High Performance Computing
Parallélisation de l'algorithme de Jacobi

Rémi Garde

Mars 2018

Définition du problème

Avec $n \in \mathbb{N}$, $A = (a_{ij})_{(i,j)\in[\![1,n]\!]^2}$, $B = (b_i)_{i\in[\![1,n]\!]}$.
Résolution du système d'équations linéaires

$$AX = B \tag{1}$$

pour $X = (x_i)_{i\in[\![1,n]\!]}$.

Résolution : Décomposition

A est séparée en 2 matrices D et R, avec D matrice de la diagonale de A et R les éléments non diagonaux de A :

$$A = \begin{pmatrix} a_{1,1} & 0 & \cdots & 0 \\ 0 & a_{2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{n,n} \end{pmatrix} + \begin{pmatrix} 0 & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & 0 & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & 0 \end{pmatrix}$$

La formulation (1) devient donc équivalente à :

$$X = D^{-1}(B - RX) \tag{2}$$

Résolution : Itérations

$$X^{(k)}, k \in \mathbb{N} = \begin{cases} X^{(0)} = 0 \\ X^{(k+1)} = D^{-1}(B - RX^{(k)}), \quad k \in \mathbb{N}^* \end{cases}$$

Formule de récurrence pour les coefficients :

$$\forall k \in \mathbb{N}^*, \forall i \in [\![1, n]\!], x_i^{(k+1)} = \frac{1}{a_{ii}}(b_i - \sum_{i \neq j} a_{ij} x_j^{(k)}) \qquad (3)$$

Convergence

Avec $A' = -D^{-1}R$ et $B' = D^{-1}B$:

$$X = A'X + B'$$

$$X^{(k+1)} - X = A'(X^{(k)} - X)$$

d'où

$$\lim_{k \to \infty} \|X^{(k+1)} - X\| = 0 \Leftrightarrow \rho(A') < 1$$

La méthode de Jacobi converge si et seulement si A' est de rayon spectral strictement inférieur à 1

Convergence

En remarquant que $a'_{i,j} = \begin{cases} 0 & \text{si } i = j \\ \frac{-a_{ij}}{a_{ii}} & \text{si } i \neq j \end{cases}$

et en prenant λ une valeur propre de A' et $Y = (y_i)$ un vecteur propre associé, dans le cas de A à diagonale strictement dominante :

$$|\lambda| \cdot \|Y\|_\infty = \max_i |\sum_{j \neq i} \frac{-a_{ij}}{a_{ii}} y_j|$$

$$|\lambda| \cdot \|Y\|_\infty \leq \max_i |\frac{1}{a_{ii}}| \sum_{j \neq i} |a_{ij}| \cdot \|Y\|_\infty \leq \|Y\|_\infty$$

La méthode de Jacobi converge pour A à diagonale strictement dominante

Structures de données

- Matrices et vecteurs stockés comme tableaux
- `matrix.c` et `vector.c` définissent les fonctions de création, destruction, affichage et chargement des données

Partage des données

Partage des n lignes entre les q processus. Avec un processus n°p, en notant $d = \lfloor \frac{n}{p} \rfloor$ et $r = \mathrm{mod}(n, p)$:

- Nombre de lignes : d si $p < r$, $d + 1$ sinon ;
- Sa première ligne est $dp + \min(r, p)$
- le processus contenant la ligne i est le n°$\lfloor \frac{iq}{n} \rfloor$

Parallélisation

- La parallélisation de (3) n'est a priori pas réalisable
- chaque processus va contenir deux vecteurs :
  - `x_local` : lignes spécifiques au processus
  - `x_global` : totalité du vecteur $X^{(k)}$
- chaque itération commence par le partage des `x_local` pour que chaque processus remplisse son `x_global`
- on peut ensuite calculer la nouvelle itération de `x_local`

Arrêt

Erreur commise : $\epsilon_k = \|X^{(k)} - X\|_\infty$
En remarquant qu'en fin d'itération, `x_local` correspond à $X^{(k+1)}$ et `x_global` à $X^{(k)}$, on va plutôt calculer :

$$\begin{aligned}
\mu_k &= \|X^{(k+1)} - X^{(k)}\|_\infty \\
&= \|A'X^{(k)} + X - A'X - X^{(k)}\|_\infty \\
&= \|A'(X^{(k)} - X) - (X^{(k)} - X)\|_\infty \\
\mu_k &\geq \left| \|A'(X^{(k)} - X)\|_\infty - \epsilon_k \right|
\end{aligned}$$

En utilisant $\rho(A') < 1$

$$\mu_k \geq \epsilon_k$$

Arrêt

- à chaque coefficient `x_local[i]` on associe un booléen `residues[i]`
- si `residues[i] = false`, $x_i^{(k+1)} = x_i^{(k)}$
- chaque processus vérifie si au moins une valeur de `residues` est à `true`
- ce résultat est envoyé au process 0, qui combine le tout et répond un ordre de continuer ou d'arrêter `run`

Communication entre les processus

Deux modèles de communication :

- synchrone, où chaque opération est bloquante. On utilise alors `MPI_Send` et `MPI_Recv`
- asynchrone, où les opérations de communications ne sont pas bloquantes, et l'on doit utiliser d'autres fonctions pour savoir lorsqu'elles ont eu lieu. Les fonctions sont `MPI_Isend` et `MPI_Irecv`. Pour resynchroniser les processus, on utilise `MPI_Barrier`

Communication entre les processus

Trois modes possibles en fonction du 4e argument `argv[4]` :

- 0 : synchrone
- 1 : asynchrone, avec `MPI_Barrier`
- 2 : asynchrone, sans `MPI_Barrier`. Dans ce cas le programme s'arrête après 5000 itérations.

Résultats

- Exemples donnés de tailles 4, 20, 300, 5000
- Pour des petites tailles ($n \leq 300$) le temps d'exécution reste constant, quelque soit le nombre de processeurs, la taille, ou l'asynchronisme
- asynchrone, sans `MPI_Barrier` pose de nombreux problèmes :
  - utilisation de la mémoire énorme, dûes aux nombreuses requêtes en cours mais non finies
  - le calcul est plus rapide que les requêtes, et à chaque itération les coefficients ne sont donc pas correctements mis à jour. Le vecteur semble tout de même commencer à converger.