RENPAR'14 / ASF / SYMPA
Hamamet, Tunisie, 10 – 13 avril 2002

Contrôle amorti des synchronisations pour le test d'arrêt des méthodes itératives

Nicolas Maillard[1], El Mostafa Daoudi[2]*, Pierre Manneback[3], Jean-Louis Roch[1]

[1] Laboratoire ID-IMAG (UMR 5132)
Projet APACHE (CNRS/INPG/INRIA/UJF),
51, av. Jean Kuntzmann
38330 Montbonnot Saint-Martin, France
Nicolas.Maillard@imag.fr,
Jean-Louis.Roch@imag.fr

[2] Université Mohamed 1er, Faculté des Sciences
Dept. Maths et Informatique
60 000 Oujda, Maroc
mdaoudi@sciences.univ-oujda.ac.ma

[3] Service d'Informatique
Faculté Polytechnique de Mons
Rue de Houdain, 9
7 000, Mons, Belgique
Pierre.Manneback@fpms.ac.be

Résumé
Dans ce travail, nous proposons des nouvelles techniques pour réduire le nombre de synchronisations nécessaires au contrôle du test d'arrêt dans les méthodes itératives. L'objectif est d'améliorer les performances en temps de réponse de ces méthodes sur des systèmes parallèles à mémoire distribuée, en particulier les grilles de processeurs. En effet, sur de telles architectures les synchronisations globales sont très pénalisantes. Nous proposons un contrôle amorti des synchronisations qui ne modifie pas la structure de l'algorithme itératif initial; par conséquent, la convergence est assurée dans les mêmes conditions que l'algorithme initial qui requiert une synchronisation globale après chaque itération.

Mots-clés : Méthodes itératives, test d'arrêt, synchronisation.

---

1. Introduction

De manière générale, une méthode itérative consiste à itérer un calcul (corps de l'itération) tant qu'une certaine condition booléenne n'est pas réalisée. Le test de la condition entraîne une synchronisation entre deux étapes consécutives de calcul. Sur une architecture parallèle, même si le corps de l'itération et le calcul de la condition peuvent éventuellement être distribués, le test booléen requiert généralement une synchronisation globale entre tous les processus. Sur les architectures à mémoire distribuée (typiquement une grappe de processeurs ou une grille de grappes), il est bien connu qu'une telle synchronisation globale est extrèmement coûteuse car elle entraîne un temps d'attente globale : tous les processeurs doivent en effet attendre la contribution du processeur le plus lent pour pouvoir poursuivre.

Aussi, différentes approches ont été proposées pour pallier à ce problème. Une première approche consiste à grouper les synchronisations globales d'une itération pour les effectuer en même temps; cette technique est par exemple utilisée dans [1] pour la méthode du gradient conjugué. Une autre approche, qui a

*Travail effectué dans le cadre de l'AI MA/01/19 et dans le cadre d'un séjour comme professeur invité à la FPMs-Mons

fait l'objet de nombreuses études, est de transformer la méthode séquentielle synchrone en une méthode asynchrone pour éviter les synchronisations; dans [5], D. El Baz présente un aperçu de ces méthodes qui sont basées sur un modèle de mémoire avec consistance faible, par exemple consitance PRM [7]. Ainsi, lorsqu'un processeur a besoin de lire une donnée, il l'accède sans synchronisation forte avec le processeur qui l'a écrit (typiquement avec un protocole lecteur-rédacteur). La sémantique "séquentielle" du calcul est alors perdue et, par conséquent, l'algorithme séquentiel peut être profondément modifié et certaines de ses propriétés perdues. Parmi les problèmes critiques se trouvent ceux de la convergence (condition de convergence et nombre d'itérations) et de la terminaison lorsqu'on atteint la convergence; de plus, les coûts de communications peuvent être très improtants pour ces méthodes [3, 5].

Dans cet article, nous proposons une autre stratégie, originale à notre connaissance, pour diminuer le coût de la synchronisation globale lié au test de la condition d'arrêt. Notre but est de diminuer le nombre de synchronisations par rapport à la méthode itérative standard sans modifier le corps de l'itération. Aussi, nous nous plaçons dans le cas où la seule synchronisation globale est celle que nécessite le contrôle du test d'arrêt. De plus, nous supposons qu'il est possible d'itérer le corps de la boucle une infinité de fois sans provoquer d'erreurs. En outre, une fois la condition d'arrêt atteinte, nous supposons qu'elle reste ensuite toujours vraie pour les itérés suivants. Ces hypothèses sont valides d'un point de vue pratique pour les méthodes itératives numériques qui convergent vers un unique point fixe; la condition d'arrêt mesure généralement l'erreur entre l'itéré et le point fixe de l'itération. Typiquement, cela sera le cas quand on itère un produit matrice vecteur où la matrice est creuse par blocs (avec quelques diagonales non nulles par exemple), l'arrêt consistant au test après chaque itération de la norme d'un vecteur résidu qui doit devenir inférieure à une tolérance ϵ donnée.

