# Projet : Parallélisation de la Méthode de Jacobi pour la résolution de systèmes linéaires

Plessia Stanislas

Mars 2018

# Méthode de Jacobi

# Définition du problème

Soit :

- $n \in \mathbb{N}$
- $A = (a_{i,j})_{i,j \in [\![1,n]\!]^2}$ matrice carrée de taille $n$
- $b = (b_i)_{i \in [\![1,n]\!]}$ vecteur de taille $n$.

On cherche alors le vecteur $x = (x_i)_{i \in [\![1,n]\!]}$ tel que :

$$Ax = b \tag{1}$$

# Résolution : Decomposition de $A$

On décompose $A$ en deux matrices :

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

Notons $D$ la matrice diagonale et $R$ le reste. Comme on suppose les coefficiens diagonaux non nuls, $D$ est trivialement inversible et on peut transformer l'équation (1) :

$$x = D^{-1}(b - Rx)$$

# Résolution : Solution Itérative

Un solution itérative peut être construite de la manière suivante :

$$\begin{cases} x^{(0)} = \vec{0} \\ x^{(k+1)} = D^{-1}(b - Rx^{(k)}) \end{cases}$$

D'après le théorème du point fixe, on peut montrer qu'on a alors :

$$\lim_{k \to \infty} x^{(k)} = x \Leftrightarrow \rho(-D^{-1}R) < 1$$

On obtient alors la formule de récurrence sur les coefficients :

$$\forall k \in \mathbb{N}, \forall i \in [\![1, n]\!], \qquad x_i^{(k+1)} = \frac{1}{a_{i,i}}(b_i - \sum_{i \neq j} a_{i,j} x_j^{(k)}) \qquad (2)$$

# Résolution : Condition suffisante de convergence

Une condition suffisante pour assurer la convergence de la méthode de Jacobi est la suivante :
Soit $\lambda$ valeur propre de $C$ et $y_\lambda$le vecteur propre associé, on a :

$$\begin{aligned} |\lambda| \cdot ||y_\lambda||_\infty &= |\frac{1}{a_{i,i}}| \cdot |\sum_{j \neq i} a_{i,j} y_{\lambda j}| \\ &\leq |\frac{1}{a_{i,i}}| \cdot \sum_{j \neq i} |a_{i,j}| \cdot ||y_\lambda||_\infty \end{aligned}$$

Donc on a $\forall (i,j) \in [\![1,n]\!]^2 \quad |a_{i,i}| > \sum_{i \neq j} |a_{i,j}| \implies \rho(C) < 1$,
ie $C$ est a diagonale strictement dominante.

# Implémentation de la Méthode

# Structures de données et Librairies

- Structures de données dynamiques (Matrix, Vector)
- Implémentées dans des librairies séparées
- Set de fonctions utilitaires pour chaque structures :
  - `build_structure`
  - `display_structure`
  - `randomize_structure`
  - `free_structure`
  - `read_structure_from_file`

# Structures de données et Librairies

Les structures de données sont implémentées par des types complexes contenant des données de taille (en entier non signés) et un tableau contenant les données de calcul.

Par exemple pour la structure matrice :

```
typedef struct Matrix{
    unsigned int col;
    unsigned int rows;
    double **matrix;
}Matrix;
```

# Programme principal

Pour paralléliser le programme, nous devons distribuer les données dans les différent processeurs :

Soit $n$ la taille du problème et $p$ le nombre processeurs. Pour le processus $i$, en notant $q = \lfloor \frac{n}{p} \rfloor$ et $r = \text{mod}(n, p)$ :

- Nombre de lignes $N$ : $q + 1$ si $i < r$, $q$ sinon ;
- Première ligne $F$ : $(q + 1)i$ si $i < r$, $q \cdot i$ sinon ;

# Programme principal

- L'équation (2) n'a pas de dépendance verticale sur la matrice, elle est donc facilement parallélisable
- Les processeurs auront 2 variables locales : `local_result` de taille $N$ et `global_result` de taille $n$ contenant $x^{(k)}$
- Au début d'une itération, chaque processeur communique son `local_result` pour remplir les `global_result` nécessitant d'être complet pour l'itération suivante.
- On applique ensuite la formule de récurrence pour mettre à jour `local_result`

# Convergence

Soit $\epsilon^{(k)} = ||x - x^{(k)}||_\infty$ et $e^{(k)} = ||x^{(k+1)} - x^{(k)}||_\infty$

On a :

$$\begin{aligned} e^{(k)} &= ||x^{(k+1)} - x^{(k)}||_\infty \\ &= ||Cx^{(k)} + d - x^{(k)}||_\infty \\ &= ||Cx^{(k)} + x - Cx - x^{(k)}||_\infty \\ &= ||(-C + I)(x - x^{(k)})||_\infty \end{aligned}$$

Finalement : $e^{(k)} = ||I - C||_\infty \epsilon^{(k)} \leq \epsilon^{(k)} + ||C||_\infty \epsilon^{(k)} \leq 2\epsilon^{(k)}$

On peut donc utiliser $e^{(k)}$ comme critère d'arrêt pour l'algorithme.

