Projet PPAR
Solution itérative de gros systèmes linéaires

Pierre Fortin, Christoph Lauter

12 octobre 2013

1 Présentation du sujet

Plusieurs applications modernes, que nous utilisons presque naturellement, sont très gourmandes en calcul :
- Prédictions météorologiques
- Recherches sémantiques dans des pages Web
- Méthodes d'éléments finis p.ex. pour
  - imagerie par résonance magnétique,
  - construction des bâtiments, routes, ponts,
  - simulation de flux en avionique, chimie, physique
- Simulations en physique
- ...

Le cœur de calcul de toutes ces applications se ramène toujours à une résolution de système linéaire $A\,x = b$.

Accélérer la résolution d'un système veut donc dire rendre des prédictions météorologiques plus fiables, permettre aux moteurs de recherche de donner des résultats plus appropriés, éviter des attentes trop longues à des patients après un scan à l'hôpital etc.

Les nombre des cœurs et des nœuds de calcul disponibles a considérablement augmenté ces dernières années. Il s'agit donc *juste* d'exploiter cette puissance de calcul par programmation parallèle afin d'accélérer cet algorithme central, la résolution de systèmes linéaires.

Pour certains types de systèmes linéaires [1], une résolution est possible avec un algorithme itératif, appelé méthode de Jacobi. Cette méthode est bien adaptée au calcul parallèle.

Pour un système linéaire $A\,x = b$ où $A \in \mathbb{R}^{n\times n}$ est la matrice du système (avec les éléments a_{ij}), $b \in \mathbb{R}^n$ le vecteur de droite connu et $x \in \mathbb{R}^n$ le vecteur des inconnues, la méthode de Jacobi consiste à répéter, jusqu'à convergence, la mise à jour suivante d'une solution approchée $x^{(m)}$:

$$x_i^{(m+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{i \neq j} a_{ij}\, x_i^{(m)} \right),\ i = 1, \ldots, n.$$

Pour commencer l'itération, tout vecteur $x^{(0)}$ peut être choisi, par exemple le vecteur initialisé comme ceci : $x_i^{(0)} = 1$.

La convergence, qui conditionne l'arrêt de l'itération, peut être détectée par plusieurs moyens différents. Un procédé simple consiste à comparer les solutions approchées de deux itérations consécutives, $x^{(m)}$ et $x^{(m+1)}$. Si elles ne diffèrent – en valeur absolue – en aucun élément plus d'un certain seuil ε petit, la mise à jour n'a plus d'effet parce que le procédé a convergé. L'algorithme précis de la méthode de Jacobi est décrit par le pseudo-code donné avec la Figure 1.

Dans ce projet, nous allons nous intéresser à la parallélisation de la méthode de Jacobi en MPI. Nous allons travailler sur des systèmes linéaires gros. Notre matrice A atteindra une taille de 67000×67000 éléments, c-à-d. elle aura près de $4,5$ milliards d'entrées. Une difficulté particulière viendra du fait que des matrices d'une telle taille ne pourront pas être stockées sur aucun nœud de calcul seul mais seulement sur plusieurs nœuds, pris en parallèle.

2 Travail à effectuer

Un exemple possible de code séquentiel vous est fourni : ce programme n'effectue bien sûr pas le calcul d'une solution pour un problème linéaire de la taille visée (67000×67000).

1. Pour les matheux d'entre vous : il s'agit de systèmes linéaires dont la matrice est à diagonale dominante

Entrées : $A \in \mathbb{R}^{n \times n}$ une matrice pour laquelle Jacobi converge,
$b \in \mathbb{R}^n$ un vecteur de droite connu,
$\varepsilon \in \mathbb{R}^+$ une constante de test de convergence
Sorties : Un vecteur $x \in \mathbb{R}^n$ résolvant $A\,x = b$ approximativement

```
début
    pour i = 1, ..., n faire x_i ← 1
    répéter
        δ ← 0;
        pour i = 1, ..., n faire
            c ← b_i;
            pour j = 1, ..., n faire
                si i ≠ j alors c ← c − a_ij · x_j
            fin
            c ← c / a_ii;
            δ ← max(δ, |x_i − c|);
            x_i ← c;
        fin
    jusqu'à δ < ε;
fin
```

Algorithme 1: La méthode de Jacobi

Le code séquentiel commence par la génération d'un système linéaire aléatoire satisfaisant les pré-conditions de l'itération de Jacobi. Ensuite, le programme résout le système avec l'itération de Jacobi. Finalement, le code calcule le vecteur résidu $r = b - A\,x$, afin de vérifier que l'itération de Jacobi a bien marché et qu'il s'agit, pour x, bien d'une solution du système linéaire donné.

Votre programme parallèle devra également exécuter ces trois étapes : génération du système, solution par Jacobi et vérification par calcul du résidu. Les étapes de génération et de vérification étant purement techniques, seule l'itération de Jacobi sera chronométrée et analysée pour accélération et efficacité parallèles.

En revanche, comme la matrice A du système pour la taille ciblée (67000×67000) occupera près de 36Go de mémoire vive ($67000^2 \cdot 8 = 35.9 \cdot 10^9$, un nombre flottant double précision occupe 8 octets), la matrice ne rentrera en mémoire sur aucun des nœuds de calcul à votre disposition. Seulement distribuée sur les 12 nœuds de calcul, elle pourra être stockée en mémoire vive.

