TP MPI N°4 : Communication locale / collective
Méthode des différences finies appliquées à l'équation de la chaleur

1. Équation de la chaleur bidimensionnelle et la résolution numérique

Considérons l'équation de la chaleur suivante :

$$\begin{cases} \dfrac{\partial u}{\partial t}(t,x,y) - c\dfrac{\partial^2 u}{\partial x^2}(t,x,y) - c\dfrac{\partial^2 u}{\partial y^2}(t,x,y) = 0, \quad \text{pour } t \in]0,+\infty[\text{ et } x,y \in]0,1[\\ u(0,x,y) = u_0(x,y), \quad \text{condition initiale} \\ u(t,0,0) = u(t,1,0) = u(t,0,1) = u(t,1,1) = 0, \quad \text{conditions limites} \end{cases}$$

u(t,x,y) représente la température du point *(x,y)* au temps *t*. u_0 est la répartition initiale de température (exemple : la Figure 1b) et la température du bord reste à *0* au cours du temps. *c* est une constante.

On cherche les valeurs approchées de *u* sur les points du maillage 2D régulier du domaine *[0,1]* x *[0,1]*. Les points du maillage sont en (x_i, y_j) avec $x_i = ih_x$ et $y_j = jh_y$, *i,j=0, 1, …, n+1* (voir la Figure 1a). La méthode de différences finies avec le schéma numérique explicite (schéma d'Euler explicite) suivant sera utilisée pour calculer l'évolution de la température dans le temps, sur chaque point du maillage.

$$\begin{cases} \dfrac{u_{ij}^{k+1} - u_{ij}^{k}}{h_t} - c\dfrac{u_{i+1,j}^{k} - 2u_{i,j}^{k} + u_{i-1,j}^{k}}{(h_x)^2} - c\dfrac{u_{i,j+1}^{k} - 2u_{i,j}^{k} + u_{i,j-1}^{k}}{(h_y)^2} = 0, \quad i,j \in [1,n] \\ u_{0,j}^{k} = u_{n,j}^{k} = u_{i,0}^{k} = u_{i,n}^{k} = 0 \quad \text{pour } i,j \in [1,n] \end{cases}$$

i.e. : $u_{ij}^{k+1} = u_{ij}^{k} + c_x\left(u_{i+1,j}^{k} - 2u_{i,j}^{k} + u_{i-1,j}^{k}\right) + c_y\left(u_{i,j+1}^{k} - 2u_{i,j}^{k} + u_{i,j-1}^{k}\right)$, c-à-d. la valeur du point *(i,j)* au temps *k+1* dépend de sa valeur et celles de ses quatre voisins au temps *k*.

Stabilité du schéma : le schéma d'Euler explicite est stable sous condition β<1/2, où $\beta = \max\{c_x, c_y\} = \max\{ch_t / h_x^2, ch_t / h_y^2\}$.

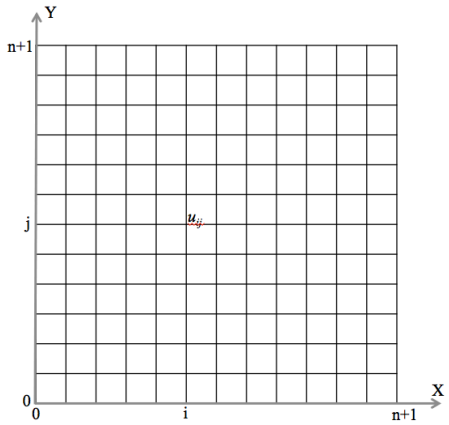

Figure 1 a. Maillage du domaine de calcul

b. u_0 généré par `initData()`, *n=78*

2. Programme séquentiel

Le programme séquentiel fourni contient une fonction d'initialisation de u_0 (`initData()`), une fonction `updateNLigs()` qui calcule l'évolution de la température sur les lignes spécifiées et les fonctions de sauvegarde de résultats dans un fichier (`saveData()` crée un nouveau fichier et `appendData()` ajoute en fin d'un fichier existant). On peut utiliser ces fonctions pour la sauvegarde des résultats, mais également pour leur vérification et leur comparaison du programme parallèle au programme séquentiel. La taille du maillage est égale à 80 par défaut. Elle peut être modifiée en exécution (avec 2 arguments sur la ligne de commande d'exécution).

Les paramètres impliqués dans le schéma d'Euler explicite doivent respecter la condition $\beta<1/2$, qui garantisse la stabilité de celui-ci. En programmation, les coefficients des quatre voisins directs sont implantés dans un masque de taille `3 x 3`. Un exemple est donné par la Figure 2, où $c_x = c_y = 0.1$, ce qui donne `MASK[]={0.0, 0.1, 0.0, 0.1, 0.6, 0.1, 0.0, 0.1, 0.0}`.

0	0.1	0
0.1	0.6	0.1
0	0.1	0

Figure 2. Masque des coefficients pour l'évolution de température

Deux versions du programme séquentiel sont fournies : l'une sans affichage graphique (`heat2D.c`) et l'autre avec (`heat2DXlib.zip`). La version graphique permet une vérification visuelle de l'évolution de température, mais sa parallélisation est plus délicate, donc plutôt réservée à ceux qui maitrise bien la programmation. Sachant qu'il est toujours possible d'utiliser les fonctions *saveData()* et *appendData()* pour vérifier la justesse de votre programme et comparer les résultats du calcul parallèle à ceux du calcul séquentiel.

