Mise en œuvre de méthodes de terminaison distribuées pour des algorithmes itératifs asynchrones

Mohamed Jarraya (1), Didier El Baz (2)

(1) Institut Supérieur d'Informatique, 2 avenue Abou Riahane Barouni,
2080 Ariana, Tunisie
(2) LAAS-CNRS, 7, avenue du Colonel Roche,
31077 Toulouse Cedex 4, France
E-mail : jarraya@isi.rnu.tn, elbaz@laas.fr

Résumé

Sur une architecture distribuée ou parallèle, la terminaison des algorithmes itératifs asynchrones présente de nombreuses difficultés vu que généralement chaque processeur possède uniquement des informations locales, il n'y a pas d'horloge globale et les délais de communication peuvent être très longs. Nous présentons une étude comparative de performance de deux méthodes de terminaison distribuées pour la résolution d'un système markovien : la méthode de Bertsekas et Tsitsiklis basée sur l'acquittement systématique de tous les messages reçus et la méthode d'El Baz basée sur l'utilisation de messages d'activation et de désactivation.
mots clés : *algorithmes de terminaison, itérations asynchrones, systèmes markoviens, systèmes distribués.*

1 Introduction

Dans cette étude on s'interesse à la résolution de problèmes de point fixe du type $x^* = F(x^*)$ où F est un opérateur de point fixe de R^n dans R^n à l'aide de méthodes d'approximations successives parallèles asynchrones (cf. [2], [3], [6], [9], [10], [12] et [15]). Un algorithme itératif parallèle asynchrone est une méthode d'approximation successive pour laquelle plusieurs processeurs coopèrent à travers l'échange de données et les réactualisations du vecteur itéré sont effectuées 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 être abandonnées au fur et à mesure que les calculs progressent. Un type de déroulement asynchrone des calculs est donné par la figure 1 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 rapidement.

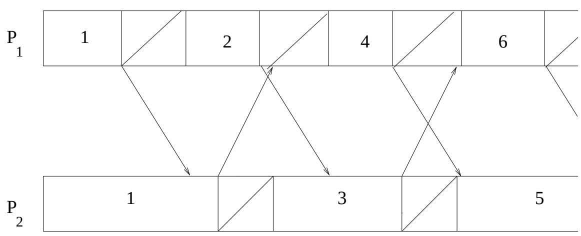

Figure 1: Itération asynchrone

La terminaison d'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 et de la mettre en œuvre.

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 longs. Aussi, il existe relativement peu de méthodes de terminaison pour les algorithmes itératifs asynchrones qui soient efficaces et pour lesquelles il existe des preuves formelles (cf. [4], [7], [8] et [11]).

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 cette étude nous présentons des solutions au problème de la terminaison distribuée des itérations asynchrones. Nous proposons une mise en œuvre de chaque méthode aux paragraphes 2 et 3. Nous appliquons ces méthodes à la résolution parallèle de systèmes Markoviens au paragraphe 4. Nous comparons les performances des différentes méthodes de terminaison au travers d'expérimentation numériques effectuées sur un supercalculateur au paragraphe 5.

2 Méthode de terminaison avec acquittement systématique

Bertsekas et Tsitsiklis ont proposé une méthode de terminaison originale où l'algorithme itératif asynchrone doit être modifié afin qu'il converge en temps fini vers un point fixe suffisamment proche de la solution du problème (cf. [4]).
Initialement un seul processeur est actif qui est appelé processeur racine et tous les autres processeurs sont inactifs. Dans l'état inactif un processeur ne transmet ni message ni acquittement. Un processeur P inactif devient actif quand il reçoit un message provenant d'un autre processeur P'; encore appelé le père de P. Le message qui a activé P, joue un rôle particulièrement important, on lui donne aussi le nom de message critique. Dans l'état actif un processeur transmet les messages et envoie immédiatement un acquittement pour tout message reçu à l'exclusion du message critique. Un processeur P actif devient inactif s'il satisfait sa condition de terminaison locale, s'il a envoyé l'acquittement de tous les messages reçus à l'exception du message critique et s'il a reçu l'acquittement de tous les messages envoyés. En devenant inactif un processeur P envoie l'acquittement du message critique. La terminaison globale est détectée au moment où le processeur racine devient inactif. L'algorithme de détection de la terminaison est donné par la machine à états finis de la figure 2 pour chaque processeur

