P.D.C., INFOR,...
Referee : conférences	6	ParCo93, Irregular95, 12th IFAC World Con., 11th ICDCS,...

TAB. 8.6 – *Responsabilités*

Cours (compl. de formation)	Syst. lin. syst. echant.	INSA Toulouse	1982-83
Cours (formation doctorale)	Intro. au parallélisme	LAAS-CNRS	1991-92
T.D. T.P.	Microinformatique	INSA Toulouse	1981-1983, 1985-1989

TAB. 8.7 – *Activités d'enseignement*

Chapitre 9

Perspectives de Recherche

Nous allions poursuivre ce succès, lorsque notre attention ayant été attirée vers la droite par un très grand tumulte, nous vîmes la plaine couverte de fuyards : c'était le moment où les chevaliers-gardes exécutaient leur vigoureuse charge. Le général Castex pensant alors qu'il ne serait pas sage d'avancer encore lorsque notre centre paraissait rétrograder en désordre, fit sonner le ralliement, et notre brigade s'arrêta.

Général Marbot
Mémoires

Lors de ces dernières années de nombreuses études novatrices ont été publiées sur les itérations asynchrones. R. Hiromoto, B. Wienke et R. Brickner [HWR92] du Los Alamos National Laboratory notamment ont étudié les performances des itérations asynchrones appliquées aux équations de transport linéarisées. Les expérimentations menées sur Denelcor HEP, Encore Multimax et hypercube Intel iPSC ont montré l'intérêt des schémas de calcul asynchrones.

On peut citer aussi les études effectuées par A. Frommer à l'Université de Wuppertal [FS94], [FS97b], [FS97a], et [FS98], ainsi que le professeur R. Bru de l'Université de Valence, [BMPS95], D. Szyld à Temple University, Philadelphie [Szy97], ou B. Zhong-Zhi [ZDE95] à l'Institute of Computational Mathematics and Scientific and Engineering Computing de Pékin. Ces travaux portent essentiellement sur la combinaison de schémas de calcul asynchrones avec des méthodes de Schwarz, des méthodes à deux niveaux (encore appelées : "two stage methods"), ou plus généralement des méthodes de multidécomposition, ou "multisplitting", afin de résoudre des grands systèmes. Cette approche est très prometteuse car elle permet de cumuler les gains de temps provenant des mises en œuvres asynchrones avec ceux des méthodes de multidécomposition qui sont réputées pour leurs performances. Notons la publication toute récente de J. Bahi, J. C. Miellou et K. Rhofir [BMR97] sur ce sujet qui donne une formulation générale des méthodes de multisplitting asynchrones.

Dans le domaine de l'optimisation les itérations asynchrones ont été combinées avec les méthodes d' ϵ-relaxation pour résoudre des problèmes de flot nonlinéaires par D. Bertsekas. P.

Beraldi, F. Guerriero de l'Université de Calabre [BF97] ont effectué plusieurs mises en œuvre de ces méthodes sur des clusters de processeurs Alpha ainsi que sur une machine Fujitsu AP1000.

On voit donc que la recherche sur les itérations asynchrones est un domaine très dynamique avec de nombreuses applications dans des secteurs très variés.

Nous comptons pour notre part valider les méthodes asynchrones de manière générale et les itérations asynchrones avec communication flexible en particulier par des tests sur de nombreuses machines parallèles à mémoire distribuées ou partagées et en comparant l'effet sur les performances des divers outils de communication tels que PVM, MPI2 ou la bibliothèque SHMEM de Cray par exemple. Cet aspect expérimental est particulièrement important si l'on veut assurer un impact industriel pour ce type de schéma de calcul parallèle. De manière plus précise, il est important d'évaluer quantitativement le seuil de granularité à partir duquel les mises en œuvre asynchrones sont plus efficace que les mises en œuvre synchrones. De plus il est intéressant de déterminer la granularité optimale pour des mises en œuvre asynchrones avec communication flexible, c'est à dire celle qui permet d'obtenir la meilleure efficacité. Nous allons aussi étudier les bénéfices d'un couplage entre l'utilisation de bibliothèques de processus légers comme Athapascan et de schémas de calcul asynchrones pour la résolution de problèmes d'équations aux dérivées partielles ou d'optimisation.

Nous appliquons actuellement les itérations asynchrones avec communication flexible à la résolution de certains problèmes de commande optimale dans un contexte M-fonction. Nous considérons en particulier la commande d'un four. Ce type de problème qui est de nature très différente de ceux traités par nous jusqu'à présent complète la gamme d'applications.

Nous comptons enfin développer les études sur la combinaison des techniques de multisplitting et des schémas asynchrones avec communication flexible pour la résolution de problèmes variés d'équations aux dérivées partielles.

De manière plus générale, la prise de décision parallèle asynchrone n'est pas une activité récente et limitée aux seuls calculateurs parallèles ou distribués, depuis toujours les activités humaines ont fait grand usage de l'asynchronisme que ce soit en matière d'activités militaires où la prise de décision au niveau tactique se fait sur la base de données partielles et locales généralement sans synchronisation globale des décisions en raison des contraintes de communication ou en ce qui concerne les opérations boursières pour lesquelles les opérateurs agissent pour leur compte ou pour un client sans synchronisation. Gageons que le concept d'asynchronisme conservera sa primauté au sein des activités humaines et que l'informatique du parallélisme lui donnera une nouvelle dimension.

Annexe A

Rappels Mathématiques

Dans cette Annexe nous avons rassemblé diverses définitions mathématiques et résultats qui sont utilisés au Chapitre 3.

A.1 Systèmes d'équations

Nous considérons tout d'abord les systèmes d'équations :

$$\mathcal{A}(x^*) = 0, \tag{A.1}$$

où $\mathcal{A}$ est une application de R^n dans R^n.

A.1.1 Rappels d'algèbre linéaire

Dans le cas linéaire ces systèmes seront aussi notés :

$$Ax^* = 0. \tag{A.2}$$

Définition A.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.2 : *L'ordre partiel naturel dans R^n, où encore ordre composante par composante est défini de la manière suivante :*

$$\textit{Pour tout } x, y \in R^n,\ x \leq y \textit{ si et seulement si } x_i \leq y_i,\ i = 1, \ldots, n. \tag{A.3}$$

Définition A.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 :

$$\langle x, y \rangle = \{z \in R^n \mid x \leq z \leq y\}. \tag{A.4}$$

On peut introduire un ordre partiel sur l'espace des matrices (n, m) analogue à celui défini dans l'équation (A.3). 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.4 : *Une matrice $A \in L(R^n, R^m)$ est non négative si*

$$A \geq 0.$$

Définition A.5 : *Toute Z-matrice N telle que l'inverse de N existe et est non négative est appelée une M-matrice.*

Définition A.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.1.2 M-fonctions

Définition A.7 : *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.8 : *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.9 : *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.10 : *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.11 : *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.12 : *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 $\lim_{j\to\infty}\ \mathcal{A}\left(x(j)\right) = +\infty$ 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 $\lim_{j\to\infty} \mathcal{A}\left(x(j)\right) = -\infty$.*

On peut caracteriser la surjectivité des M-fonctions au moyen de la coercivité pour l'ordre (cf [Rhe70]).

