HPC - M1 CSMI

Convection-diffusion 2D sur grille régulière

On considère l'équation de convection-diffusion en deux dimensions sur le carré $[0,1]^2$:

$$\frac{\partial u}{\partial t} + \mathbf{v} \cdot \nabla u = D\Delta u,$$

où :

- $u = u(x, y, t)$ est l'inconnue,
- $\mathbf{v} = (v_x, v_y)$ est un champ de vitesse constant. On suppose que les deux composantes de la vitesse sont > 0.
- D est une constante de diffusion > 0.

On impose des conditions de Dirichlet homogènes : $u = 0$ sur le bord du domaine.

On discrétise l'espace à l'aide d'une grille régulière de taille $n \times n$ (espacement $h = 1/(n-1)$, $(x_i, y_j) = (ih, jh)$, $0 \leq i, j < n$) et le temps par un schéma explicite à pas de temps Δt.

La discrétisation spatiale est réalisée par :

- un schéma centré pour la diffusion (Δu),
- un schéma décentré amont pour la convection ($\mathbf{v} \cdot \nabla u$).

Questions :

1. Écrire le schéma de façon détaillée. Combien de vecteurs faut-il pour stocker la solution ?
2. Déterminer la condition CFL reliant Δt, h, v_x, v_y et D pour garantir la stabilité du schéma en norme L^∞.
3. Donner la complexité asymptotique du schéma en fonction de n pour atteindre un temps final T fixé (nombre d'opérations flottantes en fonction de n, ne pas compter les opérations de lecture/écriture en mémoire).

4. **(Modèle de temps de calcul)**

   (a) Estimer le temps d'exécution en supposant que tous les accès mémoire sont instantanés (ne comptabiliser que les opérations flottantes). On notera t_c le temps (supposé constant) d'une opération de virgule flottante (addition, multiplication, etc.).

   (b) Refaire l'estimation en supposant que chaque accès mémoire (lecture ou écriture) prend un temps fixe t_s.

   (c) Proposer un modèle simplifié avec cache :

   - Mémoire principale de grande capacité, accès lents, de temps unitaire t_s.
   - Cache unique de taille C, avec des accès rapides, de temps unitaire t_f.

   Décrire comment la présence d'un cache change le temps d'exécution attendu quand le maillage est balayé ligne par ligne. Pour simplifier les calculs, on supposera que n est grand devant C (une ligne de la grille ne rentre pas dans le cache). On négligera les variations de performances liées aux bords ainsi qu'au début et à la fin du calcul. On fera l'hypothèse que la mémoire cache est gérée par le système de façon très simple : quand le cache est plein, la dernière donnée lue remplace la plus ancienne.

5. **(Optimisation par tiling)**

   (a) Proposer un algorithme basé sur un *tiling* (blocage) de la grille pour améliorer la localité mémoire et l'efficacité du cache. On supposera que les tuiles sont de taille $b \times b$ et que n est un multiple de b.

   (b) Donner le pseudo-code du schéma temporel explicite avec tiling.

   (c) Estimer le gain en temps d'exécution en fonction des paramètres n, C, t_c, t_s, t_f, b. Là aussi, on fera des hypothèses simplificatrices : négliger les cas de lecture/écriture sur les bords des tuiles, gestion simple du cache par le système, $1 \ll b \ll n$, etc. En pratique, on constate que $t_c \ll t_f \ll t_s$. Montrer dans ce cas que le calcul est plus rapide d'un facteur 4.

6. **(Programmation)**

   (a) Implémenter le schéma explicite en `C++` **sans tiling**.

   (b) Implémenter également la version **avec tiling**.

   (c) Comparer expérimentalement les temps d'exécution obtenus pour différentes tailles de grille (n variant entre 512 et 8192). Ne pas forcément réaliser toutes les itérations en temps pour estimer les temps de calcul. On comparera les temps de calcul dans un tableau de la forme :

n	b	sans tiling	avec tiling
512	32		
1024	⋮		
2048			
⋮			

(d) Commenter les résultats que vous obtenez.

(e) S'il vous reste du temps, programmez et testez une version OpenMP du schéma. On imposera un nombre de threads OpenMP de 4 (avec la commande `export OMP_NUM_THREADS=4`).

Remarques :

- Vous pouvez supposer que $v_x, v_y = 1$.
- Vous pouvez choisir $D = 1$.
- Vous pouvez choisir une valeur de Δt légèrement inférieure à la limite de stabilité.
- Vous pouvez utiliser ce code comme point de départ :
  https://irma.math.unistra.fr/~helluy/hpc/grid2d.tgz
  Commande de compilation : `g++ *.cpp`