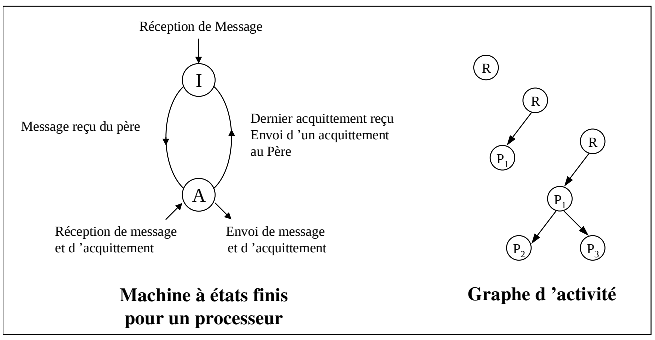

Figure 2: Modèle de l'algorithme de détection de la terminaison avec acquittement systématique

La réactualisation d'une composante du vecteur itéré ne peut avoir lieu que lorsque sa valeur change d'une manière significative. Dans ce cas, cette valeur peut être communiquée aux autres processeurs, sinon elle ne sera ni communiquée ni affectée à cette composante. L'algorithme asynchrone se termine lorsqu'il n'y a aucune modification des valeurs des composantes du vecteur itéré pour tous les processeurs (toutes les conditions de terminaison locales sont satisfaites) et aucun message n'est en transit dans le réseau de communication.
Tous les processeurs sont progressivement activés par la réception de message. Le graphe d'activité évolue en fonction des messages critiques reçus, de la satisfaction des conditions de terminaison locales et de la réception d'acquittements.
La mise en œuvre a été effectuée en associant à chaque processeur actif une liste des processeurs fils. Cette liste contient un ensemble d'attributs: le compteur des messages envoyés à un processeurs fils, un compteur de messages reçus et un compteur d'acquittement. Un processeur fils qui devient inactif est automatiquement supprimé de cette liste. Un processeur père devient inactif quand la liste des fils est vide et sa terminaison locale est satisfaite. Un message de terminaison globale est envoyé par le processeur racine au moment ou il devient inactif.
Cette méthode de terminaison double le nombre de communications de l'algorithme itératif en raison de l'envoi d'acquittements. De plus elle nécessite des conditions plus restrictives que les conditions du théorème général de convergence asynchrone de Bertsekas afin de garantir la convergence de l'algorithme asynchrone modifié (cf. [4]).
Bertsekas et Tsitsiklis ont suggéré dans [4] une autre procédure de terminaison à savoir l'algorithme de "snapshot", ou cliché de Chandy et Lamport (cf. [5]). Cette méthode est basée sur la production de messages de marquage et l'enregistrement des états de liens et des processeurs lorsque les messages de marquage sont délivrés.

3 Méthode de terminaison sans acquittement systématique

Cette méthode de terminaison repose sur l'hypothèse réaliste que le temps de transmission t_t des messages entre toute paire de processeurs adjacents est borné par une constante positive δ dépendant de l'architecture du réseau qui relie les différents processeurs (cf. [7]). On a :

$$t_t < \delta.$$

On suppose aussi que les messages entre toute paire de processeurs adjacents sont reçus suivant leur ordre d'émission (cf. [7] et [9]). Le fonctionnement d'un processeur quelconque mettant en œuvre cette méthode de terminaison est représenté par la machine à états finis de la figure 3 dans laquelle chaque processeur peut avoir trois états : actif (A), inactif (I), et terminal (T).

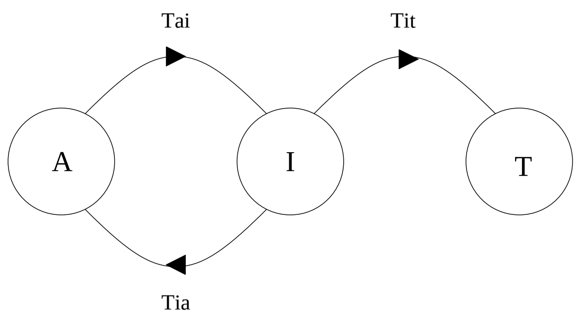

Figure 3: Modèle de terminaison avec messages d'activation et désactivation

Initialement, un seul processeur est actif. Ce processeur est aussi appelé la racine. Tous les autres processeurs sont inactifs. Quatre types de message peuvent être émis par chaque processeur: les valeurs des composantes du vecteur itéré, les messages d'activité, les messages d'inactivité et les messages de terminaison.
Dans l'état actif (A), un processeur P' évalue le test de terminaison local sur la base des dernières valeurs des itérés qui sont disponibles dans sa mémoire locale. Si le test de terminaison local est satisfait, alors le processeur n'effectue pas de réactualisation ; sinon les composantes du vecteur

