Mise en œuvre de méthodes itératives asynchrones avec communication flexible. Application à la résolution d'une classe de problèmes d'optimisation

Didier EL BAZ *, Didier GAZEN *,
Jean Claude MIELLOU **, Pierre SPITERI ***

** LAAS du CNRS, L.P. CNRS 8001*
7 avenue du Colonel Roche
31077 Toulouse Cedex 4, France
E-mail : elbaz@laas.fr
*** Université de Franche Comté*
LCS, U.A. CNRS 40741
16, Route de Gray
25030 Besançon Cedex France
**** Institut National Polytechnique de Toulouse*
IRIT-ENSEEIHT, U.A. CNRS 1399, LIMA
2 rue Camichel
31071 Toulouse Cedex 4

RÉSUMÉ. *Dans cet article nous présentons une manière originale de mettre en œuvre une nouvelle classe de méthodes itératives parallèles asynchrones : les itérations asynchrones avec communication flexible. La caractéristique essentielle de cette nouvelle classe de méthodes est de permettre des communications très souples entre processeurs par échange d'itérés partiels ; son application à la résolution de problèmes de flot non linéaires dans les réseaux est considérée. Des résultats expérimentaux sur une machine parallèle à mémoire distribuée sont présentés et analysés.*

ABSTRACT. *An original implementation of a new class of parallel asynchronous iterative algorithms: asynchronous iterations with flexible communication is presented. Communication of partial updates is the main feature of this new class of methods. Application to convex network flow problems is considered. Computational results are presented and analyzed.*

MOTS-CLÉS : *méthodes itératives parallèles, itérations asynchrones, communication flexible, optimisation non linéaire, problèmes de flot convexes, réseaux.*

KEYWORDS: *parallel iterative methods, asynchronous iterations, flexible communication, nonlinear optimization, convex network flow problems*

1. Introduction

Une nouvelle classe de méthodes itératives asynchrones appelées itérations asynchrones avec communication flexible a été proposée récemment dans [MES 94]. Cette nouvelle classe de méthodes parallèles a été tout d'abord introduite dans un contexte faisant intervenir des M-fonctions; elle a été appliquée en particulier à la résolution d'équations aux dérivées partielles non linéaires dans [SME 95] ainsi qu'à la résolution d'une classe de problèmes d'optimisation: les problèmes de flot convexes dans les réseaux dans [ESM 95]. Dans cette étude nous considèrerons plus particulièrement la résolution de problèmes de flot convexes dans les réseaux. Les itérations asynchrones avec communication flexible autorisent plus d'échanges d'information entre processeurs que les méthodes itératives asynchrones classiques étudiés dans [Bau 78], [BeT 89], [ChM 69] et [Mie 75] ; contrairement à ces dernières méthodes pour lesquelles la valeur des composantes du vecteur itéré est communiquée seulement à la fin de chaque réactualisation, les itérations asynchrones avec communication flexible permettent la communication de la valeur courante des composantes du vecteur itéré lors des réactualisations. Ainsi les réactualisations peuvent être effectuées en utilisant des itérés partiels. Nous verrons dans cet article que l'utilisation d'itérés partiels permet d'accélérer la convergence.

L'originalité de cet article est de proposer une mise en œuvre très efficace des itérations asynchrones avec communication flexible sur une machine à mémoire distribuée; cette mise en œuvre repose notamment sur un principe différent de celui présenté dans [ESM 95].

Le paragraphe 2 traite des problèmes de flot dans les réseaux. Les itérations asynchrones avec communication flexible sont présentées au paragraphe 3. Une mise en œuvre originale est proposée au paragraphe 4. Le paragraphe 5 présente des résultats expérimentaux obtenus sur un multiprocesseur à mémoire distribuée Telmat Tnode 16-32.

2. 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. [ZeM 88]). Aussi la résolution de ces problèmes de manière parallèle semble t'elle particulièrement intéressante (cf. [BCE 94], [TsB 86] et [ZeL 88]).

2.1. *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 $c_{ij} : R \to (-\infty, +\infty]$, la fonction de coût associée à l'arc $(i, j) \in A$, c_{ij} est une fonction du flux f_{ij} dans l'arc (i, j). Soit b_i l'entrée ou la sortie 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 :

$$min \sum_{(i,j)\in A} c_{ij}(f_{ij}),\ \text{sous} \sum_{(i,j)\in A} f_{ij} - \sum_{(m,i)\in A} f_{mi} = b_i, \forall i \in N. \qquad [1]$$

Nous supposons que le problème [1] a une solution admissible et nous considérons les hypothèses suivantes sur les fonctions de coût c_{ij}.

Hypothèse 2.1 : c_{ij} est strictement convexe.
Hypothèse 2.2 : c_{ij} est semi-continue inférieurement.
Hypothèse 2.3 : la fonction convexe conjuguée de c_{ij}, définie par

$$c^*_{ij}(t_{ij}) = \sup_{f_{ij}}\{t_{ij}.f_{ij} - c_{ij}(f_{ij})\}, \qquad [2]$$

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

Les hypothèses 2.1 à 2.3 correspondent aux hypothèses générales effectuées dans [BeE 87]. L'hypothèse 2.3 implique que $\lim_{|f_{ij}|\to\infty} c_{ij}(f_{ij}) = +\infty$. Par conséquent les ensembles circonscrits par les courbes de niveau de la fonction critère du problème [1] sont tous bornés (cf. [Roc 70], chapitre 8). Il s'en suit que le problème [1] possède une solution optimale qui est unique en raison de l'hypothèse 2.1. Il résulte aussi de l'hypothèse 2.1 que les fonctions convexes conjuguées c^*_{ij} sont continûment différentiables et que leur gradient noté $\nabla c^*_{ij}(t_{ij})$, est l'unique f_{ij} qui atteint le supremum dans l'équation [2] (cf. [Roc 70] p. 218 et p. 253 et [BeE 87]). Notons enfin que ∇c^*_{ij}, qui est le gradient d'une fonction convexe différentiable est monotone non décroissant.