# Arrêt

Pour décider de l'arrêt du programme, nous avons une condition pour chaque processeur qui une fois rempli renvoi un booléen au processeur root :

- chaque processeur a un tableau de booléens `continue_iterate` de taille $N$
- quand une ligne a atteind la convergence ($e_i^{(k)} = x^{(k+1)}[i] - x^{(k)}[i] \leq 1.10^{-6}$) l'indice $i$ du tableau est inversée (true $\rightarrow$ false)
- On affecte la variable `local_continue_to_process` $| =$ `continue_iterate[i]` $\forall i$
- run $| =$ local_continue_to_process $\forall$ processeur

# Résultats

# Performances

| nb_processor | size = 7 | size = 20 | size = 200 | size = 1000 |
|---|---|---|---|---|
| 1 | 0,239 | 0,071 | 0,155 | 0,368 |
| 2 | 0,386 | 0,1 | 0,094 | 0,42 |
| 3 | 0,094 | 0,085 | 0,4 | 0,492 |
| 4 | 0,081 | 0,462 | 0,081 | 0,452 |
| 5 | 0,399 | 0,17 | 0,42 | 0,506 |
| 6 | 0,214 | 0,178 | 0,102 | 0,5 |
| 7 | 0,154 | 0,336 | 0,402 | 0,49 |
| 8 | NAN | 0,241 | 0,199 | 0,512 |
| | | | | |
| Iterations | 22 | 11 | 16 | 17 |

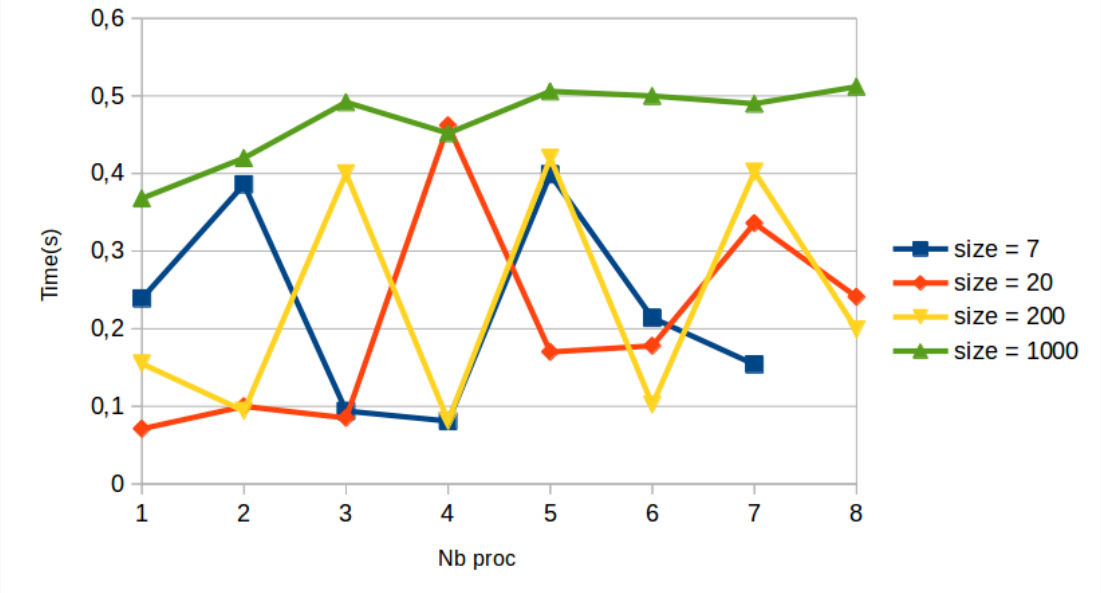


**Figure 1** – Tableau des performances du programme en fonction du nombre de processeur et des dimensions du problème

# Limites et Améliorations

- Données essayées non représentatives : Matrices de tailles supérieur à 7 générées par un script qui génère des matrice peu denses
- Le programme semble tourner en moyenne aussi rapidement peu importe la taille du problème
- Pas d'analyse de l'impact mémoire
- Désynchronisations possibles, ne semblent pas arriver sur des petites dimensions, non vérifé sur des grandes dimensions