Théorème A.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 A.1 : *Soit $\mathcal{A}\ :\ R^n \to R^n$ une M-fonction continue surjective. Alors le système (A.1) possède une solution unique.*

Remarque A.2 : *Soit $n, \alpha \in N$, $\alpha \leq n$. Nous considérons la décomposition de R^n en $\prod_{i=1}^{\alpha} R^{n_i}$, $\sum_{i=1}^{\alpha} n_i = n$. Chaque sous-espace R^{n_i} est muni de l'ordre partiel naturel associé au cône $K_i = R_+^{n_i}$ des vecteurs de composantes non negatives dans R^{n_i}. Soit le vecteur $x \in R^n$ on a alors la décomposition suivante de x :*

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

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

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

Par la suite le vecteur $(x_1, \ldots, x_{i-1}, y_i, x_{i+1}, \ldots, x_\alpha)$ sera aussi noté $(y_i; x)$.
Si $\mathcal{A}\ :\ R^n \to R^n$ est une M-fonction continue surjective. Alors pour tout $i \in \{1, ..., \alpha\}$ et tout $x \in \prod_{i=1}^{\alpha} R^{n_i}$, l'application de R^{n_i} dans R^{n_i} qui associe $\mathcal{A}_i(y_i; x)$ au sous-vecteur y_i est

une M-fonction continue surjective sur R^{n_i} *(cf. [Rhe70]). De plus pour tout* $i \in \{1, ..., \alpha\}$*, le problème*

$$\mathcal{A}_i(\hat{x}_i; x) = 0, \tag{A.9}$$

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

$$F(x) = \hat{x} = \{\hat{x}_1, \ldots, \hat{x}_i, \ldots, \hat{x}_\alpha\}, \tag{A.10}$$

où $\hat{x}_i$ *est solution de (A.9) est bien définie. On a ainsi une formulation de type point fixe du problème (A.1) puisque*

$$F(x^*) = x^* \Leftrightarrow \mathcal{A}(x^*) = 0.$$

Théorème A.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.13 : *Un vecteur* $x \in R^n_+$ *est une* $\mathcal{A}$*-sur solution si* $\mathcal{A}(x) \geq 0$.

Définition A.14 : *Un vecteur* $x \in R^n_+$ *est une* $\mathcal{A}$*-sous-solution si* $\mathcal{A}(x) \leq 0$.

Définition A.15 : *Un vecteur* $x \in R^n$ *est une sur solution si* $F(x) \leq x$.

Définition A.16 : *Un vecteur* $x \in R^n$ *est une sous-solution si* $F(x) \geq x$.

Théorème A.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 A.3 : *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* $\mathcal{A}x \geq 0$.

Les concepts suivants qui ont été introduits dans [Mie86] jouent un rôle clé dans la définition des méthodes itératives parallèles asynchrones avec communication flexible.

Définition A.17 : *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 A.18 : *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 A.19 : *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 $F_i(x) \neq x_i$.*

Définition A.20 : *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 $F_i(x) \neq x_i$.*

Nous avons les résultats suivants (cf. [MEBS98]).

Théorème A.4 : *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 A.5 : *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 A.4 : *La réciproque des théorèmes A.4 et A.5 est en général fausse.*

Nous introduisons maintenant quelques propriétés se rapportant à la continuité.

Définition A.21 : *Une $\mathcal{A}$-sous-application ou une sous-application $\check{F}$ est continue pour l'ordre si*

$$x(j) \uparrow x^\star, j \to \infty \Rightarrow \check{F}_i(x(j)) \uparrow \check{F}_i(x^\star), j \to \infty, \text{ pour tout } i = 1, ..., \alpha, \tag{A.11}$$

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 A.22 : *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 A.23 : *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 A.24 : *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 A.25 : *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 A.26 : *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 A.27 : *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 A.28 : *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 A.29 : *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*

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

A.1.3 Notions d'applications H-accrétives

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 :

$$\Lambda^d(x) + \Lambda(x) \ni 0, \; x \in E. \tag{A.12}$$

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 A.5 : *Si l'on considère la décomposition du système (A.12) 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.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 :*

$$(\Lambda_l(x) - \Lambda_l(x'), g_l)_l \geq \sum_{i=1}^{\alpha} n_{li}\|x_l - x_l'\|_l.\|x_i - x_i'\|_i. \tag{A.13}$$

Définition A.30 : *Sous l'Hypothèse A.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.2 : *Pour tout $l \in \{1, ..., \alpha\}$, le domaine D_l est quasi-dense au sens de Kato (cf. [Kat64]).*

Hypothèse A.3 : *Pour tout $l \in \{1, ..., \alpha\}$ et $w \in D(\Lambda)$, l'application Λ_{ll}^w est hémi-continue sur D_l.*

Remarque A.6 : *L'Hypothèse A.2 traduit le fait que D_l est un ouvert dense dans E_l. Les Hypothèses A.2 et A.3 correspondent à des propriétés de régularité d'une part du domaine $D(\Lambda)$ et d'autre part, de l'application Λ.*

Théorème A.6 : *Sous les Hypothèses A.2 et A.3, l'Hypothèse A.1 est équivalente à l'ensemble des conditions suivantes.*

$$\forall l \in \{1, ..., \alpha\}, \forall w \in D(\Lambda), \forall x_l, x_l' \in D_l, \forall g_l \in G_l(x_l - x_l'), (\Lambda_{ll}^w(x_l) - \Lambda_{ll}^w(x_l'), g_l)_l \geq n_{ll}\|x_l - x_l'\|_l^2, \tag{A.14}$$

$$\forall l, i \in \{1, ..., \alpha\} \mid i \neq l, \forall w \in D(\Lambda), \forall x_i, x_i' \in D_i, \|\Lambda_{li}^w(x_i) - \Lambda_{li}^w(x_i')\|_l \leq -n_{li}\|x_i - x_i'\|_i. \tag{A.15}$$

Remarque A.7 : *Les conditions (A.14) et (A.15) 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.31 : *Sous les Hypothèses A.2, A.3, et A.1 si N est une minorante M-accrétive de l'application Λ on dira que Λ est H-accrétif.*

Définition A.32 : *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

$$\forall x_l, x_l' \in D(\Lambda_l^d), \forall \eta_l \in \Lambda_l^d(x_l), \forall \eta_l' \in \Lambda_l^d(x_l'), \exists g_l \in G_l(x_l - x_l') \mid (\eta_l - \eta_l', g_l)_l \geq 0. \tag{A.16}$$

A.2 Problèmes de point fixe

Soit E un espace de Banach. Nous considérons maintenant des problèmes de point fixe :

$$F(x^*) = x^*, \tag{A.17}$$

où F est une application de domaine $D(F) \in E$ dans E.