2.2. *Le problème dual*

Dans cet article, nous considérons plus particulièrement le dual du problème [1] :

$$\min_{p\in R^n} q(p),\ \text{sans constrainte sur le vecteur } p = \{p_i | i \in N\}, \qquad [3]$$

où q est la fonction duale définie par

$$q(p) = \sum_{(i,j)\in A} c^*_{ij}(p_i - p_j) - \sum_{i\in N} b_i.p_i. \qquad [4]$$

Le vecteur p est appelé vecteur prix. Le i-ème prix p_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, p) est donnée dans [Roc 70] : un vecteur de flot admissible $f = \{f_{ij} | (i, j) \in A\}$ est optimal pour [1] et un vecteur prix $p = \{p_i | i \in N\}$ est optimal pour [3] si et seulement si pour tout $(i, j) \in A, p_i - p_j$ est un sous-gradient de c_{ij} en f_{ij}. Une condition équivalente est : $f_{ij} = \nabla c^*_{ij}(p_i - p_j)$, pour tout $(i, j) \in A$.

2.3. *L'ensemble des solutions duales optimales*

L'existence d'une solution optimale pour le problème dual peut être garantie sous une hypothèse supplémentaire d'admissibilité régulière (cf. [Roc 84] p. 360 et p. 329 et [BeE 87]). 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 $p.$ 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 $P = \{p \in R^n | p_d = 0\}$ et le problème dual réduit :

$$min_{p \in P}\, q(p).$$

L'ensemble des solutions duales optimales réduit P^* est défini par : $P^* = \{p' \in P | q(p') = \min_p q(p)\}$. Par la suite nous utiliserons l'hypothèse supplémentaire suivante.

Hypothèse 2.4 : L'ensemble des solutions duales optimales réduit P^* est non vide et compact.

L'hypothèse 2.4 n'est pas très restrictive, le lecteur pourra se reporter à [BeE 87] pour de nombreux exemples. Il résulte de l'hypothèse 2.4 que le problème dual réduit admet une solution minimale et une solution maximale. C'est à dire qu'il existe $\underline{p}, \overline{p} \in P^*$ tel que $\underline{p} \leq p \leq \overline{p}$, pour tout $p \in P^*$, où $\underline{p} \leq p$ représente l'ordre partiel dans R^n (cf. [BeE 87], Proposition 2).

Nous notons $g(p)$ le gradient de la fonction duale. Il résulte de [4] que les composantes $g_i(p)$ de $g(p)$ sont données par :

$$g_i(p) = \frac{\partial q(p)}{\partial p_i} = \sum_{(i,j) \in A} \nabla c^*_{ij}(p_i - p_j) - \sum_{(m,i) \in A} \nabla c^*_{mi}(p_m - p_i) - b_i, i \in N. \quad [5]$$

On note que $g_i(p)$ est uniquement fonction de valeurs locales à savoir : les prix des sommets adjacents au sommet i. De plus, $g_i(p)$ est une fonction continue et monotone non décroissante de p_i, puisque $g_i(p)$ est la dérivée partielle d'une fonction convexe différentiable (cf. [TsB 87]). Par simplicité, le vecteur de R^n dont la i-ème composante est égale à $\hat{p}_i$ et la j-ème composante est égale à p_j pour tout $j \in N - \{i\}$ sera noté par la suite $(\hat{p}_i\,; p)$.

3. Méthodes itératives parallèles

Dans ce paragraphe, nous étudions diverses méthodes itératives parallèles pour la résolution du problème dual réduit, en particulier nous considérons les itérations asynchrones avec communication flexible.

3.1. *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 ces 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 [Ber 95], [BCE 94] et [BeT 89]. Notons aussi que cette méthode est particulièrement intéressante pour le problème traité ici en raison de la simplicité de sa mise en œuvre. Etant donné un vecteur prix $p \in P$, un sommet $i \in N - \{d\}$ est selectionné et son prix p_i prend une valeur $\hat{p}_i$ qui minimise le coût dual par rapport au i-ème prix, les autres prix étant inchangés (on a : $g_i(\hat{p}_i; p) = 0$). La méthode procède de manière cyclique en relaxant les prix de tous les sommets éléments de $N - \{d\}$ (cf. [BeE 87]). Considérons maintenant l'application multivoque $F_i, i \in N - \{d\}$, qui associe à tout vecteur prix $p \in P$, l'ensemble des prix $\hat{p}_i$ qui minimisent le coût dual par rapport au i-ème prix (c'est à dire : $F_i(p) = \{\hat{p}_i \in R | g_i(\hat{p}_i; p) = 0\}$). Il est bien connu qu'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. [Roc 70] p. 70). Par conséquent, il résulte de l'hypothèse 2.4 que les ensembles $F_i(p), p \in P, i \in N - \{d\}$, sont non vides et compacts. Il s'en suit que les applications minimales et maximales de relaxation $\underline{F}$ et $\overline{F}$, de composantes :

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

sont bien définies sur l'ensemble P (cf. [BeE 87]). Bertsekas et El Baz ont aussi montré dans [BeE 87] que $\underline{F}$ et $\overline{F}$ sont continues et monotone croissantes sur P.

Un avantage important de cette méthode est qu'elle se prête bien à une mise en œuvre parallèle. En particulier on remarque qu'on garde la propriété de décroissance du critère lorsqu'on réactualise simultanément un sous-ensemble de composantes indépendantes.

