Projet 2 : Problème stationnaire avec grille non-structurée

L'objectif de ce projet est d'utiliser un code d'éléments finis parallélisé, d'en étudier les performances sur le cluster gin, et d'analyser des solveurs de systèmes linéaires. Le code de calcul résout un problème non-structuré obtenu en discrétisant une équation elliptique avec une méthode d'éléments finis P^1.

Description du problème et du code de calcul

- ***Problème continu***

Étant donné un domaine $\Omega = [0,2] \times [0,2]$, on cherche la solution $u(\mathbf{x}) \in H^1(\Omega)$ du système

$$\begin{cases} \alpha u - \Delta u = f, & \forall \mathbf{x} \in \Omega, \\ \partial_n u = 0, & \forall \mathbf{x} \in \partial\Omega, \end{cases}$$

où α est un scalaire réel constant et $f(\mathbf{x}) \in L^2(\Omega)$. Ce problème peut se réécrire sous la forme variationelle

$$u \in H^1(\Omega) : \quad \alpha \int_\Omega uv \, d\Omega + \int_\Omega \nabla u \cdot \nabla v \, d\Omega = \int_\Omega fv \, d\Omega, \quad \forall v \in H^1(\Omega). \tag{1}$$

- ***Problème discrétisé***

Pour la résolution numérique, on utilise un schéma d'éléments finis basé sur un maillage de triangles $\mathcal{T}_h$, avec des éléments de Lagrange P^1. La solution numérique peut alors s'écrire

$$u_h(\mathbf{x}) = \sum_{i=1}^{N_{\text{nodes}}} w_i(\mathbf{x}) \, u_i,$$

où les u_i sont les inconnues discrètes et les $w_i(\mathbf{x})$ sont les fonctions de base globales. Ces fonctions sont continues, linéaires par morceaux, et telles que $w_i(\mathbf{x}_j) = \delta_{ij}$, pour $i, j = 1 \dots N_{\text{nodes}}$, où $\mathbf{x}_i$ est la position du $i^{\text{ième}}$ nœud du maillage. En utilisant cette représentation de la solution numérique et les fonctions de base comme fonctions test de la formulation (1), on obtient la formulation discrète

$$\alpha \sum_j \left[u_j \int_\Omega w_i w_j \, d\Omega \right] + \sum_j \left[u_j \int_\Omega \nabla w_i \cdot \nabla w_j \, d\Omega \right] = \sum_j \left[f(\mathbf{x}_i) \int_\Omega w_i w_j \, d\Omega \right],$$

pour $i = 1 \dots N_{\text{nodes}}$, où on a utilisé une projection de $f(\mathbf{x})$. Le système à résoudre peut se réécrire sous la forme abstraite

$$\underline{\underline{\mathbf{A}}} \, \underline{\mathbf{u}} = \underline{\mathbf{b}},$$

avec $\underline{\underline{\mathbf{A}}} = \alpha \underline{\underline{\mathbf{M}}} + \underline{\underline{\mathbf{K}}}$ et $\underline{\mathbf{b}} = \underline{\underline{\mathbf{M}}} \, \underline{\mathbf{f}}$. Les éléments de la matrice de masse $\underline{\underline{\mathbf{M}}}$ et de la matrice de rigidité $\underline{\underline{\mathbf{K}}}$ sont définis par

$$\left(\underline{\underline{\mathbf{M}}}\right)_{ij} = \int_\Omega w_i w_j \, d\Omega,$$
$$\left(\underline{\underline{\mathbf{K}}}\right)_{ij} = \int_\Omega \nabla w_i \cdot \nabla w_j \, d\Omega,$$

pour $i, j = 1 \dots N_{\text{nodes}}$.

La convergence de l'erreur L^2 de ce schéma est d'ordre 2, c'est à dire

$$\mathcal{E} = \|u_h - u_{\text{ref}}\|_{L^2(\Omega)} \leq Ch^2,$$

où h est le pas de maille et C est une constante indépendante de h. Cette erreur peut être évaluée numériquement en utilisant l'approximation d'une norme L^2,

$$\|v\|^2_{L^2(\Omega)} \;=\; \int_\Omega |v|^2 \, d\Omega \;\approx\; \underline{\mathbf{v}}^T \underline{\underline{\mathbf{M}}} \, \underline{\mathbf{v}}.$$

- ***Code de calcul***

On fournit un code de calcul C++ pour résoudre le problème décrit ci-dessus. Le fichier `main.cpp` de ce code effectue les opérations suivantes :

1. Initialisation de MPI;
2. Lecture du maillage et du partitionnement (fonction `readMsh`) et construction de listes de nœuds pour les communications (fonction `buildListNodesMPI`);
3. Construction de la solution de départ du solveur (u_0), de la solution de référence (u_{ref}), du terme source (f) et des matrices du système (fonction `buildLinearSystem`);
4. Résolution parallèle du système par la méthode de Jacobi (fonction `jacobi`);
5. Écriture de la solution dans un fichier gmsh (fonction `saveToMsh`);
6. Finalisation MPI.