3. Parallélisation --- Décomposition du domaine en blocs de lignes

Dans cette parallélisation, le processus 0 initialise u_0, décompose sans duplication le domaine du calcul par blocs de lignes, distribue un bloc à chaque processus (y compris lui-même), tous les processus font évoluer la température de son sous-domaine et envoient ses résultats au processus 0 à la fin de chaque étape du calcul. Ce rassemblement des résultats à chaque pas de temps permet au processus 0 de les sauvegarder dans un fichier (ou de les visualiser à l'aide du module graphique) et facilite la comparaison avec les résultats du programme séquentiel.

La Figure 3 montre la décomposition du domaine avec 4 processus. La température évolue uniquement sur les points intérieurs du maillage. Celle du bord reste à 0 au cours de la simulation.

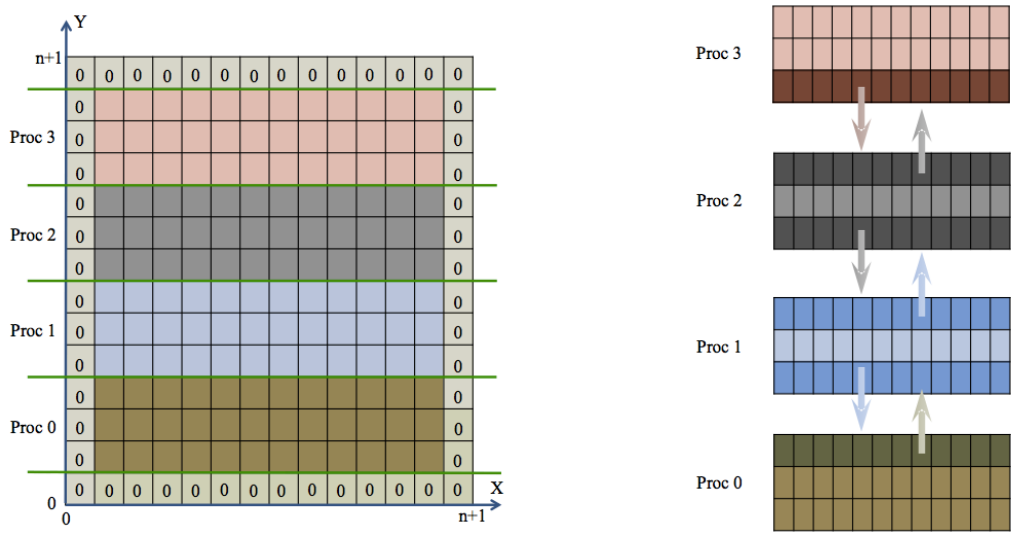

Figure 3. Décomposition du domaine et Communication entre les processus, nbProcs=4

La distribution sans duplication des données initiales oblige les processus voisins à s'envoyer les données du bord aux voisins directs avant chaque étape de calcul (ex. dans la Figure 3, le Proc 3 envoie sa dernière ligne au Proc 2 et le Proc 2 envoie sa première ligne au Proc 3). La communication entre les processus sera en premier temps réalisée à l'aide de la communication point-à-point.

Une fois que le programme parallèle fonctionne avec la communication point-à-point, on remplacera la distribution des données (u_0) et la collecte des résultats (u^l) par des communications collectives (*MPI_Scatter* et *MPI_Gather*).

Méthodologie de la programmation : procéder la parallélisation par étape et vérifier la justesse des résultats à chaque étape de programmation. Ici, on la découpe en 3 étapes :

- la distribution des blocs de lignes de u_0 : vérifier que les données reçues par chaque processus correspondent à celles distribuées par le processus 0,
- l'échange des données du bord avec les voisins et l'évolution de la température : vérifier que les données échangées sont correctes pour toutes les étapes,
- le rassemblement des résultats dans le processus 0.

Effectuer la mesure de performance de votre programme parallèle selon la même procédure vue en TP d'OpenMP, avec n suffisamment grand. Mesurer aussi le temps de calcul sans la synchronisation à chaque étape, que constate vous ?

Les fonctions de mesure du temps sont similaires à celles d'OpenMP :

- *double MPI_Wtime(void) ;* donne le temps en micro/nano secondes à partir d'un point donné. *double MPI_Wtick(void) ;* donne la résolution de l'horloge.
- Une synchronisation de tous les processus (à l'aide de *MPI_Barrier*) est nécessaire avant le 1er mesure du temps, afin que les processus démarrent leur mesure en même temps et que les temps d'exécution de tous les processus soient comparables.

4. Notes

Le même programme en une seule ou un nombre limité (d')itération(s) peut être utilisé dans les opérations de traitement d'image. On applique le masque correspondant selon l'opération de traitement d'image. On utilise un masque de taille 3x3, 5x5, etc et seuls les filtres linéaires peuvent être appliqués avec cette méthode. On peut citer quelques filtres suivants :

- Flottage ou lissage
    i. Filtre moyen :

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

    ii. Filtre gaussien : valeur plus grande au centre et proche du centre. Par exemple :

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

- Déplacement d'image : masque avec un seul 1 (pas au centre)

- Augmentation du contraste
    i. Filtre différence : on calcule la différence entre le pixel du centre et chacun de ses voisins, puis ajoute ces différences au pixel central. Exemple de masque :

$$\begin{bmatrix}-1 & -1 & -1\\ -1 & 9 & -1\\ -1 & -1 & -1\end{bmatrix}$$

- Détection de contour …