3.2. *Sous-applications et surapplications*

Les deux concepts suivants jouent un rôle clé dans la définition des méthodes itératives parallèles asynchrones avec communication flexible.

Définition 3.1 : Une application $\check{F}$, de composantes $\check{F}_i, i \in N - \{d\}$, est une sous-application associée à l'application de relaxation minimale $\underline{F}$ sur $P' = \{p \in P | p \leq \underline{p}\}$, on dit aussi par simplicité que $\check{F}$ est une sous-application sur P', si pour tout $i \in N - \{d\}$ et $p \in P'$ tel que $p_i \leq \underline{F}_i(p)$, on a $\check{F}_i(p) \in [p_i, \underline{F}_i(p)]$ et $\check{F}_i(p) \neq p_i$ si $\underline{F}_i(p) \neq p_i$.

Définition 3.2 : Une application $\hat{F}$, de composantes $\hat{F}_i, i \in N - \{d\}$, est une surapplication associée à l'application maximale de relaxation $\overline{F}$ sur $P" = \{p \in P | \overline{p} \leq p\}$, on dit aussi par simplicité que $\hat{F}$ est une surapplication sur $P"$, si pour tout $i \in N - \{d\}$ et $p \in P"$ tel que $\overline{F}_i(p) \leq p_i$, on a $\hat{F}_i(p) \in [\overline{F}_i(p), p_i]$ et $\hat{F}_i(p) \neq p_i$ si $\overline{F}_i(p) \neq p_i$.

Nous introduisons maintenant quelques propriétés se rapportant à la continuité de $\check{F}$ et $\hat{F}$.

Définition 3.3 : Une sous-application $\check{F}$ est continue pour l'ordre sur P' si

$$p(k) \uparrow p' \in P', k \to \infty \Rightarrow \check{F}_i(p(k)) \uparrow \check{F}_i(p'), k \to \infty, \text{ pour tout } i \in N - \{d\},$$

où la notation $p(k) \uparrow p', k \to \infty$, signifie que $p(0) \leq p(1) \leq ... \leq p(k) \leq p(k+1) \leq ... \leq p'$ et $\lim_{k\to\infty} p(k) = p'$.

De manière analogue on dira qu'une surapplication $\hat{F}$ est continue pour l'ordre sur P" si $p(k) \downarrow p' \in P$", $k \to \infty \Rightarrow \hat{F}_i(p(k)) \downarrow \hat{F}_i(p'), k \to \infty,$ pour tout $i \in N - \{d\}$, où la notation $p(k) \downarrow p', k \to \infty$, signifie que $p(0) \geq p(1) \geq ... \geq p(k) \geq p(k+1) \geq ... \geq p'$ et $\lim_{k\to\infty} p(k) = p'$.

Définition 3.4 : Une sous-application $\check{F}$ est minorée par une sous-application $\check{F}'$ continue pour l'ordre sur P', on dit aussi que $\check{F}$ est m-continue sur P' si pour tout $i \in N - \{d\}$ et $p \in P'$, tel que $p_i \leq \underline{F}_i(p)$, on a :

$$\check{F}'_i(p) \in [p_i, \check{F}_i(p)].$$

Définition 3.5 : Une surapplication $\hat{F}$ est majorée par une surapplication $\hat{F}'$ continue pour l'ordre sur P", on dit aussi que $\hat{F}$ est $\mathcal{M}$-continue sur P" si pour tout $i \in N - \{d\}$ et $p \in P$", tel que $p_i \geq \overline{F}_i(p)$, on a : $\hat{F}'_i(p) \in [\hat{F}_i(p), p_i]$.

3.3. *Exemples de sous-applications et de surapplications*

Nous présentons maintenant quelques sous-applications et surapplications. Nous considèrerons essentiellement dans cette étude des sous-applications et surapplications construites à partir de l'application de gradient. Les composantes F'_i de l'application de gradient F' sont définies par :

$$F'_i(p) = p_i - \frac{1}{\alpha} g_i(p), \text{ pour tout } i \in N - \{d\} \text{ et } p \in P.$$

où α est une constante positive. De manière claire F' est continue puisque le gradient de la fonction duale g est continu. Nous introduisons maintenant l'hypothèse suivante.

Hypothèse 3.1 : c_{ij} est fortement convexe de module $\frac{1}{\beta}$.

Il a été montré dans [Elb 96] que sous les hypothèses 2.1 à 2.4 et 3.1, il existe une constante $\alpha = \beta.\max_{i\in N} a_i$ (où a_i est le degré du sommet $i \in N$), telle que pour tout $p, p' \in P$ satisfaisant $p' \leq p$, on a : $g(p) - g(p') \leq \alpha.(p - p')$. Il en résulte que l'application de gradient F' est monotone croissante sur P pour $\alpha = \beta.\max_{i\in N} a_i$. On peut montrer également que F' est monotone croissante sur un sous-ensemble de P si c_{ij} est fortement convexe sur un sous-domaine associé (cf. [Elb 96]). De plus, nous avons montré dans [ESM 95] que l'application de gradient F' est alors une sous-application continue pour l'ordre sur P', une surapplication continue pour l'ordre sur $P"$ et que l'application $\check{F}$ définie de manière récursive par : $\check{F}_i(p) = p_i^{q'}$, pour tout $i \in N - \{d\}$ et $p \in P$, où q' est un entier positif, $p_i^q = F_i'(p_i^{q-1}; p), q = 1, ..., q'$ et $p_i^0 = p_i$ est une sous-application m-continue sur P'. Nous avons montré le même résultat pour l'application $\check{F}$ définie de manière récursive par : $\check{F}_i(p) = p_i^{q'}$, pour tout $i \in N - \{d\}$ et $p \in P$, où $p_i^q = F_i'(p_i^{q-1}; p), q = 1, ..., q', p_i^0 = p_i$, et q' est tel que $\left| g_i(p_i^{q'}; p) \right| \leq \epsilon$, ϵ étant une constante. Par la suite nous appellerons méthode de type gradient toute méthode itérative associée à ces deux dernières applications. On se rapportera à [ESM 95] pour des exemples similaires de surapplications.