A.2.1 Applications contractantes en norme vectorielle

Définition A.33 : *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 :*

$$\psi(x) = (\|x_1\|_1, ..., \|x_\alpha\|_\alpha), \tag{A.18}$$

est une norme vectorielle canonique sur E.

Définition A.34 : *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.35 : *Soit $x^\star \in D(F)$ et T une matrice réelle de dimension (α, α). Si F et T vérifient :*

$$\psi(F(x^\star) - F(x)) \leq T\psi(x^\star - x), \ \forall x \in D(F), \tag{A.19}$$

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

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

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.36 : *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.17) (cf. [OR70]).

Théorème A.7 : *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 $x^* \in D(F)$.*

Les applications contractantes définies ci-dessus ont été introduites par Kantorovitch, Vulich, et Pinsker (cf. [KVP50]) et étudiées par Robert dans [Rob69], [RR73], et [Rob76] (cf. aussi [OR70]).

A.2.2 Applications contractantes en norme pondérée $\|.\|_T^\nu$

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 :

$$\mid w^\nu \mid_2 = 1 \text{ et } T w^\nu \leq \nu w^\nu, \tag{A.21}$$

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 :

$$\| x \|_T^w = \max_{i=1,\ldots,\alpha} \frac{\|x_i\|_i}{w_i^\nu}. \tag{A.22}$$

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

Théorème A.8 : *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 :*

$$\psi(F(x) - F(x')) \leq T\psi(x - x'), \tag{A.23}$$

on a alors :

$$\|F(x) - F(x')\|_T^w \leq \nu \|x - x'\|_T^w. \tag{A.24}$$

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 $\|, \|_T^w$.

Annexe B

Liste des publications

Thèse

1 [EB84] D. El Baz, Etude d'algorithmes itératifs de calcul parallèle. Application à la résolution distribuée du problème du routage optimal dans un réseau maillé à commutation de paquets. Thèse de Docteur Ingénieur de l'INSA de Toulouse, soutenue le 12 Janvier 1984 au LAAS du CNRS. Rapport LAAS 2985.

Composition du jury, G. Grateloup : président et directeur de thèse, G. Authie : directeur de thèse, D, J. Bernussou : examinateur, P. Bertrand : examinateur, P. Dorio : examinateur, J. C. Miellou : examinateur.

Revues Internationales avec Comité de Lecture

1 [BACEB84] J. Bernussou, G. Authie, J.L. Calvet, D. El Baz, Routage distribué dans les réseaux de communication, RAIRO Automatique, Vol. 18, No 2, p. 161-172, 1984.

2 [BEB87] D. P. Bertsekas et D. El Baz, Distributed asynchronous relaxation methods for convex network flow problems, SIAM Journal on Control and Optimization, Vol. 25, No 1, p. 74-85 , 1987.

3 [EB90] D. El Baz, M-functions and parallel asynchronous algorithms, SIAM Journal on Numerical Analysis, Vol. 27, No 1, p. 136-140, 1990.

4 [REB92b] C. Ribeiro and D. El Baz, A parallel optimal routing algorithm, Parallel Computing, Vol. 18, p. 1393-1402, 1992.

5 [EB93a] D. El Baz, Asynchronous implementation of relaxation and gradient algorithms for convex network flow problems, Parallel Computing, Vol.19, p. 1019-1028, 1993.

6 [EB96a] D. El Baz, Asynchronous gradient algorithms for a class of convex separable network flow problems, Computational Optimization and Applications, Vol.5, p. 187-205,

1996.

7 [EB96c] D. El Baz, A method of terminating asynchronous iterative algorithms on message passing systems, Parallel Algorithms and Applications, Vol. 9, No 1, p. 153-158, 1996.

8 [EBSMG96] D. El Baz, P. Spiteri, J.C. Miellou, D. Gazen, Asynchronous iterative algorithms with flexible communication for nonlinear network flow problems, Journal of Parallel and Distributed Computing Vol. 38, p. 1-15, 1996.

9 [MEBS98] J.C. Miellou, D. El Baz, P. Spiteri, A new class of asynchronous iterative methods with order intervals, Mathematics Of Computation, Vol. 67, No 221 p. 237-255, 1998.

Revues Nationales avec Comité de Lecture

1 [EB89b] D. El Baz, Mise en œuvre d'algorithmes itératifs distribués asynchrones sur un réseau de Transputers, Calculateurs Parallèles, Vol 1, No 3, p. 31-40, 1989.

2 [EBGMS96] D. El Baz, D. Gazen, J. C. Miellou, P. Spiteri, Mise en œuvre de méthodes iteratives asynchrones avec communication flexible I, application à la résolution d'une classe de problèmes d'optimisation, Calculateurs Parallèles, Vol. 8, No 4, p. 393-410, 1996.

3 [JEBG98] M. Jarraya, D. El Baz, D. Gazen, Mise en œuvre de méthodes iteratives asynchrones avec communication flexible II, implémentations sur Cray T3E, SMP, et réseau de stations, Journées Numeriques de Besançon 1997, Ballon d'Alsace, 23-25 Septembre 1997 à paraître dans la revue Calculateurs Parallèles.

Ouvrages

1 [GEB82] G. Authie and D. El Baz, A multimicroprocessor for parallel processing, in Parallel Processing Techniques For Simulation, edited by M. G. Singh, A. Y. Allidina and B. K. Daniels, Plenum Press, New York and London, p. 229-238, 1986.

2 [CBA+87] J. L. Calvet, J. Bernussou, G. Authie, D. El Baz, F. Le Gall, A. Titli, Décomposition, parallélisme, distribution : méthodes et structures, dans Derniers Développements en Automatique, Informatique , Robotique et Micro-électronique, Cepadues-Editions, Toulouse, p. 41-66, 1987.

3 [ea94] G. Authie et al. Optimisation parallèle, in Algorithmes Parallèles, Analyse et Conception, Authie et al. éditeurs, Hermes, p. 309-360, 1994.

Colloques avec actes et comité de lecture

1 [GBEB82] G. Authie, J. Bernussou, D. El Baz, Commande par algorithmes itératifs asynchrones distribués. Application au routage optimal, Proceedings of the IFAC Symposium Component and Instruments for Distributed Control Systems, Paris, 9-11 Décembre 1982, p. 49-58.

2 [EBA83] D. El Baz, G. Authie, Distributed algorithms for optimal routing in a packet-switched computer network. Shortest path and non linear flow problems. First International Workshop on methodologies and application of complex system theory, Le Caire, Novembre 15-17, 1983.

3 [GEB85] G. Authie et D. El Baz, Un multiprocesseur pour l'analyse et l'exploitation d'algorithmes de calcul parallèle, Congrès AFCET Automatique 85, Des Outils pour Demain, Toulouse, 23-25 Octobre 1985.

4 [BEB85] D. P. Bertsekas et D. El Baz, Distributed asynchronous relaxation methods for convex network flow problems, ORSA/TIMS 1985 Conference, November 4-6, 1985, Atlanta, USA.