itéré qui sont assignées à P' sont réactualisées, les valeurs qui résultent de ces calculs sont envoyées aux processeurs adjacents, et le processeur P' attend des messages d'activité pendant un laps de temps égal à $2.\delta$. Le processeur P' ajoute le processeur P à la liste de ses fils dès la reception d'un message d'activité provenant du processeur P. Le processeur P' retire le processeur P de la liste de ses fils dès la reception d'un message d'inactivité provenant de P.
Dans l'état d'inactivité (I), un processeur est en attente de message.
Dans l'état terminal (T) un processeur ne fait plus rien.
Un processeur actif devient inactif (transition Tai) si la liste de ses fils est vide et son test de terminaison local est satisfait. Lorsque la transition Tai est tirée, le processeur envoie un message d'inactivité à son père .
Le processeur racine R passe immédiatement de l'état inactif à l'état terminal (Transition Tit). Lorsque la transition Tit est tirée, des messages de terminaison sont envoyés aux nœuds adjacents de R dans l'arbre recouvrant des processeurs noté T. Un processeur inactif P' différent de R passe de l'état inactif à l'état terminal lorsqu'il reçoit un message de terminaison provenant d'un sommet adjacent de niveau supérieur 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 inférieur dans l'arbre T.
Le fonctionnement de la méthode peut être résumé de la manière suivante : initialement, seul le processeur racine R est actif et tous les autres processeurs sont inactifs. Tous les processeurs deviennent progressivement actifs dès la réception de la valeur d'un sous-ensemble de composantes du vecteur itéré calculées par d'autres processeurs. Un graphe d'activité est engendré ; la topologie de ce graphe évolue au fur et à mesure que les différents messages sont reçus et les tests de terminaison locaux sont satisfaits.
Dans [7], il a été démontré de manière formelle que la méthode présentée dans ce paragraphe détecte la terminaison en un temps fini lorsque celle ci s'est produite.

4 Application à un processus de naissance et de mort

4.1 Le problème

Nous considérons un cas particulier de système à événement discret : un processus de naissance mort donné par la figure 4.

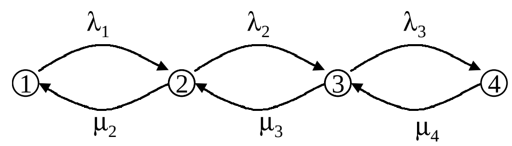

Figure 4: processus de naissance et de mort

La matrice Q aura dans ce cas là une structure tridiagonale par bloc :

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

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

Nous considérons le système Markovien :

$$Qp = 0, \tag{1}$$

où $p = [p_0, ..., p_{n-1}]$ est le vecteur probabilité stationnaire ayant n blocs et de dimension totale $z = \sum_{k=0}^{n-1} z_k$, z_k étant la dimension du bloc k. Les composantes du k-ème bloc z_k, notées par $\pi_{k,i}$ où i est l'indice de la i-ème composante et k l'indice du k-ème bloc, vérifient, $\sum_{k=0}^{n-1} \sum_{i=0}^{z_k-1} \pi_{k,i} = 1$. Q est le générateur infinitésimal; c'est une Z-matrice tridiagonale par bloc, irréductible et apériodique (cf. [13] et [14]). Sous ces hypothèses, il existe un unique vecteur p^* solution de (1) telle que $\sum_{k=0}^{n-1} \sum_{i=0}^{z_k-1} \pi^*_{k,i} = 1$.