3.4. *Itérations parallèles asynchrones avec communication flexible*

Les méthodes itératives associées aux sous-applications ou aux surapplications présentées au sous-paragraphe 3.3 peuvent être mises en œuvre de manière parallèle. En particulier, tous les prix peuvent être réactualisés simultanément par différents processeurs. Dans ce type de mise en œuvre parallèle les calculs sont effectués selon un certain ordre et des synchronisations sont requises. Puisque les temps d'attente dûs aux synchronisations peuvent être non négligeables, les performances des méthodes itératives parallèles peuvent être améliorées par une mise en œuvre asynchrone, où les calculs sont effectués concurremment sans ordre ni synchronisation. Les restrictions imposées aux calculs sont alors très faibles : aucune composante du vecteur itéré ne doit être abandonnée définitivement et les valeurs des prix associées à des itérés trop anciens doivent cesser d'être utilisées au fur et à mesure que les calculs progressent. Pour davantage de détails sur les méthodes itératives asynchrones on pourra se rapporter à [Bau 78], [BeT 89], [ChM 69], [Mie 75] et [MiS 85].

Nous présentons maintenant une nouvelle classe de méthodes itératives asynchrones : les itérations asynchrones avec communication flexible. Cette nouvelle classe a été tout d'abord proposée dans [MES 94].

Définition 3.6 : Soit T_i, le sous-ensemble infini des itérations auxquelles la i-ème composante du vecteur p est réactualisée tel que $T_i \cap T_j = \emptyset$, pour tout $i, j \in N - \{d\}, j \neq i$. Soit $\{\rho_i(k)\}$ la suite de numéros d'itération retardés satisfaisant pour tout $i \in N - \{d\}$, $0 \leq \rho_i(k) \leq k - 1$ avec $\rho_i(k) = k - 1$ si $k \in T_i$ et $\lim_{k \to \infty} \rho_i(k) = +\infty$. Une itération asynchrone avec communication flexible associée à $\check{F}$ une sous-application m-continue sur P' et au point initial $p^0 \in P'$ satisfaisant $p^0 \leq \underline{F}(p^0)$ est une suite $\{p^k\}$ de vecteurs de P telle que pour tout $i \in N - \{d\}$:

$$\begin{cases} p_i^k = \check{F}_i(\tilde{p}^k), & \text{si } k \in T_i, \\ p_i^k = p_i^{k-1}, & \text{si } k \notin T_i, \end{cases}$$

où $\tilde{p}^k$ est implicitement défini par

$$\tilde{p}^k \in \langle \max\{p^{\rho(k)}, \tilde{p}^q\}, p^{k-1} \rangle,$$

$\langle , \rangle$ étant un segment d'ordre dans P, le vecteur $p^{\rho(k)}$ ayant pour composantes $p_i^{\rho_i(k)}$, $i \in N - \{d\}$ et $q = \max\{l \in T_i | l < k\}$ (on pose $q = 0$ et $\tilde{p}^0 = p^0$, si $\{l \in T_i | l < k\} = \emptyset$).

On définit de manière analogue les méthodes itératives asynchrones avec communication flexible associées à une surapplication $\hat{F}$, $\mathcal{M}$-continue sur P" et au point initial $p^0 \in P$" tel que $p^0 \geq \overline{F}(p^0)$.

Afin de simplifier la présentation nous avons supposé que $T_i \cap T_j = \emptyset$ pour tout $i, j \in N - \{d\}, j \neq i$, cette hypothèse ne diminue en rien la portée de cette étude (cf. [ESM 95]).

Les itérations asynchrones avec communication flexible sont des méthodes générales pour lesquelles les itérations sont implémentées en parallèle sans ordre ni synchronisation sur un nombre de processeurs pouvant aller jusqu'à $n - 1$. L'originalité de cette nouvelle classe de méthodes provient des communications flexibles entre processeurs. En effet dans ce nouveau modèle $\tilde{p}_i^k$ n'est pas issu nécessairement d'une nouvelle réactualisation de p_i, il peut aussi correspondre à la valeur courante du prix p_i produite par quelques pas du processus construisant la sous-application $\check{F}$. Ainsi, $\tilde{p}_i^k$ peut provenir d'une réactualisation ou d'une réactualisation partielle. Il en résulte une meilleure interaction entre communication et calcul (cf. figure 2). Ceci se traduit dans le modèle par le fait que le vecteur $\tilde{p}^k$ utilisé lors de la réactualisation de p_i^k est inclus dans un segment d'ordre borné par en dessus par p^{k-1}.

Sous les hypothèses générales précédentes, nous avons montré la convergence des itérations asynchrones avec communication flexible associées à des sous-applications m-continues sur P' et des surapplications $\mathcal{M}$-continues sur P" dans [ESM 95].

4. Mise en œuvre des itérations asynchrones avec communication flexible

Dans ce paragraphe, nous proposons un algorithme permettant de mettre en œuvre les méthodes itératives asynchrones avec communication flexible. La version présentée est développée pour un multiprocesseur MIMD à mémoire distribuée suivant le modèle SPMD. Nous supposons que le graphe $G = (N, A)$ est partitionné en P sous-graphes où P correspond au nombre de processeurs utilisés. Chaque sous-graphe est attribué de manière statique à un processeur qui est alors chargé de mettre à jour un sous-ensemble de composantes du vecteur prix associées aux sommets du sous-graphe.