5 [DEB86] Ph. Desroches et D. El Baz, Application of dynamic programming to station acquisition of a geostationary satellite, Proceedings of the Second International Symposium on Spacecraft Flight Dynamics, Darmstadt, FR Germany, 20-23 Octobre 1986, ESA SP-255, p. 135-139.

6 [EB89a] D. El Baz, A computational experience with distributed asynchronous iterative methods for convex network flow problems, Proceedings of the 28th IEEE Conference on Decision and Control, Tampa, U.S.A. 13-15 Décembre 1989, p. 590-591.

7 [EB91] D. El Baz, Asynchronous iterative algorithms for convex network flow problems, Proceedings of the European Control Conference 91, Grenoble, France, 2-5 Juillet 1991, p. 2397-2402.

8 [REB92a] C. Ribeiro, D. El Baz, A dual method for optimal routing in packet-switched networks, Lecture Notes in Control and Information Sciences, 180, 1992, Springer Verlag, Conference IFIP System Modelling and Optimization, Zurich p. 199-208, 1991.

9 [EB92] D. El Baz, Distributed asynchronous gradient algorithms for convex network flow problems, Proceedings of the 31st IEEE Conference on Decision and Control, Tucson, U.S.A., 16- 18 Décembre 1992, p. 1638-1642.

10 [EB93b] D. El Baz, Numerical Performances of parallel algorithms for a class of optimization problems, in Application of Supercomputers in Engineering III, editors C.A. Brebbia, H. Power, Elsevier Applied Science, p. 74-94, 1993.

11 [EB94a] D. El Baz, Nonlinear systems of equations and parallel asynchronous iterative

algorithms, Advances in Parallel Computing, 9, Parallel Computing: Trends and Applications, Parallel Computing 93, G. Joubert et al. editors North Holland, Amsterdam, p. 89-97, 1994.

12 [EB94b] D. El Baz, Parallel iterative algorithms for the solution of Markov systems, Proceedings of the 33rd IEEE Conference on Decision and Control, Orlando, U.S.A. 14-16 Décembre 1994, p. 2524-2527.

13 [GEB95] D. Gazen, D. El Baz, Efficient implementation of parallel algorithms for nonlinear network problems, Proceedings of the International Conference on High-Performance Computing and Networking HPCN 1995, Lecture Notes in Computer Science, 919, Bob Hertzberger and Giuseppe Serazzi editors, Springer, Milan, Italy, May 1995, p. 945-946.

14 [GSMEB96] R. Guivarch, P. Spiteri, J.C. Miellou, D. El Baz, Parallelisation of subdomains methods with overlapping for the solution of convection-diffusion problem, Poster Session, International Linear Algebra Year, Workshop Iterative Methods, Toulouse, 10-13 Juin 1996.

15 [EB96b] D. El Baz, An efficient termination method for asynchronous iterative algorithms on message passing architectures, Proceedings of the International Conference on Parallel and Distributed Computing Systems, Dijon, 25-27 Septembre 1996 Vol. 1, p. 1-7.

16 [EBSM97] D. El Baz, P. Spiteri, J.C. Miellou, Asynchronous multisplitting methods with flexible communication for pseudolinear P.D.E. Conférence Invitée, Proceedings of the Eighth International Colloquium on Differential Equations, Plovdiv, Bulgaria, 18-23 August 1997, p. 145-152.

17 [EBGJ+98] D. El Baz, D. Gazen, M. Jarraya, P. Spiteri, J.C. Miellou, Flexible communication for parallel asynchronous methods with application to a nonlinear optimization problem, Proceedings of ParCo 97, Bonn, Germany, 16-19 September 1997, in Advances in Parallel Computing: Fundamentals, Applications and New Directions, E. Dhollander, G. Joubert, F. Peters, and U. Trottenberg, editors, Elsevier Science B.V.,North Holland.

Annexe C

Thèses encadrées

C.1 Cassilda Ribeiro, boursière du gouvernement brésilien, 1991

Thèse de Doctorat de l'INSA de Toulouse intitulée : Une méthode parallèle pour le routage optimal dans les réseaux à commutation de paquets (cf. [Rib91]) et soutenue le 4 Octobre 1991 au LAAS du CNRS.

Composition du jury, B. Pradin : président, G. Authié : rapporteur, J.C. Miellou : rapporteur, J. Bernussou : examinateur, M. Dayde : examinateur, D. El Baz : directeur de thèse.

Cette étude a traité de la conception d'algorithmes parallèles pour le routage des données dans un réseau informatique à commutation de paquets. Les méthodes développées ont été conçues pour présenter un bon taux de convergence, ainsi qu'une bonne simplicité algorithmique et pour utiliser le parallélisme de façon effective. Une des originalités des méthodes proposées réside dans le fait que tous les produits sont traités simultanément alors qu'ils sont habituellement traités de manière cyclique, le problème multiflot étant alors ramené à la résolution d'une séquence de problèmes de flot à critère convexe. Les méthodes proposées sont des méthodes duales qui exploitent la décomposition par arcs du Lagrangien ; ce point étant particulièrement important pour la parallélisation. La minimisation du Lagrangien est effectuée par une méthode de Quasi-Newton et la fonction duale est maximisée par une méthode de Newton approchée. La décomposition du Lagrangien en Lagrangiens élémentaires associés à chaque arc et l'utilisation d'une méthode de Newton approchée basée sur un algorithme itératif par blocs donnant une direction de montée pour la fonction duale rendent la méthode duale bien adaptée à la mise en œuvre parallèle. Une approximation du Hessien de la fonction duale peut être aisément calculée à partir du Hessien de la fonction de Lagrange lequel est connu après minimisation du Lagrangien.

Deux versions de la méthode duale ont été proposées pour résoudre le problème du routage multiflot ; la deuxième méthode étant une version simplifiée ne retenant que les blocs diagonaux du Hessien. Dans ce dernier cas la réactualisation des variables duales se ramène à la résolution d'un ensemble de systèmes linéaires symétriques et indépendants résolus par la méthode de Choleski. Ces méthodes peuvent être utilisées, de façon plus générale, pour résoudre des problèmes non linéaires avec des fonctions objectif convexes et partiellement séparables.

La parallélisation des méthodes a été considérée, le modèle de programmation adopté étant de type SPMD. Le réseau a été partitionné en sous-régions et chacune des sous-régions est attribuée à l'un des processeurs de la machine parallèle. Le type de parallélisation qui a été choisi n'implique que des communications entre sous-régions voisines. Les aspects communication ont été étudiés en détail ainsi que les stratégies de partitionnement associées. Enfin des résultats experimentaux obtenus sur une machine multiprocesseurs T-NODE à mémoire distribuée ont été présentés et analysés. En particulier les bonnes performances en temps de calcul ont été mises en évidence et la bonne efficacité voisine de 0.6 des versions parallèles a été montrée. Cette dernière valeur étant voisine des efficacités habituellement présentées dans la littérature pour ce type de problème. On notera qu'il est difficile d'obtenir de meilleures efficacités en raison des problèmes déquilibrage de charge, le nombre d'itérations effectuées pour chaque arc par la méthode de quasi-Newton étant variable et dépendant fortement du point initial. Le parallélisme a permis de résoudre plus rapidement les problèmes et d'envisager la résolution de problèmes de plus grande taille.

