ENSTA MS01 – Calcul scientifique parallèle – Année 2024-2025

TP 2 : Gestion des communications avec MPI

L'objectif de cette séance est d'apprendre à utiliser différentes commandes de communication de la librairie MPI. Les exercices seront réalisés à distance sur les stations de travail de l'école. Les codes de départ sont disponibles sur le site Internet du cours.

Exercice 1. Communications point-à-point non-bloquantes

Reprenez le code développé lors de l'exercice 2 du TP 1 *(Communications point-à-point bloquantes)*. Pour les cas ou vous observiez un *deadlock*, utilisez les fonctions de communication point-à-point non-bloquantes `MPI_Isend` et `MPI_Irecv` pour résoudre le problème. Testez et vérifiez vos solutions.

Exercice 2. Méthode du trapèze

On fournit un code `C++` de calcul séquentiel (`trapeze.cpp`) pour l'intégration d'une function $f(x)$ par la méthode des trapèzes :

$$I \;=\; \int_a^b f(x)\,dx \;\approx\; \Delta x \sum_{i=1}^{N} \frac{f(a+(i-1)\Delta x) + f(a+i\Delta x)}{2},$$

avec $\Delta x = (b-a)/N$.

Parallélisez ce code pour un nombre arbitraire de trapèze N et un nombre arbitraire de processus MPI. Le programme final doit calculer l'intégrale en parallèle pour un nombre N donné, et afficher la valeur totale de l'intégrale.

Une fois le code parallélisé, vérifiez si la version parallèle est plus rapide que la version séquentielle lors d'une exécution avec plusieurs processus MPI. Pour $N = 10^9$, analysez le temps de calcul obtenu en fonction du nombre de processus MPI *(testez 1, 2, 4, 8, 16, 32 et 64 processus)*.

Conseil : Pendant le travail de parallélisation du code, vérifiez régulièrement celui-ci donne toujours bien la bonne solution, et ***enregistrez les versions intermédiaires****. Le code parallélisé final doit donner rigoureusement la même solution que le code séquentiel. Il est inutile de paralléliser un code qui ne fonctionne plus.*

Exercice 3. Différences finies 1D

On fournit un code `C++` de calcul séquentiel (`FD1D.cpp`) pour la résolution de l'équation de la chaleur instationnaire 1D avec un schéma de différences finies :

$$\frac{u_n^\ell - u_n^{\ell-1}}{h_t} = \frac{u_{n-1}^{\ell-1} - 2u_n^{\ell-1} + u_{n+1}^{\ell-1}}{h_x^2}, \quad \text{pour } n = 1,\dots,N \text{ et } \ell = 1,\dots,L,$$

avec la donnée initiale $u_n^0 = \sin(n\pi/(2(N+1)))$ pour tout n et les conditions aux limites $u_0^\ell = 0$ et $u_{N+1}^\ell = 1$ pour tout ℓ. Les paramètres N et L doivent être donnés à l'exécution du code. La solution finale est écrite dans un fichier `FD1Drezu.dat`. On fournit également un code LaTeX qui génère un fichier `FD1Dvizu.pdf` avec un visuel de la solution finale.

Voici un exemple d'utilisation des fichiers :

```
>> g++ FD1D.cpp
>> a.out 100 200
>> pdflatex FD1Dvizu.tex
```

L'objectif de cet exercice est de paralléliser ce code pour un nombre N arbitraire et un nombre arbitraire de processus MPI. Il est recommandé de procéder par étape :

1. Si vous lancez le code séquentiel en parallèle *(i.e. avec* `mpirun` *et plusieurs processus)*, chaque processus fera l'ensemble du travail. Chaque processus aura donc une copie des tableaux `sol` et `solNew`, et écrira la solution finale dans un fichier. Modifiez le code pour que chaque processus écrive la solution finale dans des fichiers différents. *(Il suffit d'ajouter le numéro du processus dans le nom du fichier.)*

2. Ensuite, modifiez la boucle spatiale pour que chaque processus ne mette à jour qu'une partie des inconnues *(correspondant à un sous-domaine)*, et ajoutez les communications locales pour échanger les inconnues entre les processus *(i.e. aux interfaces entre les sous-domaines)*. *(Chaque processus alloue donc des tableaux de taille $N + 2$, mais ne modifie que les valeurs dont les indices vont de n_{start} à n_{end}.)*

3. Enfin, modifiez l'allocation et l'utilisation des tableaux `sol` et `solNew` sur chaque processus pour n'allouer que ce qui est nécessaire. *(Pour chaque processus, la taille de chaque tableau devrait être $n_{end} - n_{start} + 3$.)*

Conseil : Comme pour le dernier exercice, vérifiez régulièrement que le code sur lequel vous travaillez donne toujours bien la bonne solution, et enregistrez des versions intermédiaires.