Soit $\mathcal{N}_k$ l'ensemble des sommets affectés au processeur $\mathcal{P}_k$ et Ω_i l'ensemble des sommets adjacents au sommet i. On appelle sommet *interne* tout $i \in \mathcal{N}_k$ tel que $\forall j \in \Omega_i$, $j \in \mathcal{N}_k$ et sommet *frontière* tout $i \in \mathcal{N}_k$ tel que $\exists j \in \Omega_i, j \notin \mathcal{N}_k$. Tout sommet frontière affecté à un processeur est relié à au moins un sommet frontière d'un autre processeur.

Comme énoncé au sous-paragraphe 2.3, il résulte de l'équations [5] que la mise à jour du prix p_i d'un sommet i nécessite la connaissance du prix de tous les sommets appartenant à Ω_i. Les processeurs vont donc devoir échanger le prix des sommets frontières. La technique employée dans l'algorithme repose sur l'émission de requêtes et l'envoi de la valeur du prix des sommets frontières en réponse à ces requêtes.

4.1. *Principe de l'algorithme*

L'algorithme général mis en œuvre sur chaque processeur est représenté par la figure 1. Le processeur qui détient le sommet d dont le prix est fixé à 0 (cf. sous-paragraphe 2.3) est noté $\mathcal{P}_d$. La boucle TANT QUE définit un processus itératif qui s'achève lorsque le processeur $\mathcal{P}_d$, après satisfaction de la condition locale au sommet d, transmet l'ordre d'arrêt à tous les processeurs. En effet pour cette application, on montre que l'on peut déduire de la condition locale de terminaison au sommet d une information sur l'état global du système. Pour de plus amples détails sur le test de terminaison on pourra se reporter à [ESM 95].

```
Terminé = FAUX;
Nombre_Iterations = 0;

TANT QUE Terminé == FAUX
  calcul_prix_sommets_frontières;
  calcul_prix_sommets_internes;
  Nombre_Iterations = Nombre_Iterations + 1;

  SI processeur ==  𝒫d
      Terminé = test_de_terminaison();
      SI Terminé == VRAI
            envoyer_signal_arrêt_des_processus;
      FINSI
  FINSI
FIN TANT QUE
```

FIG. 1 – *Algorithme général*

Les procédures de calcul de prix des sommets internes et des sommets frontières mettent en œuvre une méthode itérative de type gradient. La mise à jour du prix des sommets affectés à un même processeur s'effectue séquentiellement et ne fait intervenir que des données (prix) contenues dans la mémoire locale du processeur dans le cas des sommets internes. Cette procédure ne présentant aucune difficulté majeure, nous détaillons dans la suite du texte la procédure de calcul du prix des sommets frontières et plus précisément la technique adoptée pour les échanges de prix.

Calculer le prix p_i d'un sommet frontière i nécessite la prise en compte du prix des sommets frontières j présents sur d'autres processeurs. La procédure itérative de calcul du prix p_i débute après requête des prix voisins p_j 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 p_i. Si cette réponse survient plus tard, le calcul s'effectue avec la valeur de p_j 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 utilisons la politique suivante :

- un processeur $\mathcal{P}_1$ n'envoie une requête relative à un prix p_j que s'il a reçu une réponse à la requête précédente concernant p_j.
- tout processeur $\mathcal{P}_2$ se doit de répondre à une requête de $\mathcal{P}_1$ relative à un prix p_j :
  - soit immédiatement si le prix p_j a évolué depuis le précédent envoi de p_j à $\mathcal{P}_1$,
  - soit dès que le prix de p_j est modifié, c'est à dire en fin de réactualisation de p_j.

La figure 2 illustre par un diagramme temporel le principe de l'envoi et de la prise en compte des requêtes entre deux processeurs. Les sommets frontières i et j sont adjacents et appartiennent respectivement à $\mathcal{P}_1$ et $\mathcal{P}_2$. Pour éviter de surcharger le diagramme seuls sont représentés :

- les périodes de mise à jour de p_i et p_j,
- les requêtes issues de $\mathcal{P}_1$ pour le prix p_j,
- les envois du prix p_j par $\mathcal{P}_2$.

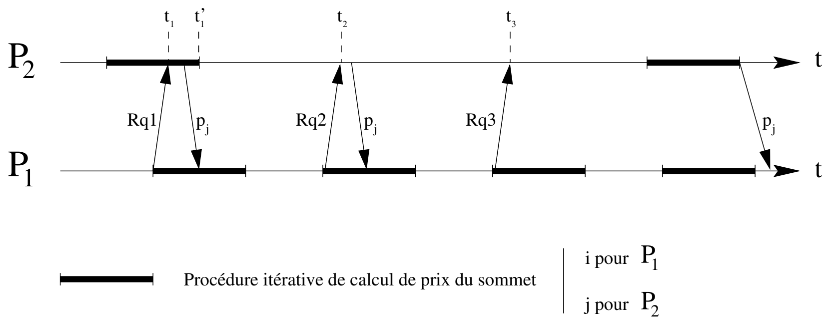

FIG. 2 – *Illustration de l'algorithme avec requêtes*

Nous pouvons vérifier que les réponses aux requêtes 1 et 2 sont "immédiates" car p_j est en cours de réactualisation à l'instant t_1 et $p_j(t_2) = p_j(t'_1) \neq p_j(t_1)$. De plus, on remarque que le prix $p_j(t_1)$ qui est pris en compte durant la mise à jour de p_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 $p_j(t_3) = p_j(t_2)$, il n'y aura donc pas d'autres requêtes pour p_j tant que le processeur $\mathcal{P}_1$ n'aura pas reçu de réponse à la requête 3.