C.2 Didier Gazen, boursier MESR, 1998

Thèse de Doctorat de l'Université Paul Sabatier de Toulouse intitulée : Contribution à l'étude d'algorithmes parallèles pour des problèmes d'optimisation de type flot dans les réseaux (cf. [Gaz98]) et soutenue le 30 Mars 1998 au LAAS du CNRS.

Composition du jury, J. C. Miellou : président, Brigitte Plateau : rapporteur, Catherine Roucairol : rapporteur, J. L. Calvet : examinateur, J. M. Garcia : examinateur, P. Spiteri : examinateur, D. El Baz : directeur de thèse.

Le travail de cette thèse a porté sur la conception d'algorithmes parallèles pour la résolution de deux classes de problèmes d'optimisation dans les graphes : les problèmes de flot de coût minimum à critère convexe et les problèmes de type flot maximum. Ces problèmes sont particulièrement d'actualités car on les retrouve notamment lors de la conception ou de la gestion de réseaux de communication. Le travail de thèse concerne également la mise en œuvre de ces algorithmes sur des machines parallèles à mémoire distribuée et à mémoire partagée.

La première partie de ce travail, traite des problèmes de flot à coût minimum et critère convexe. La théorie de la dualité conduit à la formulation d'un problème sans contrainte et différentiable. Les méthodes de gradient et de relaxation permettant de résoudre cette classe de problème sont performantes et bien adaptées à une mise en œuvre parallèle. Une attention particulière a été portée sur les méthodes itératives parallèles dépourvues d'un contrôle des itérations : les itérations asynchrones. Une nouvelle classe de méthodes : les itérations asynchrones avec communication flexible a été étudiée et mise en œuvre sur deux architectures parallèles : le T-Node (mémoire distribuée) ainsi qu'une machine Sun quadriprocesseur de type SMP (mémoire partagée). A cette occasion plusieurs stratégies de mise en œuvre ont été proposées. Les performances des différentes versions synchrones ou asynchrones des méthodes considérées ont été présentées et analysées en détail. Les excellentes performances des méthodes asynchrones ont été constatées et l'apport des communications flexibles a été attesté.

La seconde partie de cette étude a été consacrée au problème de flot maximum, qui est un cas particulier du problème de flot de coût minimum à critère linéaire (cf. [Ber91] et [AMO93]). Dans un premier temps, le problème a été présenté ainsi que les deux principales classes d'algorithmes séquentiels permettant de le résoudre: les algorithmes basés sur une chaîne améliorante et ceux basés sur la notion de préflot. Puis la comparaison des performances de ces algorithmes pour des problèmes de topologie différente a été effectuée à partir d'expérimentations numériques. A ce niveau, l'effet de nombreuses heuristiques pour améliorer la résolution a été étudié en détail. Pour finir, une stratégie de parallélisation du préflot par l'utilisation de processus légers sur architecture faiblement parallèle a été proposée.

Bibliographie

[AMO93] R. Ahuja, T. Magnati, and J. Orlin. *Network flows.* Prentice Hall, 1993.

[AR82] J. Abram and I. Rhodes. Some shortest path algorithms with decentralized informations and communication requirements. *IEEE Transactions on Automatic Control*, 27:570–582, 1982.

[Aut87] G. Authie. Contribution à l'optimisation de flots dans les réseaux, un multiprocesseur expérimental pour l'étude des itérations asynchrones. *Thèse de Doctorat d'Etat, UPS Toulouse*, 1987.

[BACEB84] J. Bernussou, G. Authie, J.L. Calvet, and D. El Baz. Routage distribué dans les réseaux de communication. *R.A.I.R.O. A.P.I.I.*, 18:161–172, 1984.

[Bah91] J. Bahi. Algorithmes asynchrones pour des systèmes différentiels algébriques, simulations numériques sur des exemples de circuits électriques. *Thèse de Doctorat de l'Université de Franche Comté*, 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.

[Bah98] J. Bahi. Algorithmes parallèles asynchrones pour des systèmes singuliers. *Note au CRAS série 1*, 326:1421–1425, 1998.

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

[BC90] D. P. Bertsekas and D. Castanon. Parallel asynchronous primal-dual methods for the minimum cost flow problem. *LIDS report MIT*, 1990.

[BC91] D. P. Bertsekas and D. Castanon. Parallel asynchronous implementation of the auction algorithm. *Parallel Computing*, 17:707–732, 1991.

[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.

[BD65] G. Birkhoff and J. Diaz. Nonlinear network problems. *Quart. Appl. Math.*, 13:431–443, 1965.

[BE82] R. Barlow and D. Evans. Synchronous and asynchronous iterative parallel algorithms for linear systems. *Comput. J.*, 25:56–60, 1982.

[BE87] D. P. Bertsekas and J. Eckstein. Distributed asynchronous relaxation methods for linear network flow problems. In *Proceedings of the 1987 IFAC Conference*, 1987.

[BEB85] D. P. Bertsekas and D. El Baz. Distributed asynchronous methods for network flow problems. In *ORSA/TIMS 1985 Conference*, Atlanta, USA, November 4-6, 1985.

[BEB87] D. P. Bertsekas and D. El Baz. Distributed asynchronous relaxation methods for convex network flow problems. *SIAM J. Control and Optimization*, 25:74–85, 1987.

[BEN88] R. Bru, L. Elsner, and M. Neumann. Models of parallel chaotic iteration methods. *Linear Algebra and its Applications*, 103:175–192, 1988.

[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*, 27:107–120, 1983.

[Ber85] D. P. Bertsekas. A distributed asynchronous algorithm for the assignment problem. In *Proceedings of the 24 th IEEE Conference on Decision and Control*, pages 1703–1704, 1985.

[Ber91] D. P. Bertsekas. *Linear Network Optimization.* MIT Press, 1991.

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

[BF97] P. Beraldi and Guerriero F. A parallel asynchronous implementation of the ϵ-relaxation method for the linear minimum cost flow problem. *Parallel Computing*, 23:1021–1044, 1997.

[BGM96a] J. Bahi, E. Griepentrog, and J.C. Miellou. Parallel treatment of a class of differential algebraic systems. *SIAM Journal on Numerical Analysis*, 23:1969–1980, 1996.

[BGM96b] D. P. Bertsekas, F. Guerriero, and R. Musmanno. Parallel asynchronous label correcting methods for shortest paths. *JOTA*, 88:297–320, 1996.

[BHT87] D. P. Bertsekas, P. Hossein, and P. Tseng. Relaxation methods for network flow problems with convex arc costs. *SIAM Journal on Control and Optimization*, 25:1219–1243, 1987.

[BKM91] A. Bhaya, E. Kaszkurewicz, and F. Mota. Asynchronous block-iterative methods for almost linear equations. *Linear Algebra and its Applications*, 154:487–508, 1991.