Notre algorithme calcule en sortie les mêmes valeurs que l'algorithme de référence, obtenues après le même nombre d'itérations; cela garantit de conserver les propriétés (en particulier de convergence) de l'algorithme synchrone initial. Notre approche est inspirée de la méthode séquentielle ρ de Pollard [4] pour calculer, avec peu de synchronisations, le plus petit itéré à partir duquel la condition d'arrêt reste vraie. Considérant l'itération d'une fonction f quelconque dans un ensemble fini à partir d'un point initial x^0, la méthode ρ calcule le premier itéré k qui tombe sur un cycle et sa longueur r; autrement dit, les plus petits k et r tel que $f^r[f^k(x^0)] = f^k(x^0)$. L'algorithme ρ de Pollard requiert pour ce calcul moins de $2(k+r)$ itérations de la fonction f. Cette méthode est notamment utilisée pour le calcul du logarithme discret. Ici, nous utilisons une technique similaire pour calculer le premier itéré à partir duquel la condition d'arrêt reste toujours vraie.

Dans la section 2 nous présentons l'algorithme séquentiel. La section 3 est consacrée à la présentation d'une première méthode, classique, basée sur un contrôle périodique à *k-pas*. La section 4 consiste en la présentation d'une méthode amortie qui est une contribution originale. Enfin, dans la section 5 nous présentons un compromis algorithmique entre ces deux techniques; ce compromis est obtenu par une combinaison des deux méthodes précédentes.

2. Algorithme synchrone standard

Supposons que la méthode itérative converge en n_* itérations. Si on effectue un contrôle du test d'arrêt à chaque itération, alors, puisque chaque test nécessite une synchronisation globale, on effectuera au total n_* synchronizations. On peut estimer le temps $T_{standard}$ d'exécution de la méthode standard à:

$$T_{standard} = n_*(t_{cal} + t_{sync})$$

où t_{sync} est le temps que prend la synchronisation globale liée au test (i.e. en incluant les opérations et les communications nécessaires à l'évaluation et à la diffusion du résultat du test) et t_{cal} est le temps du corps d'une itération (qui peut comprendre le temps de calcul et éventuellement des coûts des communications généralement locales ici). Dans la suite, le temps sera mesuré en nombre d'itérations et en nombre de synchronisations.

3. Algorithme synchrone avec k-pas

Afin de diminuer le nombre de synchronisations, la méthode classiquement utilisée consiste à faire le contrôle du test d'arrêt de manière périodique, tous les k pas. Le choix de k dépend de différents critères : non seulement de la méthode elle-même, mais aussi des données en entrée. Dans [2], P.E. Bernard et al. utilisent une telle méthode pour contrôler la convergence et l'équilibrage d'un calcul de dynamique moléculaire. Dans ce cas pratique, le calcul s'arrête lorsque le test d'arrêt devient vrai, donc uniquement après une itération d'indice multiple de k. La valeur en sortie de la simulation est alors différente de la valeur délivrée lorsque le test est effectué à chaque pas; en effet, dans ce cas, le résultat correct est celui obtenu après n_* itérations, où n_* est l'itération pour laquelle le test d'arrêt est vérifié pour la première fois.
Il est cependant facile de modifier cette méthode pour délivrer en sortie le résultat de l'itération n_*. La méthode résultante est appelée *méthode k-pas*. Plus précisément, la méthode k-pas est composée de deux phases. Dans la première phase, le contrôle du test d'arrêt ne sera effectué qu'après k itérations successives, c'est à dire aux itérations $k, 2k, \cdots, qk = n_1$ où $qk = n_1$, avec $q \in \mathbb{N}^*$, est l'itération pour laquelle le test d'arrêt est vérifié pour la première fois. La deuxième phase consite à déterminer l'itération $n_* \in [(q-1)k+1, qk]$. Pour cela on repart de l'itération $(q-1)k+1$ en effectuant cette fois le test d'arrêt après chaque itération. On aboutit ainsi à l'itération n_* en effectuant au total

$$qk + (n_* - (q-1)k) = n_* + k$$

itérations et

$$q + (n_* - (q-1)k) = q + r$$

synchronisations avec $1 \leq r < k$, soit $q + k - 1$ synchronisations au pire des cas.