4.2. *Mise en œuvre sur un multiprocesseur à mémoire distribuée*

L'algorithme a été programmé en langage 3LC sur une machine TNode 16/32 de Telmat équipée de 32 transputers T800. Elle offre entre autre 1Mo de mémoire locale par transputer et dispose de circuits crossbar C004 permettant de connecter les transputers suivant diverses topologies : pipeline, anneau, grille, hypercube, etc...

Dans cette étude nous avons considéré un réseau de processeurs organisé en *pipeline* avec liens bidirectionnels. On suppose par simplicité que deux sommets frontières adjacents sont affectés à des processeurs voisins ce qui rend inutile tout routage d'information.

La mise en œuvre de l'algorithme décrit au sous-paragraphe 4.1. met en jeu plusieurs processus au sein d'un même processeur :

– d'une part, un processus de *Calcul*, noté C, permettant la mise à jour du prix des sommets internes et des sommets frontières,

– d'autre part, plusieurs processus de communication :

  – un processus R de *Réception* de requête ou de prix émis par les processeurs voisins,
  – deux processus E_R d'*Envoi immédiat* de prix activés par le processus de Réception,
  – deux processus E_C d'*Envoi différé* de prix ou d'*Envoi* de requête, activés par le processus de Calcul.

On notera que les processeurs situés en extrémité du pipeline ne dialoguent qu'avec un processeur et ne disposent donc que d'une seule 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 3. Le processus de calcul C met à jour séquentiellement le prix des sommets qui lui ont été affectés à l'aide d'une méthode itérative de type gradient. Lors de la mise à jour du prix des sommets frontières, C peut avoir besoin de prendre connaissance du prix de sommets calculés par des processeurs voisins ou d'envoyer son prix en fin de réactualisation. Il active alors le processus E_C par émission d'un message sur le canal unidirectionnel L_i.

Le processus E_C de priorité haute interrompt immédiatement C et transmet une requête ou un prix 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

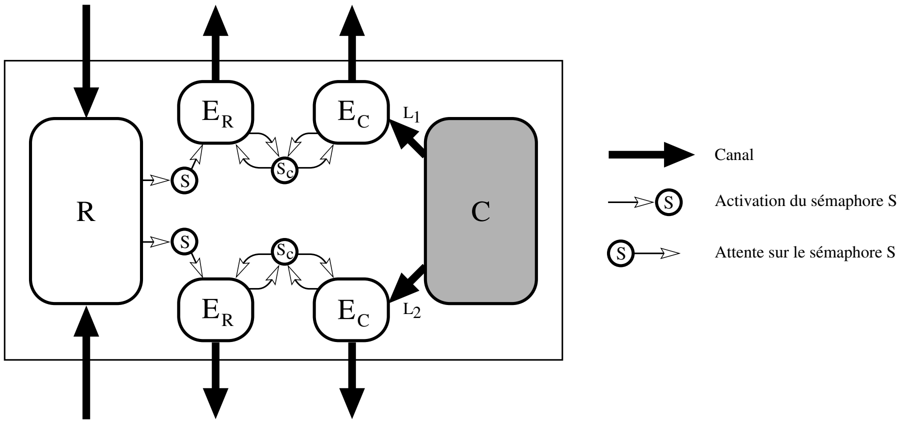

FIG. 3 – *les processus et leurs intéractions*

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 est un prix, il le stocke dans la zone mémoire correspondant aux prix frontières adjacents,

– si le message est une requête et que le prix n'a pas évolué depuis le précédent envoi, R la mémorise. Si le prix 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 le prix correspondant au processus de réception du processeur voisin concerné. L'utilisation d'un sémaphore plutôt que d'un canal entre R et E_R 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.

Remarques :

– 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

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.

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.

– 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).

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

5. Résultats expérimentaux

5.1. *Problèmes et méthodes*

Nous avons considéré des problèmes de flot dans des réseaux de distribution d'eau. Les fonctions de coût définies par : $c_{ij}(f_{ij}) = |f_{ij}|^{\frac{1+b}{b}}$ satisfont les hypothèses 2.1 à 2.4 et 3.1 sur un sous-domaine borné. Nous avons pris $b = 1.85$, ce cas correspond à des flots turbulents dans des canalisations (cf. [BiD 65], [Por 69] et [Rhe 70]) ; on a alors : $\nabla c_{ij}^*(p_i - p_j) = sign(p_i - p_j)\,|p_i - p_j|^{1.85}$. La topologie est celle d'un réseau maillé de faible degré (on a $\max_{i \in N} a_i = 4$). Le nombre de nœuds varie de 48 à 144 et le nombre d'arcs de 77 à 237. Pour chaque problème il y a trois entrées et trois sorties non nulles.

Nous avons considéré plusieurs méthodes itératives séquentielles associées à des sous-applications : une méthode de descente duale notée D proposée dans [ChZ 91] et basée sur une recherche unidirectionnelle inexacte, une méthode de gradient notée G et une méthode de type gradient notée TG (cf. sous-paragraphe 3.3) ; cette dernière méthode étant utilisée avec la précision $\epsilon = 10^{-2}$ sur le test $\left| g_i(p_i^{q'}; p) \right| \leq \epsilon$.

Les méthodes parallèles de descente duale, de gradient et de type gradient synchrones et asynchrones, ainsi que la méthode de type gradient asynchrone avec communication flexible ont été mises en œuvre sur la machine Tnode 16-32. Elles sont notées respectivement DS, DA, GS, GA, TGS, TGA et TGAF, où S, A et AF correspondent au type de mise en œuvre, respectivement synchrone, asynchrone et asynchrone avec communication flexible. Le nombre de processeurs est égal à 2, 4, 8 ou 16. Pour une implémentation détaillée des méthodes asynchrones de gradient, le lecteur pourra se reporter à [Elb 93].