[BLGA87] J. Bernussou, F. Le Gall, and G. Authie. About some iterative synchronous and asynchronous methods for markov chain distribution computation. In *Proceedings of the 10-th IFAC World Congress*, 1987.

[BM93] J. Bahi and J.C. Miellou. Contractive mappings with maximum norms: comparisons of constant of contraction and application to asynchronous iterations. *Parallel Computing*, 19:511–523, 1993.

[BM97] J. Bahi and J.C. Miellou. Asynchronous multisplitting algorithms for differential-algebraic systems discretized by runge-kutta methods. In *Proceedings of the Eighth International Colloquium on Differential equations*, pages 35–41, Plovdiv, Roumanie, 18-23 August, 1997. VSP International Science Publishers, Utrecht.

[BMPS95] R. Bru, V. Migallon, J. Penadés, and D. Szyld. Parallel, synchronous and asynchronous two-stage multisplitting methods. *Electronic Transactions on Numerical Analysis*, 3:24–38, 1995.

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

[Bou88] L. Bougé. Sémantiques du parallélisme : un tour d'horizon. *Rapport de Recherche 88-6, Laboratoire d'Informatique Fondamentale d'Orléans, Université d'Orléans*, 1988.

[BP93] B. Beidas and G. Papavassilopoulos. Convergence analysis of asynchronous linear iterations with stochastic delays. *Parallel Computing*, 19:281–302, 1993.

[BP94] B. Beidas and G. Papavassilopoulos. Distributed asynchronous algorithms with stochastic delays for constrained optimization problems with conditions of time drift. In *Proceedings of 33rd Conference on Decision and Control*, pages 2663–2665, Lake Buena Vista, FL, USA, 1994.

[BPF89] L. Brochard, J.P. Prost, and F. Faurie. Synchronization and load unbalance effects of an iterative process on large scale multiprocessors. In *European symposium on high performance computing*, Montpellier, 1989.

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

[BT91] D. Bertsekas and J. Tsitsiklis. Parallel and distributed iterative algorithms: a selective survey. *Automatica*, 25:3–21, 1991.

[CBA+87] J. L. Calvet, J. Bernussou, G. Authie, D. El Baz, F. Le Gall, and A. Titli. Décomposition parallélisme distribution : méthodes et structures. In *Derniers Développements en Automatique, Informatique , Robotique et Micro-électronique*, pages 41–66. Cepadues-Editions Toulouse, 1987.

[CF94] M. Cosnard and P. Fraigniaud. Analysis of asynchronous polynomial root finding methods on a distributed memory multicomputer. *IEEE Transactions on Parallel and Distributed Systems*, 5:639–648, 1994.

[CGM+92] D. Conforti, L. Grandinetti, R. Musmanno, M Cannataro, G. Spezzano, and D. Talia. A model of efficient asynchronous parallel algorithms on multicomputer systems. *Parallel Computing*, 18:31–45, 1992.

[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.*, 2:199–222, 1969.

[Com76] P. Comte. *Algorithmes de relaxation-décentralisation.* Thèse de 3-ème cycle la Faculté des Sciences et des Techniques de l'Université de Franche-Comté, 1976.

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

[DEB86] Ph. Desroches and D. El Baz. Application of dynamic programming to station acquisition of a geostationary satellite. In *Proceedings of the Second International Symposium on Spacecraft Flight Dynamics*, pages 135–139, Darmstadt, Germany, 20-23 Octobre, 1986. ESA SP-255.

[DLM88] R. De Leone and O. L. Mangasarian. Asynchronous parallel successive over-relaxation for the symmetric linear complementarity problem. *Mathematical Programming , Serie B*, 42:347–361, 1988.

[Don71] J. D. P. Donnely. Periodic chaotic relaxation. *Linear Algebra Appl.*, 4:117–128, 1971.

[DS80] E.W. Dijkstra and C.S. Scholten. Termination detection for diffusing computation. *Information Processsing Letters*, 11:1–4, 1980.

[DZE97] W. Deren, B. Zhongzhi, and D. J. Evans. A class of asynchronous parallel matrix multisplitting relaxation methods. *Parallel Algorithms and Applications*, 12:173–192, 1997.

[ea94] Authie G. et al. *Optimisation parallèle.* Hermes, 1994.

[EB84] D. El Baz. Etude d'algorithmes itératifs de calcul parallèle, application à la résolution distribuée du problème du routage optimal dans un réseau maillé à commutation de paquets. *Thèse de Docteur Ingénieur, Rapport LAAS 2985, Toulouse, Janvier*, 1984.

[EB89a] D. El Baz. A computational experience with distributed asynchronous iterative methods for convex network flow problems. In *Proceedings of the 28 th IEEE Conference on Decision and Control*, pages 590–591, Tampa, U.S.A., 1989.

[EB89b] D. El Baz. Mise en œuvre d'algorithmes itératifs distribués asynchrones sur un réseau de transputers. *Calculateurs Parallèles*, 3:31–40, 1989.

[EB90] D. El Baz. M-functions and parallel asynchronous algorithms. *SIAM Journal on Numerical Analysis*, 27:136–140, 1990.

[EB91] D. El Baz. Asynchronous iterative algorithms for convex network flow problems. In *Proceedings of the first European Control Conference*, pages 2397–2402, Grenoble, France, 1991.

[EB92] D. El Baz. Distributed asynchronous gradient algorithms for convex network flow problems. In *Proceedings of the 31 st IEEE Conference on Decision and Control*, pages 1638–1642, Tucson, U.S.A., 1992.

[EB93a] D. El Baz. Asynchronous implementation of relaxation and gradient algorithms for convex network flow problems. *Parallel Computing*, 19:1019–1028, 1993.

[EB93b] D. El Baz. Numerical performances of parallel algorithms for a class of optimization problems. In *Applications of Supercomputers in Engineering III*, pages 79–94, Bath, United Kingdom, 1993.

[EB94a] 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.

[EB94b] 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.

[EB96a] D. El Baz. Asynchronous gradient algorithms for a class of convex separable problems. *Computational Optimization and Applications*, 5:187–205, 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.

[EB96c] D. El Baz. A method of terminating asynchronous iterative algorithms on message passing systems. *Parallel Algorithms and Applications*, 9:153–158, 1996.

[EBA83] D. El Baz and G. Authie. Distributed algorithms for optimal routing in a packet-switched computer network, shortest path and non linear flow problems. In *First International Workshop on Methodologies and Applications of Complex System Theory*, Le Caire, Egypte, Novembre 15-17, 1983.

[EBGJ+98] 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.

[EBGMS96] D. El Baz, D. Gazen, J.C. Miellou, and P. Spiteri. Mise en œuvre de méthodes itératives asynchrones avec communication flexible, application à la résolution d'une classe de problèmes d'optimisation. *Calculateurs Parallèles*, 8:393–410, 1996.

[EBSM97] 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.

[EBSMG96] 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.

[ED91] D. Evans and W. Deren. An asynchronous parallel algorithm for solving a class of nonlinear simultaneous equations. *Parallel Computing*, 17:165–180, 1991.

[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. *Num. Math.*, 39:325–340, 1982.

