Projet de programmation CHP
Différences finies pour l'équation d'advection avec MPI

N. Barral (nicolas.barral@enseirb-matmeca.fr), H. Beaugendre (heloise.beaugendre@enseirb-matmeca.fr),

1 Présentation du problème

On considère l'équation d'advection scalaire linéaire sur un domaine 2D $\Omega = [x_{min}, x_{max}] \times [y_{min}, y_{max}]$ et un intervalle de temps $[0, T]$, avec conditions aux bords périodiques:

$$\begin{cases} \partial_t u(x,y,t) + \mathbf{v}(x,y,t) \cdot \nabla u(x,y,t) = 0 \\ u(x_{min}, y, t) = u(x_{max}, y, t) \quad \forall y, t \\ u(x, y_{min}, t) = u(x, y_{max}, t) \quad \forall x, t \end{cases} \tag{1}$$

On discrétise le domaine spatial Ω avec une grille cartésienne avec N_x sous-intervalles de taille dx sur l'axe des x et N_y sous-intervalles de taille dy sur l'axe des y. La Figure 1 détaille les notations et numérotations utilisées pour cette discrétisation. Le domaine temporel est subdivisé de manière régulière avec un pas de temps dt.

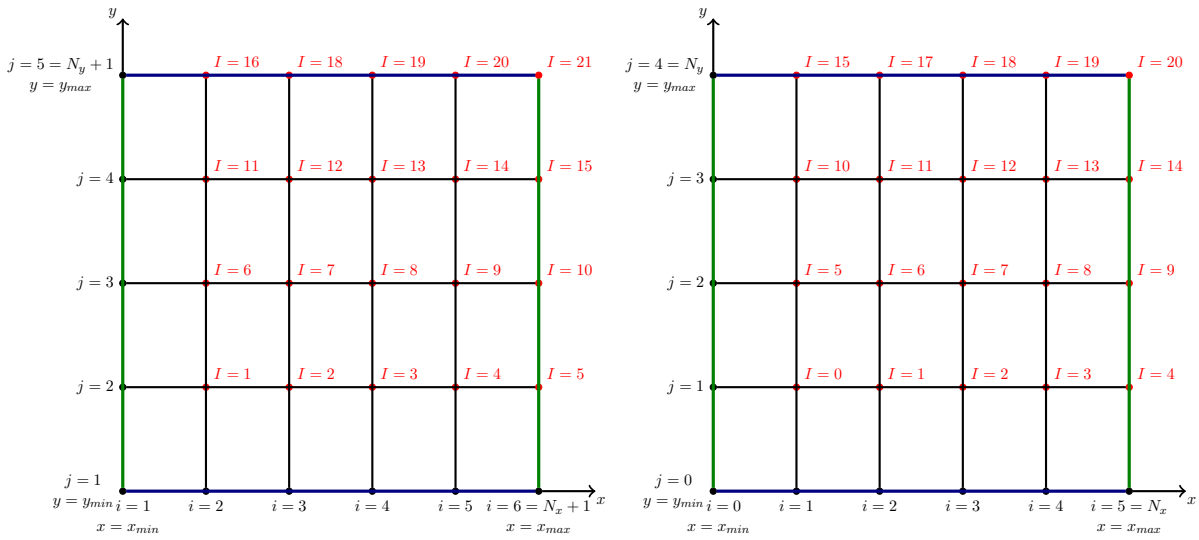

Figure 1: Discrétisation du domaine Ω par une grille cartésienne. En rouge, on numérote les inconnues. Comme les conditions de bord sont périodiques (les deux bords bleus (resp. verts) sont égaux), on choisit de ne pas compter les bords gauches et bas comme des inconnues. A gauche, numérotation en Fortran, à droite en C/C++.

L'objectif du projet est d'écrire un code parallèle utilisant MPI pour résoudre l'Equation 1 par différences finies sur un domaine carré discrétisé comme dans la Figure 1. On considérera les schémas centrés et upwind en espace, et explicite (avec matrice)

et implicite en temps. Pour le schéma implicite, une inversion de matrice est nécessaire. On utilisera la méthode du Gradient Biconjugué Stabilisé [1].

2 Analyse du problème

3 Implémentation

3.1 Séquentielle

1. Cahier des charges pour le nom des variables : `cas` un entier contenant le numéro du cas, `xmin`, `xmax`, `ymin`, `ymax` les bornes du domaine Ω, `Tf` le temps final, `Nx` le nombre de noeuds dans la direction x, `Ny` le nombre de noeuds dans la direction y, `dx`, `dy` les pas d'espace, `dt` le pas de temps, `vx`, `vy` les composantes de la vitesse d'advection, `space_scheme` un entier indiquant le schéma en espace: 1 pour centré, 2 pour upwind, etc., `time_scheme` un entier indiquant le schéma blahen temps: 1 pour explicite, 2 pour implicite, etc., `U` le vecteur solution au temps t^n, `U0` le vecteur solution au temps t^{n-1} si nécessaire.

2. `cas`, `xmin`, `xmax`, `ymin`, `ymax`, `Tf`, `Nx`, `Ny`, `dx`, `dy`, `dt`, `space_scheme` et `time_scheme` seront lues dans un fichier de paramètres ou dans la ligne de commande *dans cet ordre.*

3. Construire un module / un fichier .c séparé initialisant les paramètres, en lisant un fichier le cas échéant.

4. Construire un module / un fichier .c séparé contenant les fonctions : $v_x(x, y, t)$, $v_y(x, y, t)$ et la solution analytique $u(x, y, t)$ (qui est au moins toujours connue à $t = 0$).