Le pas des méthodes de gradient et de type gradient est 0.34. Pour toutes les méthodes, la précision du test de terminaison est de 0.1 ; nous avons montré dans [ESM 95] que la somme des valeurs absolues des déficits au sommets du réseau est alors inférieure à 0.1. Le point initial $p^0 \in P'$ satisfait $p_i^0 \leq \underline{F}_i(p^0)$ pour tout $i \in N - \{d\}$. Pour tous les problèmes et méthodes sauf le problème de taille 120 avec 16 processeurs nous avons équilibré le nombre de sommets sur les différents processeurs. De plus, à nombre de processeurs égal, le partitionnement de chaque problème est identique pour les différentes méthodes considérées.

5.2. *Résultats*

Le tableau 1 donne le temps en secondes des méthodes séquentielles de descente duale (D), de gradient (G) et de type gradient (TG) en fonction du nombre de nœuds dans le réseau. Le temps et l'efficacité des différentes méthodes parallèles sont donnés dans les tableaux 2 à 6 en utilisant la définition classique de l'efficacité : $e = \frac{1}{p}.\frac{t_1}{t_p}$, où t_1 représente le temps de calcul obtenu en utilisant un processeur et t_p le temps de calcul obtenu en utilisant p processeurs.

méthode / taille	D	G	TG
48	58.92	41.73	31.53
72	194.71	135.11	94.70
96	456.02	312.65	210.77
120	883.40	601.28	398.56
144	1518.07	1027.81	674.89

Tableau 1 : temps des méthodes séquentielles

	2 processeurs		*4 processeurs*		*8 processeurs*	
méthode	temps	efficacité	temps	efficacité	temps	efficacité
DS	30.67	0.961	16.93	0.870	9.44	0.780
DA	29.98	0.983	15.99	0.921	9.54	0.772
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
TGS	17.91	0.880	10.20	0.773	5.85	0.674
TGA	17.24	0.914	9.40	0.839	5.74	0.687
TGAF	17.16	0.919	9.19	0.858	5.42	0.727

Tableau 2 : temps et efficacité des méthodes parallèles pour un problème de taille 48

	2 processeurs		*4 processeurs*		*8 processeurs*	
méthode	temps	efficacité	temps	efficacité	temps	efficacité
DS	99.99	0.974	53.44	0.911	28.82	0.845
DA	98.30	0.990	50.56	0.963	27.83	0.875
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
TGS	53.81	0.880	29.32	0.808	16.13	0.734
TGA	52.09	0.909	27.21	0.870	15.15	0.781
TGAF	51.59	0.918	26.93	0.879	14.56	0.813

Tableau 3 : temps et efficacité des méthodes parallèles pour un problème de taille 72

méthode	2 processeurs		4 processeurs		8 processeurs		16 processeurs	
	temps	efficacité	temps	efficacité	temps	efficacité	temps	efficacité
DS	232.54	0.981	122.35	0.932	64.78	0.880	35.98	0.792
DA	229.49	0.994	116.67	0.977	62.11	0.918	36.63	0.778
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
TGS	117.75	0.895	62.92	0.838	34.00	0.775	19.46	0.677
TGA	115.19	0.915	59.61	0.884	31.92	0.825	19.21	0.686
TGAF	114.13	0.923	59.12	0.891	31.09	0.847	18.23	0.723

Tableau 4 : temps et efficacité des méthodes parallèles pour un problème de taille 96

méthode	2 processeurs		4 processeurs		8 processeurs		16 processeurs	
	temps	efficacité	temps	efficacité	temps	efficacité	temps	efficacité
DS	448.82	0.984	233.82	0.945	122.45	0.902	80.03	0.690
DA	443.85	0.995	224.76	0.983	117.05	0.943	67.94	0.813
GS	303.65	0.990	153.07	0.982	77.06	0.975	46.62	0.806
GA	301.74	0.996	151.53	0.992	76.57	0.982	40.95	0.918
TGS	220.20	0.905	116.20	0.858	62.08	0.803	41.30	0.603
TGA	216.60	0.920	110.31	0.903	58.09	0.858	34.39	0.724
TGAF	214.06	0.930	109.83	0.907	57.46	0.867	32.67	0.762

Tableau 5 : temps et efficacité des méthodes parallèles pour un problème de taille 120

méthode	2 processeurs		4 processeurs		8 processeurs		16 processeurs	
	temps	efficacité	temps	efficacité	temps	efficacité	temps	efficacité
DS	769.45	0.986	398.24	0.953	207.04	0.917	111.33	0.852
DA	762.23	0.996	384.69	0.987	197.68	0.960	107.76	0.881
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
TGS	368.79	0.915	192.82	0.875	101.64	0.830	55.41	0.761
TGA	364.73	0.925	185.18	0.911	95.73	0.881	52.73	0.800
TGAF	360.07	0.937	183.83	0.918	95.06	0.887	52.01	0.811

Tableau 6 : temps et efficacité des méthodes parallèles pour un problème de taille 144

5.3. *Analyse des résultats*

Le tableau 1 indique clairement que la méthode TG a été plus rapide que G et D. Notons que D requiert plus de calculs que G à chaque itération.

Pour les méthodes parallèles de gradient, 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 2 à 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 du partitionnement équitable des sommets du réseau ; 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 méthodes parallèles de descente duale, les charges de calcul sont aussi 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 2 à 6 indiquent qu'une mise en œuvre asynchrone a accéléré de manière très efficace la méthode de descente duale. Les méthodes de descente duale asynchrones ont été plus rapides que les méthodes de descente duale synchrones pour une granularité suffisamment grande. Les bonnes performances des méthodes de descente duale synchrones sont dûes essentiellement au partitionnement équitable des sommets du réseau.