[ET84] M.N. El Tarazi. Algorithmes mixtes asynchrones, étude de la convergence monotone. *Num. Math.*, 44:363–369, 1984.

[FS94] A. Frommer and D. Szyld. Asynchronous two-stage iterative methods. *Numerische Mathematik*, 69:141–153, 1994.

[FS97a] 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.

[FS97b] A. Frommer and D. Szyld. Asynchronous weighted additive schwarz methods. *Electronic Transactions on Numerical Analysis*, 5:48–61, 1997.

[FS98] A. Frommer and D. Szyld. Asynchronous iterations with flexible communication for linear systems. *à paraître dans Calculateurs Parallèles*, 1998.

[Gaz98] D. Gazen. *Contribution à l'étude d'algorithmes parallèles pour les problèmes d'optimisation de type flot dans les réseaux.* Thèse de Doctorat en Informatique de l'Université Paul Sabatier, Rapport LAAS N 98116, 1998.

[GBEB82] Authie G., J. Bernussou, and D. El Baz. Distributed asynchronous iterative control algorithms, optimal routing applications. In *Proceedings of the IFAC Symposium Component and Instruments for Distributed Control Systems*, pages 49–58, Paris, France, 1982.

[GEB82] Authie G. and D. El Baz. *A multimicroprocessor for parallel processing.* Plenum Press, New York and London, 1982.

[GEB85] Authie G. and D. El Baz. Un multiprocesseur pour l'analyse et l'exploitation d'algorithmes de calcul parallèle. In A. Y. Allidina M. G. Singh and B. K. Daniels, editors, *Proceedings du Congrès AFCET Automatique 85, Des Outils pour Demain*, Toulouse, 23-25 Octobre, 1985. AFCET.

[GEB95] D. Gazen and D. El Baz. Efficient implementation of parallel algorithms for nonlinear network flow problems. In *Lecture Notes in Computer Science*, volume 919, pages 945–946, 1995.

[Gir91] L. Giraud. *Implantations parallèles de méthodes de sous-domaines synchrones et asynchrones pour la résolution de problèmes aux limites.* Thèse de Doctorat en Informatique, de l'Institut National Polytechnique de Toulouse, 1991.

[Gre89] A. Greenbaum. Synchronization costs on multiprocessors. *Parallel Computing*, 10:3–14, 1989.

[GS89] L. Giraud and P. Spiteri. Résolution parallèle des équations d'hamilton-jacobi-bellman sur un calculateur distribué. *rapport IRIT/ENSEEIHT, Janvier*, 1989.

[GS91] L. Giraud and P. Spiteri. Résolution parallèle de problèmes aux limites non linéaires. *M.2 A.N.*, 25:73–100, 1991.

[GS92] L. Giraud and 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, Amsterdam, 1992.

[GSB89] L. Giraud, P. Spiteri, and Ph. Berger. Parallel asynchronous and synchronous 2d poisson equation solvers on a processor network. In *Computational Mechanics Publications 5-th International Symposium on Numerical Methods in Engineering, Lausanne*, pages 265–271. Springer Verlag, 1989.

[GSMEB96] R. Guivarch, P. Spiteri, J.C. Miellou, and D. El Baz. Parallelization of subdomain methods with overlapping for the solution of convection-diffusion problem. In *Book of abstracts of the Workshop on Iterative Methods International Linear Algebra Year, Toulouse, 10-13 Juin*, pages 25–26, 1996.

[Gui97] R. Guivarch. *Résolution parallèle de problèmes aux limites couplés par des méthodes de sous-domaines synchrones et asynchrones.* Diplôme de Doctorat en Informatique de l'Institut National Polytechnique de Toulouse, 1997.

[Hoa85] C. Hoare. *Communicating Sequential processes.* Prentice Hall, 1985.

[HP94] A. Heddeya and K. Park. *Mapping parallel iterative algorithms onto workstation networks.* Boston University College report BU-CS-94-003, 1994.

[HWR92] R. Hiromoto, B. Wienke, and Brickner R. The performance of asynchronous iteration schemes applied to the linearized boltzmann transport equation. *Parallel Computing*, 18:241–268, 1992.

[Jac77] C. Jacquemard. *Contribution à l'étude d'algorithmes à convergence monotone.* Diplôme de Docteur en Mathématiques, Faculté des Sciences et Techniques de l'Université de Franche Comté, 1977.

[Jar97] M. Jarraya. *Mise en œuvre de méthodes itératives asynchrones avec communication flexible sur une machine parallèle Cray T3E, Application à la résolution* *d'une classe de problèmes d'optimisation.* Rapport de D.E.A. Automatique, Informatique Industrielle, LAAS du CNRS, 1997.

[JEBG98] M. Jarraya, D. El Baz, and D. Gazen. Mise en œuvre de méthodes itératives asynchrones avec communication flexible 2, implémentations sur cray t3e, smp et réseau de stations. *à paraître dans Calculateurs Parallèles*, 1998.

[Kat64] I. Kato. Demi-continuity, hemi-continuity and monotonicity. *Bull. Amer. Math. Soc.*, 70:548–550, 1964.

[KB90] E. Kaszkurewicz and A. Bhaya. On the convergence of parallel asynchronous block-iterative computations. *Linear Algebra and its Applications*, 131:139–160, 1990.

[Kun76] H.T. Kung. Synchronized and asynchronous parallel algorithms for multiprocessors. In J.F. Traub, editor, *Algorithms and Complexity: New Directions and Recent Results*, pages 153–200. Academic Press, New York, 1976.

[KVP50] L.V. Kantorovitch, B.Z. Vulich, and A.G. Pinsker. *Functional analysis in partially ordered spaces.* Moscow, 1950.

[KW84] C. P. Kruskal and A. Weiss. Allocating independent subtasks on parallel processors. In *Proc. international conference on parallel processing*, volume 10, pages 236–240, 1984.

[LAN96] *LANDA version 2.5.0 Manuel d'utilisation.* Delta partners S.A., 1996.

[Lao88] M. Laouar. Aspects de l'analyse numérique de méthodes itératives de point fixe : erreurs d'arrondi, accélération de la convergence, sous-domaines. *Thèse de Doctorat de l'Université de Franche Comté*, 1988.

[LB87] S. Li and T. Basar. Asymptotic agreement and convergence of asynchronous stochastic algorithms. *IEEE Trans. Auto. Contr.*, 32:612–618, 1987.

[LG92] F. Le Gall. Solving a bloc-tridiagonal markov system on parallel computers. In *advances in Parallel Computing 4*, pages 213–220. North Holland, 1992.

[LLMS79] F. Lhote, B. Lang, J.C. Miellou, and P. Spiteri. Relaxation methods for parallel in line calculations of the optimum control of large scale systems. In *Proceedings of the 9-th IFIP Conference, Varsovie*, pages 324–331, 1979.

[LM79] F. Lhote and J.C. Miellou. Algorithmes de décentralisation et de coordination par relaxation en commande optimale. In Titli et al., editor, *Analyse et commande des systèmes complexes*, pages 133–143. Cepadues Editions, 1979.