Le coût total de cet algorithme est :

$$T_{k-pas} \leq (n_* + k).t_{cal} + (\left\lceil \frac{n_*}{k} \right\rceil + n_* \mathrm{mod} k).t_{sync}.$$

Il est clair que le choix de k est complétement dépendant de la valeur de n_* qui est inconnue. Si k est grand, alors on gagne au niveau nombre de synchronisations dans la première phase de la méthode mais on perd dans la deuxième phase pour retrouver la valeur exacte de n_*. De manière duale, si k est petit alors on perd pendant la première phase au niveau nombre de synchronisations, mais on gagne lorsqu'on cherche la valeur exacte de n_*.
Le gain apporté par la méthode k-pas par rapport à la méthode standard est :

$$T_{standard} - T_{k-pas} \geq -k.t_{cal} + \left[\left\lfloor \frac{n_*}{k} \right\rfloor .(k-1) + 1 \right] .t_{sync}$$

On en déduit que, si l'on peut choisir k entier tel que $\frac{k^2}{k-1} \leq n_* \frac{t_{sync}}{t_{cal}}$, la méthode k-pas est toujours plus performante que la méthode standard.

4. Contrôle ρ-amorti

Cette technique est basée sur un contrôle périodique du test d'arrêt, comme dans le cas précédent, mais cette fois avec un pas variable. Plus précisément, elle consiste à faire les tests de contrôle d'arrêt aux itérations $\rho^0, \rho^1, \cdots, \rho^k, \cdots$. Supposons que le test d'arrêt est vérifié pour la première fois à l'itération $n_1 = \rho^{k_1}$. On a alors $\frac{n_1}{\rho} = \rho^{k_1 - 1} < n_* \leq \rho^{k_1} = n_1$.
Afin de calculer les sorties obtenues exactement après l'itération n_*, on ré-applique le même procédé en repartant du dernier itéré $\frac{n_1}{\rho} + 1 = \rho^{k_1 - 1} + 1$, pour lesquelles le contexte de l'itération a été sauvegardé : on effectue alors les tests d'arrêt aux itérations $\frac{n_1}{\rho} + 1, \frac{n_1}{\rho} + \rho^1, \cdots, \frac{n_1}{\rho} + \rho^{k_2}$, où k_2, avec $k_2 \leq \lceil log(n_1/\rho) \rceil$, est la plus petite valeur pour laquelle le test d'arrêt est vérifié pour la première fois. On a alors

$$\frac{n_1}{\rho} < \frac{n_1}{\rho} + \rho^{k_2 - 1} < n_* \leq \frac{n_1}{\rho} + \rho^{k_2} \leq n_1.$$

On pose $n_2 = \rho^{k_2}$ et on itère le procédé à nouveau à partir de $\frac{n_1}{\rho} + \frac{n_2}{\rho} + 1$. On calcule ainsi par récurrence k_l, $l \leq \log_\rho n_*$, tels que:

$$\sum_{i=1}^{l-1} \rho^{k_i - 1} + \rho^{k_l - 1} < n_* \leq \sum_{i=1}^{l-1} \rho^{k_i - 1} + \rho^{k_l}.$$

Par construction des entiers $k_1, \ldots, k_l$, on a :

$$n_* = \sum_{i=1}^{\log_\rho n_*} \rho^{k_i - 1}.$$

Cette propriété permet de majorer facilement les temps requis par cette méthode pour le calcul des itérations et des tests :

- le nombre total S de synchronisations effectuées est $S = k_1 + k_2 + \cdots + k_l$. Or, d'après la proposition précédente, $k_i - 1 \leq \log_\rho n_*$; d'où :

$$S = \sum_{i=l}^{1} k_i \leq \lceil log_\rho(n_*) \rceil^2$$

- Le nombre N d'itérations calculées pour aboutir à l'itération n_* est au pire égal à

$$N = \sum_{i=1}^{l} \rho^{k_i} = \rho . \sum_{i=1}^{l} \rho^{k_i - 1} \leq \rho . n_*$$

Finalement, le temps du calcul avec contrôle ρ-amorti s'écrit :

$$T_{\rho - amorti} \leq \rho . n_* . t_{cal} + \lceil log_\rho(n_*) \rceil^2 . t_{sync}.$$

