TP MPI: Équation de la chaleur

ENS-Rennes

L'objectif du TP est de paralléliser avec MPI la simulation de la diffusion de la chaleur dans une plaque 2D.

Objectifs pédagogiques:

- Écrire un programme de type stencil;
- Parallélisation par décomposition spatiale;
- Convergence d'algorithmes numériques itératifs;
- Échanges de données point-à-point en MPI;

La diffusion de chaleur est représentée par une équation aux dérivées partielles parabolique. On peut simuler cette diffusion en discrétisant l'espace (la température de chaque point de la plaque est un coefficient dans une matrice) et le temps (on calcule successivement les évolutions après chaque pas de temps). Une simplification grossière consiste à faire la moyenne de chaque case avec ses voisines à chaque itération. Bien sûr, une modélisation réaliste de la diffusion de la chaleur serait plus compliquée[1], mais la forme du calcul et la façon de le paralléliser seraient inchangées.

Les problèmes de cette catégorie sont connus comme des **stencils**: On applique à chaque case de la matrice une opération prenant les voisines en paramètre. Ce schéma de calcul est extrêmement courant puisqu'il permet donc de résoudre numériquement des équations aux dérivées partielles. Dans le cas présent, on parle d'un stencil à 5 points (car on utilise les quatre voisins nord, sud, est, ouest pour calculer une case donnée) en 2D (car les données sont matricielles), mais toutes sortes de variantes du schéma existent. Ce sont le plus souvent des algorithmes itératifs, où l'on applique le même traitement en boucle jusqu'à atteindre un **critère de convergence** correspondant à un état stationnaire du système.

En pratique ici, la température du point (x, y) à l'étape $n + 1$ est donnée par l'équation suivante:

$$M_{n+1}(x, y) = \frac{M_n(x + 1, y) + M_n(x - 1, y) + M_n(x, y + 1) + M_n(x, y - 1) + M_n(x, y)}{5}$$

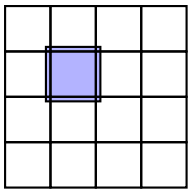
Les conditions aux bordures sont à préciser pour savoir quoi faire quand on n'a pas de voisin nord, par exemple. Pour simplifier encore, on considérera ici que les bords de la matrice représentent le milieu extérieur qui ne change pas de température.

Le critère de convergence à utiliser est le suivant: $\sum_{\forall x,y} |M_{n+1}(x, y) - M_n(x, y)| < \epsilon$ C'est là encore une simplification grossière puisqu'on utilise habituellement une distance au lieu de la différence, mais cela ne change rien à la forme du calcul.

À l'initialisation, la matrice d'entrée contient la même valeur partout, sauf sur la colonne centrale qui représente la source de chaleur. Si vous décidez de mettre des points chauds sur la bordure de la matrice qui ne change pas, veillez à le faire dans toutes les copies mémoires de la matrice.

▷ **Question 1:** Écrire la version séquentielle[2] de la diffusion de chaleur en 2D. Commencez par écrire une itération, vérifiez le résultat en l'affichant, puis faites plusieurs itérations. Allouer deux matrices en mémoire suffit pour toutes les étapes, en calculant de l'une dans l'autre pour les étapes paires, puis de l'autre dans l'une pour les étapes impaires.

Indiquez dans votre rapport les tests que vous avez effectué pour vous assurer que votre programme est correct. Il est suffisant d'indiquer les lignes de commande utilisées si les tailles de données y sont visibles. N'oubliez pas d'utiliser valgrind (en ajoutant `-vgdb` aux paramètres de `smpirun`) pour vous assurer que votre programme ne contient pas d'erreur simple.



Le schéma de parallélisation classique des stencils consiste à découper la matrice initiale en zones, puis d'attribuer chaque sous-domaine à un rank MPI donné. La difficulté se trouve aux frontières entre les sous-domaines: chaque rank a besoin d'un peu de données de ses voisins pour réaliser ses calculs. Chaque rank alterne donc les phases de calcul de son sous-domaine avec des phases de communication où l'on échange les bordures avec ses voisins. Les zones de données que chaque rank lit chez ses voisins sont appelées **ghost regions**.

Dans notre cas, on fera un découpage en grille, et on les *ghost regions* seront d'une seule ligne ou colonne puisqu'il s'agit d'un stencil 5 points seulement. Si l'opération nécessitait d'avoir plus les 24 voisins, il faudrait des fantômes de deux lignes ou colonnes.

[1] Pour une modélisation plus réaliste, voir par exemple `https://www-almasty.lip6.fr/~bouillaguet/static/hpc/TDTP4.pdf`. Le fichier `heatsink.c` décrit dans ce sujet se trouve dans le même répertoire du serveur.

[2] séquentiel = non-parallèle. Sans MPI, donc.

▷ **Question 2:** Parallélisez votre code en 2D avec une décomposition en grille. Pour simplifier, on supposera que le nombre de processus est une puissance de deux p, et que la taille des données est un multiple de p. Pour permettre au *runtime* MPI d'optimiser les communications, chaque rank postera ses communications en utilisant `MPI_Isend` et `MPI_Irecv` avant d'attendre la complétion de toutes les communications avec `MPI_Waitall`.

▷ **Question 3: (optionnelle)** Étendez votre code pour permettre des tailles de grilles quelconques de processeurs, et des tailles de données quelconques. On s'intéresse en particulier aux cas où la taille des données n'est pas exactement divisible par le nombre de processeurs, mais il n'est pas utile de traiter le cas où les données sont un rectangle non carré. Mesurez le déséquilibre de performance lié à la surcharge de travail des ranks en bout de ligne ou colonne.

▷ **Question 4: (optionnelle)** Ajoutez un paramètre en ligne de commande pour calculer la diffusion de la chaleur dans un volume 3D. Les ranks seront encore répartis selon une grille 2D et seront en charge de calculer toute une colonne du volume.

▷ **Question 5:** Mesurez les performances de votre programme quand le nombre de processus varie. Dans un premier temps, la taille des données est fixe et partagée entre les processus participants (**strong scaling**, vise à calculer plus vite une taille fixe). Dans un second temps, la part allouée à chaque rank est fixe et la taille totale varie (**weak scaling**, vise à calculer un problème plus grand à moyens constants).

▷ **Question 6: (optionnelle, mais recommandée:** Si vous faites une seule optionnelle, faites celle-là).
En utilisant SMPI, explorez ce qu'il se passe quand la vitesse du réseau augmente quand celle des noeuds de calcul diminue (ou le contraire). Il suffit de modifier le fichier décrivant la plate-forme virtuelle pour cela. Que constatez-vous?

▷ **Question 7: (optionnelle)** Mesurez le strong scaling obtenu sur votre machine en local, en utilisant le vrai MPI (sans SimGrid). Que constatez-vous? Pourquoi un tel effet bizarre?