[LM86] B. Lubachevsky and D. Mitra. A chaotic asynchronous algorithm for computing the fixed point of nonnegative matrix of unit spectral radius. *J. Assoc. Comput. Mach.*, 33:130–150, 1986.

[LMS86] B. Lang, J. C. Miellou, and P. Spiteri. Asynchronous relaxation algorithms for optimal control problems. *Mathematics and Computers in Simulation*, 28:227–242, 1986.

[MCDB90] J. C. Miellou, Ph. Cortey-Dumont, and M. Boulbrachêne. Perturbation of fixed-point iterative methods. *Advances in Parallel Computing*, 1:81–122, 1990.

[ME89] J. C. Miellou and D. J. Evans. Stopping criteria for parallel asynchronous algorithms. *Computer Studies Report 503 Loughborough University of Technology*, 1989.

[MEBS98] J.C. Miellou, D. El Baz, and P. Spiteri. A new class of iterative algorithms with order intervals. *Mathematics of Computation*, 67:237–255, 1998.

[Mie75a] J. C. Miellou. Algorithmes de relaxation chaotique à retards. *R.A.I.R.O.*, 1:55–82, 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.

[Mit87] D. Mitra. Asynchronous relaxations for the numerical solution of differential equations by parallel processors. *SIAM J. Sci. Stat. Comput.*, 8:43–58, 1987.

[MS85a] J. C. Miellou and P. Spiteri. Two criteria for the convergence of asynchronous iterations. In P. Chenin et al., editor, *Computers and Computing*, pages 90–95. J. Wiley, 1985.

[MS85b] J. C. Miellou and P. Spiteri. Un critère de convergence pour des méthodes générales de point fixe. *R.A.I.R.O. M2AN*, 19:645–669, 1985.

[MT84] D. May and R. Taylor. Occam an overview. *Microprocessors and Microsystems*, 8:73–79, 1984.

[OR70] J.M. Ortega and W.C. Rheinboldt. *Iterative solution of Nonlinear Equations in Several Variables*. Academic Press, New York, 1970.

[Por69] T. Porshing. Jacobi and gauss-seidel methods for nonlinear network problems. *SIAM Journal on Numerical Analysis*, 6:437–449, 1969.

[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.

[REB92a] C. Ribeiro and D. El Baz. A dual method for optimal routing in packet-switched networks. In *Lecture Notes in Control and Information Sciences, 180*, pages 199–208. Springer Verlag, 1992.

[REB92b] C. Ribeiro and D. El Baz. A parallel optimal routing algorithm. *Parallel Computing*, 18:1393–1402, 1992.

[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.

[Rib91] C. Ribeiro. *Une méthode parallèle pour le routage dans les réseaux à commutation de paquets.* Thèse de Doctorat en Automatique de l'Institut National des Sciences Appliquées de Toulouse. Rapport LAAS N 91340, 1991.

[RKBM98] K. Rhofir, M. El Kyal, J. Bahi, and J.C. Miellou. Résolution d'un système différentiel algébrique raide par une méthode de multidécomposition asynchrone. *à paraître dans Calculateurs Parallèles*, 1998.

[Rob69] F. Robert. Blocs-h-matrices et convergence des méthodes iteratives classiques par blocs. *Linear Algebra and Applications*, 2:223–265, 1969.

[Rob76] F. Robert. Contraction en norme vectorielle: Convergence d'itérations chaotiques pour des équations non linéaires de point fixe à plusieurs variables. *Linear Algebra and Applications*, 13:19–35, 1976.

[Rob86] F. Robert. *Discrete Iterations, a metric study.* Springer Verlag, Berlin Heidelberg, New York, 1986.

[Rob89] F. Robert. Itérations chaotiques discrètes. *Rapport IMAG R.R.780-M*, 1989.

[Rob95] F. Robert. *Les Systèmes Dynamiques Discrets.* Springer Verlag, Berlin Heidelberg, New York, 1995.

[Roc70] R. Rockafellar. *Convex Analysis.* Princeton University Press, Princeton, N.J., 1970.

[Roc84] R. Rockafellar. *Network Flows and Monotropic Optimization.* John Wiley and Sons, New York, N.Y., 1984.

[Ros69] J. L. Rosenfeld. A case study in programming for parallel processors. *Communications of the ACM*, 12:645–655, 1969.

[RR73] F. Robert and M. Racle. Contraction faible en norme vectorielle. *Linear Algebra and Applications*, 6:305–335, 1973.

[SB96] S.A. Savari and D. P. Bertsekas. Finite termination of asynchronous iterative algorithms. *Parallel Computing*, 22:39–56, 1996.

[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.

[SGLM91] P. Spiteri, L. Giraud, A. Laouar, and J.C. Miellou. 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.

[SL88] P. Spiteri and B. Lang. Optimal control: Decomposition and coordination using asynchronous iteration. In *System and Control Encyclopedia*, pages 3475–3481. Pergamon Press, Oxford, 1988.

[SMEB95] P. Spiteri, J.C. Miellou, and D. El Baz. Asynchronous altenating schwarz method for the solution of nonlinear partial differential equations. *Rapport LAAS 95309*, page 37, 1995.

[Spi83] P. Spiteri. Simulation d'exécutions parallèles pour la résolution d'inéquations variationnelles stationnaires. *Revue E.D.F. Informatique et Mathématiques Appliquées, Série 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, 1986.

[Szy97] D. Szyld. Different models of parallel asynchronous iterations with overlapping blocks. *Temple University report 97-81*, 1997.

[TB86] J. Tsitsiklis and D. Bertsekas. Distributed asynchronous optimal routing in data networks. *IEEE Trans. Auto. Contr.*, 31:325–332, 1986.

[TB87] P. Tseng and D. Bertsekas. Relaxation methods for problems with strictly convex separable costs and linear constraints. *Mathematical Programming*, 38:303–321, 1987.

[TBA86] J. Tsitsiklis, D. Bertsekas, and M. Athans. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE Trans. Auto. Contr.*, 31:803–812, 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.

[Tsa89] W.K. Tsai. Convergence of gradient projection routing methods in an asynchronous stochastic quasi-static virtual circuit network. *IEEE Transactions on Automatic Control*, 34:20–33, 1989.

[Tse90] P. Tseng. Dual coordinate ascent for problems with strictly convex costs and linear constraints, a unified approach. *SIAM Journal on Control and Optimization*, 28:214–242, 1990.

[Tsi87] J. Tsitsiklis. On the stability of asynchronous iterative processes. *Math. Systems Theory*, 20:137–153, 1987.

[UD89] A. Uresin and D. Dubois. Sufficient conditions for the convergence of asynchronous iterations. *Parallel Computing*, 10:83–92, 1989.

[UD90] A. Uresin and D. Dubois. Parallel asynchronous algorithms for discrete data. *J.A.C.M.*, 37:588–606, 1990.

[Var62] R. Varga. *Matrix Iterative Analysis.* Prentice Hall, Englewood Cliffs, 1962.

[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. In *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.