Ainsi, la méthode ρ-amorti permet de diminuer exponentiellement les synchronisations globales, au prix de l'augmentation d'un facteur constant du nombre d'itérations.
Ceci permet de préciser des choix pour ρ qui garantisse un temps d'un facteur plus rapide que l'algorithme standard. En effet,

$$\frac{T_{\rho - amorti}}{T_{standard}} \leq \rho \frac{t_{cal}}{t_{cal} + t_{sync}} + \frac{\log_\rho^2(n_*)}{n_*} \frac{t_{sync}}{t_{cal} + t_{sync}}$$

Ainsi, si l'on suppose que $\log_\rho^2(n_*) < n_*$ (asymptotiquement vrai et toujours vérifié), si $t_{sync} > t_{cal}$, alors en choisissant $\rho < 1 + \frac{t_{sync}}{t_{cal}}$, la méthode ρ-amorti est au moins d'un facteur plus rapide que la méthode standard. Ses performances sont alors garanties toujours meilleures que l'algorithme k-pas.

5. Compromis k-pas/ρ-amorti

Dans cette section, nous étudions un compromis entre les deux approches de contrôle *k-pas* et ρ-amorti, appelé (k, ρ)-compromis :

- On applique la première phase de l'algorithme *k-pas* en effectuant des synchronisations de contrôle tous les k pas jusqu'à $n_1 = qk$, afin de pallier le défaut de l'approche amortie qui est de faire des "petit pas" au départ des itérations.

- Une fois n_1 atteint on utilise l'algorithme ρ-amorti à partir de $(q-1)k + 1$ avec $(q-1)k < n_* \leq qk$. Posons $n_* = (q-1)k + r$ avec $r < k$, l'algorithme converge en effectuant au plus $\rho.r$ itérations supplémentaires et $\log_\rho^2(r)$ synchronisations.

On déduit que l'algorithme (k, ρ)-compromis demande au plus $qk + \rho.r$ itérations et $q + \log_\rho^2(r) = q + \log_\rho^2(n_* - (q-1)k)$ synchronisations. D'où, en remarquant que $qk = n_* - r + k$ et $r < k$:

$$T_{((k,\rho) - compromis} \leq (n_* + k.\rho).t_{cal} + \left(\left\lceil \frac{n_*}{k} \right\rceil + \log_\rho^2(k) \right) .t_{sync}$$

6. Comparaison des algorithmes

En bilan de ces 3 algorithmes, on peut dresser le tableau 4.1., avec $k \geq 2$ et entier et $\rho > 1$: Comme

	Itérations	Synchronisations
Standard	n_*	n_*
k-pas	$n_* + k$	$< \frac{n_*}{k} + k$
ρ-amorti	$< \rho.n_*$	$< \log_\rho^2(n_*)$
(k,ρ)-amorti	$< n_* + k.\rho$	$< \frac{n_*}{k} + \log_\rho^2(k)$

Table 1: *Coûts des trois algorithmes*

$n_* = (q-1)k+r$, $r < k$, l'algorithme à *k-pas* est meilleur en nombre d'itérations. Il est ρ fois meilleur (en ordre de grandeur) pour le nombre d'itérations que l'algorithme ρ-amorti alors que l'algorithme (k,ρ)-amorti, compromis entre les deux, conserve un nombre d'itérations proche de celui du k-pas, la différence entre les deux est au plus égale à $\rho.k$.

L'avantage primordial de l'algorithme ρ-amorti est le gain exponentiel en nombre de synchronisations. Pour l'algorithme (k,ρ)-compromis, le gain en terme de nombre de synchronisations dépend de l'ordre de grandeur de k. Si k est de l'ordre de n_* il est évident que dans ce cas les deux algorithmes sont équivalents en nombre de synchronistions (de l'ordre de $log^2(n_*)$). Par contre si $k << n_*$, ceci conduit à ce que q est de l'ordre de n_*, dans ce cas l'algorithme ρ-amorti est meilleur en nombre de synchronisations. Dans le cas où k et q sont du même ordre, alors $k+q$ peut être vu comme de l'ordre de grandeur de $2\sqrt{n_*}$, à comparer à $log^2(n_*)$.