4.2 Méthode itératives de résolution

Nous présentons dans ce paragraphe la méthode de relaxation basée sur le schéma itératif de Gauss-

Seidel. Ce dernier présente l'avantage de calculer les composantes du vecteur itéré en fonction des dernières valeurs réactualisées des composantes qui sont disponibles en mémoire. Ceci permet d'une part d'économiser l'espace mémoire utilisé, ce qui est très important pour un problème de très grande dimension et d'autre part d'avoir une exécution plus rapide comparée au schéma itératif de Jacobi.
Soit $Q = D - L - U$, L étant la matrice triangulaire inférieure par bloc, D la matrice diagonale par bloc et U la matrice triangulaire supérieure par bloc; le schéma itératif de Gauss-Seidel est le suivant:

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

La matrice d'itération de Gauss-Seidel est donnée par:

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

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

5 Etude de performance

Nous comparons dans ce paragraphe les performances des itérations asynchrones obtenues sur le supercalculateur CRAY T3E pour les différentes méthode de terminaison présentées aux paragraphes 2 et 3 et qui sont appliquées à la résolution d'un système Markovien de dimension 6110 réparti sur 31 blocs de tailles différentes. La précision de terminaison est $\epsilon = 10^{-4}$. Nous notons par P le nombre de processeurs, I le nombre d'itérations, T le temps d'exécution et E l'efficacité de l'exécution parallèle qui est le rapport entre le temps séquentiel et le temps parallèle divisé par le nombre de processeurs.

Nous avons mesuré expérimentalement le délai de communication entre deux processeurs du CRAY T3E et nous avons obtenu une borne supérieure égale à 0.5ms, donc $2 * \delta = 1ms$ conformément au paragraphe 3. Le tableau 1 montre que la méthode de terminaison avec messages d'activation et désactivation donne généralement de meilleures performances que celle avec acquittement. Le nombre de messages d'activité et d'inactivité est négligeable comparativement au nombre de messages relatifs aux valeurs des composantes du vecteur itéré qui sont échangées entre les processeurs et donc au nombre de messages d'acquittement de l'algorithme de Bertsekas et Tsitsiklis. Toutefois il peut arriver que la méthode avec messages d'activation et désactivation donne de moins bons résultats dans des cas très déséquilibrés qui entrainent nécessairement plus de message d'activation et de désactivation comme par exemple lorsque le nombre de processeur est égal à 12.

	acquittement systématique			activation-désactivation		
P	I	T	E	I	T	E
1	197	48.01		197	48.01	
2	256	27.24	0.881	244	25.05	0.958
3	408	23.09	0.692	369	20.85	0.767
4	431	16.92	0.709	399	15.63	0.767
6	565	12.41	0.644	560	12.19	0.655
8	822	11.78	0.509	853	11.74	0.511
10	1115	9.74	0.492	1301	10.10	0.475
12	997	8.51	0.386	1127	9.08	0.440
16	1700	8.49	0.353	1734	7.87	0.380

Table 1: Performance des methodes de terminaison

References

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

[2] Baudet, G., M.: Asynchronous iterative methods for multi-processors. J. Assoc. Comput., 2:226-244, 1978.

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

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

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

[6] D. El Baz: M-functions and parallel asynchronous algorithms. SIAM J. Numer. Anal.,136-140, 1990. 2524-2527, 1994.

[7] 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.

[8] D. El Baz. A method of terminating asynchronous iterative algorithms on message passing systems. *Parallel Algorithms and Applications*, 9:153–158, 1996.

[9] D. El Baz, *Contribution à l'algorithmique parallèle, le concept d'asynchronisme, étude théorique, mise en œuvre et application.* Habilitation à Diriger les Recherches de l'Institut National Polytechnique de Toulouse, 1998.

[10] 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.

[11] D. El Baz, P. Spiteri, J.C. Miellou, M. Jarraya. *Mathematical study of perturbed asynchronous iterations designed for distributed termination.* International Mathematical Journal, Vol. 1, no. 5, 491-503, 2002.

[12] A. Frommer, D.B. Szyld. *On asynchronous iterations.* Journal of Computation and Applied Mathematics,777-783, 2000.

[13] M. Jarraya *Mise en œuvre et étude de performance d'algorithmes itératifs parallèles sur diverses architectures. Application à l'optimisation, à la commande et à la résolution de systèmes Markoviens.* Doctorat, Université Paul Sabatier, Toulouse, 2000.

[14] M. Jarraya, D. El Baz. *Asynchronous iterations for the solution of Markov systems.* 3^{rd} International Workshop on the Numerical Solution of Markov Chains, (NSMC'99), Saragosse (Espagne), 335-338, 1999.

[15] J. C. Miellou. *Algorithmes de relaxation chaotique retards.* RAIRO, vol. R1, 55-82, 1975.