5. Construire un module / un fichier .c séparé contenant l'assemblage de la matrice A et/ou le produit matrice-vecteur et toutes les opérations d'algèbre linéaire.

6. A est une matrice creuse, on ne stockera pas toute la matrice.

7. Construire un module / un fichier .c séparé contenant le gradient conjugué.

8. Ecrire les resultats dans des fichiers textes `sol.ite.dat` où `ite` est le numéro de l'itération du schéma en temps. Le fichier est formé de trois colonnes : x, y et U.

Indication: lors de l'implémentation, vous pouvez commencer par coder les paramètres en dur au début du code. Pour les conditions aux bords, vous pouvez commencer par les ignorer en considérant des fonctions advectées non nulles "assez loin du bord".

3.2 Propositions de cas test

Les cas tests suivants pourront être utilisés. Vous pouvez construire d'autres cas tests en vous appuyant sur ceux-ci, il faudra alors soigneusement les documenter dans le code et dans le rapport.

[1] https://en.wikipedia.org/wiki/Biconjugate_gradient_stabilized_method

Translation d'une gaussienne

- $\Omega = [-1, 1] \times [-0.5, 0.5]$
- $T = [0, ...]$
- $\mathbf{v}(x, y, t) = (1, 0)^\top$
- $u_0(x, y) = e^{\frac{-(x-0)^2}{0.0075} + \frac{-(y-0)^2}{0.0075}}$

Translation d'un cylindre

- $\Omega = [-1, 1] \times [-1, 1]$
- $T = [0, ...]$
- $\mathbf{v}(x, y, t) = (0.5, 0.5)^\top$
- $u_0(x, y) = \begin{cases} 1 & \text{si} \quad \|(x, y)\|_2 \leq 0.4 \\ 0 & \text{sinon} \end{cases}$

Rotation d'un cylindre

- $\Omega = [-1, 1] \times [-1, 1]$
- $T = [0, ...]$
- $\mathbf{v}(x, y, t) = (-y, x)^\top$
- $u_0(x, y) = \begin{cases} 1 & \text{si} \quad \|(x, y)\|_2 \leq 0.4 \\ 0 & \text{sinon} \end{cases}$

Note: vous pouvez adapter le cas pour faire tourner une gaussienne.

3.3 Parallèle

1. Chaque processeur connaîtra une partie de `U` seulement : `U(iBeg:iEnd)`.
2. Vous vous insipirez du paritionnement vu en cours pour répartir les inconnues. Vous pouvez faire l'hypothèse que $iEnd - iBeg > N_x$
3. L'algorithme du BICGSTAB devra être parallélisé :
   - Identifiez les opérations nécessitant des communications.
   - Détaillez, dans le rapport, les communications pour chacun des processeurs.
   - Optimisez les communications.
4. Chaque processeur écrira sa partie de solution dans son fichier `sol.ite.00Me.dat` où le numéro du processus est écrit avec 3 chiffres (ex: 001, 005, 040, 134...). Pour le Fortran, on pourra utiliser la subroutine `Rename` proposée en annexe.

4 Pour aller plus loin

Vous pourrez considérer un schéma en temps de Crank-Nicholson et des schémas en espace d'ordre élevé. D'autres cas tests peuvent être considérés:

Disque de Zalesak

Déformation d'une bulle

- $\Omega = [0,1] \times [0,1]$
- $T = [0, ...]$
- $\mathbf{v}(x,y,t) == \begin{pmatrix} 2\sin^2(\pi x)\sin(2\pi y)\cos(2\pi\frac{t}{6}) \\ -\sin(2\pi x)\sin^2(\pi y)\cos(2\pi\frac{t}{6}); \end{pmatrix}$
- $u_0(x,y) = \begin{cases} 1 & \text{si} \quad \|(x,y) - (0.5, 0.75)\|_2 \leq 0.15 \\ 0 & \text{sinon} \end{cases}$

Vous pouvez également rajouter un terme de diffusion dans l'équation.

5 Rendu

1. Un rapport contenant
   - L'analyse mathématique du problème :
     - L'écriture du schéma numérique proposé.
     - La mise en forme matricielle du problème.
     - La description détaillée de la structure de la matrice et de ses propriétés.
   - Une description rapide de votre code, et de sa validation.
   - L'explication précise du parallélisme mis en oeuvre dans votre code contenant :
     - La description de votre répartition du travail (et donc des inconnues) entre les différents processeurs.
     - Les différents points de l'algorithme nécessitant des communications entre les processeurs.
     - Une description détaillée des communications réalisées (taille des messages, émetteur, récepteur, schéma le cas échéant).
   - Le processus de validation du code.
   - Les courbes de speed-up et d'efficacité de votre code parallèle et autres courbes que vous jugerez utiles pour l'analyse de votre code.
   - Une analyse **détaillée** des performances parallèles de votre code.
   - Une étude des performances globales de votre code (choix d'implémentation, choix des schémas, choix de parallélisme, etc.) et le compromis proposé entre précision du calcul et temps de calcul.
2. Le code parallèle documenté et commenté. Vous adjoindrez au code un fichier `README.txt` précisant comment compiler le code, et comment exécuter votre programme, en détaillant bien comment changer les différents paramètres du problème.

Annexe:

```
subroutine Rename(Me,name)
   implicit none
   integer ::  Me
   character*13 ::name
   character*3 ::  tn
   integer ::  i1,i2,i3
   i1 = Me/100
   i2 =( Me - 100*i1)/10
   i3 = Me - 100*i1 -10*i2
   tn = char(i1+48)//char(i2+48)//char(i3+48)
   name='sol'//tn//'.dat'
end subroutine Rename
```