Ceci implique que, bien que seule l'itération de Jacobi soit prise en compte pour l'analyse d'efficacité parallèle, les deux phases de génération de la matrice aléatoire et de calcul du résidu devront être parallélisées aussi. Ceci dit, la parallélisation que vous choisirez pour ces deux phases ne doit pas être très sophistiquée.

Concrètement, votre programme parallèle devra afficher, tout comme le fait le programme séquentiel fourni, le temps « horloge » écoulé (*wall-clock time*) pour l'itération de Jacobi, la valeur maximale du résidu et si cette valeur satisfait la condition de correction (`Solution OKAY` ou `Solution NOT okay`).

Dans votre travail, vous vous intéresserez aux points suivants :

1. Que devient la condition de terminaison de l'itération de Jacobi en parallèle ?
2. La décomposition suivante est naturelle pour la parallélisation de la méthode de Jacobi : la matrice A est décomposée en blocs de lignes et les vecteurs x et b sont distribués (et non dupliqués) sur tous les processus. Décrire et implémenter un premier algorithme d'itération de Jacobi parallèle, en se servant d'une routine MPI de communication collective.
3. Proposer une méthode ne nécessitant pas de communication collective pour la mise à jour du vecteur x. Décrire la méthode et l'implémenter. Analyser et comparer les performances de cette deuxième implémentation par rapport à l'implémentation qui utilise une communication collective.
4. Rapidement décrire les algorithmes parallèles choisis pour la parallélisation des phases de génération aléatoire du système et de vérification par calcul du résidu. Implémenter ces algorithmes.
5. Améliorer la solution retenue pour l'itération de Jacobi en introduisant, par exemple :
   - un recouvrement des communications par le calcul,
   - une programmation hybride MPI+OpenMP permettant de mieux exploiter les processeurs multicœur dont vous disposez. On pourra alors comparer et analyser les performances d'une implémentation « MPI+OpenMP » et d'une implémentation « MPI pur » (sans multi-threading), et ce pour un nombre fixé de cœurs CPU.
   - ...

3 Travail à remettre

Vous devrez remettre les documents suivants :

- le code source, sous la forme d'une archive `tar` compressée et nommée suivant le modèle `projet_PPAR_nom1_nom2.tar.gz`. L'archive ne doit contenir aucun exécutable, et elle devra contenir un fichier `Makefile` : la commande `make` devra lancer la compilation, et la commande `make exec` devra lancer l'exécution parallèle (pour la configuration d'une matrice 67000×67000 visée, et avec le code final). Veillez à ce que le code compile sans avertissement et surtout sans aucune erreur.
- un rapport (au format PDF [2], de 5 à 10 pages, nommé suivant le modèle `rapport_PPAR_nom1_nom2.pdf`) présentant vos choix, vos implémentations (sans code source), vos résultats et vos conclusions.
  Les tests de performance devront présenter les temps de calcul accompagnés soit des accélérations parallèles, soit des efficacités parallèles, soit les deux. Ces tests doivent couvrir des tailles variées et des nombres de processus différents. Il faudra notamment indiquer le meilleur résultat obtenu pour le temps de calcul moyen sur un problème avec une matrice A de taille 67000×67000 sur 12 nœuds multicœurs.
  Pour procéder à vos tests, vous utiliserez les machines des salles prévues à cet effet. Comme le matériel installé varie avec les salles, vous devez indiquer dans votre rapport dans quelle salle vous aurez fait vos mesures.
  Une comparaison des différentes implémentations, une analyse la plus approfondie possible de tous vos résultats accompagnée d'une réflexion sur le comportement de votre programme seront particulièrement appréciées.
  Tout phénomène d'accélération (ou d'absence d'accélération) inattendu et toute optimisation supplémentaire doivent être analysés dans le rapport.

4 Quelques précisions importantes

- Le projet est à réaliser par binôme. Aucun trinôme ne sera accepté.
- Les programmes parallèles rendus doivent être basés sur le code séquentiel donné. Ils ne doivent pas constituer une réimplémentation complète différente de la méthode de Jacobi. Ceci implique que le langage utilisé pour les programmes rendus doit être le langage C.
- Vous devez lire le document intitulé « Projet PPAR : conditions d'utilisation d'OpenMPI ». Vous veillerez notamment à n'utiliser qu'une salle à la fois pour vos tests (12 machines sont nécessaires), et vous n'oublierez pas de tuer tous vos processus sur toutes les machines utilisées à la fin de vos tests.
- Attention aux quotas sur les disques : il n'est pas demandé de (et vous ne devez donc pas) sauvegarder les matrices A ou vecteurs x et b utilisés sur disque.
- Veillez à bien compiler avec l'option de compilation `-O3`, et en séquentiel et en parallèle.
- Le projet devra être remis au plus tard le 11 novembre à 23h59 (heure de Paris) par courriel à : `pierre.fortin@lip6.fr` et `christoph.lauter@lip6.fr`
- En cas d'imprévu ou de problème technique commun, n'hésitez pas à nous contacter pour que nous puissions vous proposer une solution ou une alternative.

2. Les rapports remis au format Microsoft Word ou dans tout autre format non-PDF ne seront pas considérés.