Les méthodes parallèles de type gradient sont caractérisées par un non équilibrage indéterministe des charges de calcul puisque la réactualisation d'un prix résulte d'un processus itératif. Les tableaux 2 à 6 indiquent qu'une mise en œuvre asynchrone a accéléré de manière très efficace la méthode de type gradient. Les méthodes asynchrones de type gradient ont été plus rapides que les méthodes synchrones de type gradient ; les temps d'inactivité dûs aux synchronisations ont été importants pour cette dernière méthode en raison du non équilibrage indéterministe des charges de calcul. De plus les méthodes de type gradient asynchrones avec communication flexible ont été plus rapides que toutes les autres méthodes. En particulier on note que TGS, TGA et TGAF ne différaient que par le type de mise en œuvre qui était respectivement synchrone, asynchrone et asynchrone avec communication flexible. Il semble donc qu'une meilleure interaction entre communication et calcul permette d'accroître l'efficacité des algorithmes parallèles. On remarque aussi que dans un cas non équilibré comme le problème avec 120 sommets et 16 processeurs, les temps d'attente dûs aux synchronisations sont plus longs ; le tableau 5 indique que les méthodes asynchrones sont alors très performantes.

Références

[Bau 78] G. M. Baudet, *Asynchronous iterative methods for multiprocessors*, J. Assoc. Comput. Mach., 2 (1978), 226-244.

[Ber 95] D. P. Bertsekas, *Nonlinear Programming*, Athena Scientific, Belmont, MA, 1995.

[BCE 94] D. P. Bertsekas D. A. Castañon, J. Eckstein, et S. Zenios, *Parallel computing in network optimization*, report LIDS-P-2236, Department of Electrical Engineering and Computer Science, M.I.T., Cambridge, MA, 1994, Handbook on Operation Research and Management Science, Elsevier New York, 1995 Vol. 7, pp. 331-399.

[BeE 87] D. P. Bertsekas and D. El Baz, *Distributed asynchronous relaxation methods for convex network flow problems*, SIAM J. on Control and Optimization, 25 (1987), 74-85.

[BeT 89] D. P. Bertsekas and J. Tsitsiklis, *Parallel and Distributed Computation, Numerical Methods*, Prentice Hall, Englewood Cliffs, N.J., 1989.

[BiD 65] G. Birkhoff and J.B. Diaz, *Nonlinear network problems*, Quart. Appl. Math., 13 (1965), 431-443.

[ChM 69] D. Chazan and W. Miranker, *Chaotic relaxation,* Linear Algebra Appl., 2 (1969), 199-222.

[ChZ 91] E. Chajakis and S.A. Zenios, *Synchronous and asynchronous implementations of relaxation algorithms for nonlinear network optimization*, Parallel Computing, 17 (1991) 873-894.

[Elb 93] D. El Baz, *Asynchronous implementation of relaxation and gradient algorithm for convex network flow problems*, Parallel Computing, 19 (1993), 1019-1028.

[Elb 96] D. El Baz, *Asynchronous gradient algorithms for a class of convex separable network flow problems*, Computational Optimization and Applications, 5 (1996), 187-205.

[ESM 95] D. El Baz, P. Spiteri, J.C. Miellou, et D. Gazen *Asynchronous iterative algorithms with flexible communication for nonlinear network flow problems*, Journal of Parallel and Distributed Computing, 38, October (1996).

[Mie 75] J. C. Miellou, *Algorithmes de relaxation chaotique à retards*, RAIRO, R1 (1975), 55-82.

[MES 94] J.C. Miellou, D. El Baz, P. Spiteri, *A new class of asynchronous iterative algorithms with order intervals*, Rapport LCS 1994-16 (1994), to appear in Mathematics of Computation.

[MiS 85] J.C. Miellou, P. Spiteri, *Two criteria for the convergence of asynchronous iterations*, Computers and Computing, Wiley-Masson, Paris (1985), 91-95.

[Por 69] T. A. Porshing, *Jacobi and Gauss-Seidel methods for nonlinear network problems*, SIAM J. Numer. Anal., 6 (1969), 437-449.

[Rhe 70] W. C. Rheinboldt, *On M-functions and their application to nonlinear Gauss-Seidel iterations and to network flows,* J. Mathematical Analysis and Applications, 32 (1970), 274-307.

[Roc 70] R. T. Rockafellar, *Convex Analysis,* Princeton University Press, Princeton New Jersey, 1970.

[Roc 84] R.T. Rockafellar, *Network Flows and Monotropic Optimization,* John Wiley & Sons, New York, 1984.

[SME 95] P. Spiteri, J.C. Miellou et D. El Baz, *Asynchronous alternating Schwarz method for the solution of nonlinear partial differential equations*, Rapport IRIT 95-17-R 1995.

[TsB 86] J. N. Tsitsiklis and D. P. Bertsekas, *Distributed asynchronous optimal routing in data networks*, IEEE Trans. Auto. Contr., AC-31 (1986), 325-332.

[TsB 87] P. Tseng and D. P. Bertsekas, *Relaxation methods for problems with strictly convex separable costs and linear constraints*, Math. Prog., 38 (1987), 303-321.

[ZeL 88] S. Zenios and R. Lasken, *The Connection Machines CM-1 and CM-2: solving nonlinear network problems,* International Conference on Supercomputing, St Malo, France, (1988), 648-658.

[ZeM 88] S. Zenios and J. Mulvey, *A distributed algorithm for convex network optimization problems*, Parallel Computing, 6 (1988), 45-56.