Remarque:

On a jusqu'ici cherché les valeurs obtenues exactement après l'itération n_*. Mais, en particulier pour les méthodes itératives numériques qui convergent vers une solution, on peut très bien se contenter des valeurs obtenues à toute itération $n_1 \geq n_*$. Dans ce cas, les algorithmes proposés se simplifient, puisqu'il suffit de s'arrêter dès que le test devient vrai. Le tableau 2. compare les performances des algorithmes k-pas et ρ-amorti dans ce cas : L'algorithme ρ-amorti est alors d'autant plus performant que l'algorithme

	Itérations	Synchronisations
Standard	n_*	n_*
k-pas	$n_* + k$	$\frac{n_*}{k}$
ρ-amorti	$\rho^{\lceil \log_\rho n_* \rceil} < \rho.n_*$	$\log_\rho n_*$

Table 2: *Comparaison des algorithmes avec arrêt sans retour à l'itération* n_*

k-pas lorsque le temps de synchronisation t_{sync} est grand devant $\rho.t_{cal}$.

7. Conclusion

Dans cet article, nous avons proposé un schéma ρ-amorti pour contrôler le test d'arrêt d'une méthode itérative. Il permet d'estimer, à la volée, le nombre n_* de pas pour atteindre la fin de l'itération, en ne réalisant que $\Theta(\log^2 n_*)$ calculs de test d'arrêt qui entraînent une synchronisation globale et en conservant un coût total du même ordre que l'algorithme séquentiel ($\Theta(n_*)$ itérations).

Notre résultat améliore d'un facteur exponentiel les autres méthodes de contrôle séquentiel ou k-pas qui requièrent toute $\Omega(n_*)$ synchronisations.
D'un point de vue théorique, l'estimation du nombre de synchronisations nécessaires pour calculer à la volée n_* sous la contrainte de n'effectuer que $O(n_*)$ itérations est, à notre connaissance, un problème ouvert. On peut remarquer que $\Omega(\log n_*)$ est une borne inférieure pour le nombre de tests d'arrêt (synchronisations) nécessaires pour évaluer à la volée n_*. L'algorithme ρ-amorti requérant $O(\log^2 n_*)$ tests, il fournit une borne supérieure à un facteur logarithmique de la borne inférieure.
D'un point de vue pratique, ce contrôle peut être appliqué à de nombreuses algorithmes numériques. Dans [6], N. Maillard montre qu'il peut être en particulier appliqué à la résolution de l'équation de Schrödinger pour la simulation d'une boîte quantique.
La perpsective la plus importante de ce travail concerne alors l'expérimentation dans un cadre pratique du contrôle amorti du test d'arrêt. En effet, sur une grille de machines, les machines sont généralement de vitesse non uniformes et éventuellement variables si elles sont partagées entre plusieurs utilisateurs : le coût d'évaluation t_{sync} du test d'arrêt, qui inclue les temps d'attente dûs à la synchronisation globale, peut alors devenir très grand devant le temps t_{calc} de calcul d'une itération. Généralement, notamment lorsqu'on manipule des structures creuses, le corps de l'itération ne requiert que peu de synchronisations bi-point. Ainsi, la diminution exponentielle du nombre de synchronisations apportée par l'algorithme avec contrôle ρ-amorti ($\log^2 n_*$ au lieu de n_*) apparaît particulièrement intéressant.

Bibliographie

1. Baserman (A.), Reichel (B.) et Scheltoff (C.) - Preconditioned CG methods for sparse matrices on massively parallel machines - Parallel Computing 23 (1997).
2. Bernard (Pierre-Eric), Gautier (Thierry) et Trystram (Denis) - Large Scale Simulation of Parallel Molecular Dynamics - Proceedings of Second Merged Symposium IPPS/SPDP 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing, San Juan, Puerto Rico, avril 1999.
3. Bertsekas (Dimitri D.) et Tsitsiklis (John N.) - Parallel and distributed computation, numerical methods - Prentice-Hall, Englewood Cliffs N.J. 1989.
4. Cormen (T.H.), Leiserson (C.E.), Rivest (R.L.) et Stein (C.) - Introduction to Algorithms - Second Edition, McGraw-Hill, 2001.
5. El Baz (Didier) - An efficient termination method for asynchronous iterative algorithms on message passing architectures - LAAS Report No , In proceedings of the International Conference on Parallel and distributed Computing Systems, Dijon, Volume 1, 1996.
6. Maillard (Nicolas) - Calcul haut-performance et mecanique quantique: analyse des ordonnancements en temps et en mémoire - Thèse, ID-IMAG, 19 Novembre 2001.
7. Raynal (Michel) et Schiper (André) - From causal consistency to sequential consistency in shared memory systems - Sringer-Verlag LNCS Series 1026, pp.180-194, dec. 1995.