Le calcul du résidu et de l'erreur L^2 de la solution ne sont pas encore implémentés. Pour accélérer le calcul, on utilise[1] la librairie eigen pour le stockage des matrices creuses et pour les opérations matricielles (produits matrice-matrice et matrice-vecteur).

Le logiciel libre gmsh est utilisé pour générer le maillage, qui est ensuite lu par le code de calcul. Pour générer le maillage, on utilise la commande

```
>> gmsh -2 -clmax 0.025 -clmin 0.025 -part 4 carre.geo -o carre.msh
```

où `0.025` est le pas de maillage h souhaité, `4` est le nombre de sous-domaines souhaité, `carre.geo` est le nom du fichier de géométrie, et `carre.msh` est le nom du fichier de maillage. Pour visualiser le maillage une solution numérique, on utilise la commande `gmsh carre.msh solNum.msh_*`. L'ensemble des fichiers de géométrie, maillage et solution se trouvent dans le dossier `benchmark/`.

La compilation du code est réalisée en utilisant le `Makefile` qui se trouve dans le dossier principal. Il suffit alors d'exécuter la commande `make`, qui crée l'exécutable `solver`.

Dans la procédure de résolution, le nombre de sous-domaine est choisi lors de la génération du maillage avec gmsh, alors que le nombre de processus MPI est choisi au moment de l'exécution du programme avec mpirun. Dans le code, le sous-domaine numéro n_{dom} sera pris en charge par le processus de rang ($n_{\text{dom}} \mod N_{\text{proc}}$), où N_{proc} est le nombre total de processus MPI.

[1] de façon aussi transparente que possible ...

Consignes

Ce que vous devez faire et analyser à la perfection ...

1. Ajoutez et parallélisez :
   - le calcul du résidu pour le critère d'arrêt de la méthode de Jacobi;
   - le calcul de l'erreur L^2 à la fin de la résolution du système.

   Pour ce faire, vous devrez implémenter une fonction pour calculer le produit scalaire de deux vecteurs en parallèle.
2. Validez la version parallèle du code en vérifiant la convergence de l'erreur L^2 pour un problème manufacturé (prenez par exemple $\alpha = 1$ et $u(\mathbf{x}) = \cos(\pi x)\cos(2\pi y)$). Vérifiez si les erreurs sont les mêmes pour les cas où le code est utilisé en séquentiel ou en parallèle.
3. Étudiez la scalabilité faible et la scalabilité forte du code parallélisé sur le cluster `gin`. Pour ce faire, prenez des maillages suffisament grands. Vous pouvez vous limiter à 32 sous-domaines.

Ce que vous pouvez faire pour aller un peu plus loin ...

A. Version avancée sur les solveurs numériques :
   (a) On fournit une fonction séquentielle `buildDirichletBC` pour imposer la condition de Dirichlet $u = u_{\text{ref}}$ sur $\partial\Omega$, à la place de la condition de Neumann. Pour l'utiliser, cette fonction doit être appellée après `buildLinearSystem`. Parallélisez et validez la fonction `buildDirichletBC`.
   (b) Implémentez la méthode du gradient conjugué en parallèle. Étudiez la convergence de la méthode, et comparez avec celle de la méthode de Jacobi pour différentes variantes du problème ($\alpha = -1$, 0 et/ou 1).
   (c) Implémentez et étudiez un préconditionneur.

B. Version avancée sur la programmation parallèle *(pour les plus intrépides)* :
   (a) Implémentez la parallélisation du code vue au cours théorique : les points d'interface entre sous-domaines sont gérés par le processus de rang 0, alors que les points intérieurs des sous-domaines sont gérés par les autres processus.
   (b) Comparez et discutez la performance des deux parallélisations.
   (c) Implémentez et validez la méthode de Gauss-Seidel avec la nouvelle parallélisation.

La version finale de vos **codes** et un **rapport écrit** *(pas de présentation orale)* doivent être envoyés ***le jeudi 7 décembre*** *(au plus tard)* aux deux encadrants du projet :

- axel.modave@ensta-paristech.fr
- nicolas.kielbasiewicz@ensta-paristech.fr