Projet 2 : Problème stationnaire avec grille non-structurée

Les objectifs de ce projet sont d'utiliser un code d'éléments finis P1 parallèle, d'y implémenter et de comparer des solveurs itératifs, et enfin d'étudier la performance parallèle du processus de résolution.

L'évaluation de ce projet se fera sur base de codes de calcul, d'une soutenance orale, et de slides. Les slides feront office de rapport et serviront de support pour la soutenance. Les slides non-présentés lors de la soutenance seront lu par les enseignants.

Une archive avec les codes de calcul *(finaux, nettoyés, fonctionnels; pas de version intermédiaire; pas de maillages; pas de fichiers résultats; sans le répertoire eigen)* et les slides *(finaux)* doivent être envoyés au plus tard le **dimanche 17 novembre 2024** aux deux enseignants. Pour des raisons pratiques, il n'y aura pas d'extension de cette date limite. *Attention, les examens commencent le lundi 20 novembre. Il n'est pas attendu que vous travailliez jusqu'à la dernière minute sur ce projet.*

Les soutenances auront lieu le **mercredi 20 et jeudi 21 novembre 2024**.

E-mails: nicolas.kielbasiewicz@ensta-paris.fr et axel.modave@ensta-paris.fr

Description du problème

- ***Problème continu***

Étant donné un domaine $\Omega =]0, a[\times]0, b[$, 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

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

- ***Problème discrétisé***

Pour résoudre ce problème, on utilise un schéma d'éléments finis P1 basé sur un maillage de triangles $\mathcal{T}_h$, dont nœuds sont notés $(M_i)_{i=1\dots N_{\text{nodes}}}$. Après approximation de la formulation variationnelle et utilisation de fonctions de base nodales, on obtient le système

$$\big[\alpha\mathbf{M} + \mathbf{K}\big]\mathbf{u} = \big[\mathbf{Mf}\big],$$

où les éléments des matrices $\mathbf{M}$ et $\mathbf{K}$ et des vecteurs $\mathbf{f}$ et $\mathbf{u}$ sont définis par

$$M_{ij} = \int_\Omega w_i w_j \, d\Omega, \qquad K_{ij} = \int_\Omega \nabla w_i \cdot \nabla w_j \, d\Omega, \qquad f_i = f(M_i), \qquad u_i = u_h(M_i),$$

pour $i, j = 1 \dots N_{\text{nodes}}$. La convergence de l'erreur en norme $\|\cdot\|_{L^2}$ du 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 du maillage et C est une constante indépendante de h. Cette erreur peut être évaluée numériquement en utilisant l'approximation d'une norme L^2,

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

Structure et utilisation du code

On fournit un code de calcul C++ qui résout 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 ($u_h^{(0)}$), de la solution de référence (u_{ref}), du terme source (f) et des matrices du système (fonction `buildProblem`);
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 code est parallélisé en suivant une stratégie *"par groupes d'éléments"*. La librairie eigen est utilisée pour le stockage des matrices creuses et pour les opérations matricielles (produits matrice-matrice et matrice-vecteur).

- ***Compiler le code***

La compilation du code est réalisée en utilisant le `Makefile` qui se trouve dans le dossier principal. Il suffit alors d'exécuter la commande suivante:

```
>> make
```

Cette commande va créer l'exécutable `solver` dans le répertoire principal.

- ***Générer et partitionner un maillage***

Les données d'un premier benchmark sont contenues dans le fichier `main.cpp` et dans le fichier de géométrie `benchmark/mesh.geo`. Le logiciel libre gmsh (`http://gmsh.info/`) est utilisé pour générer et partitionner le maillage *(avant l'exécution du programme)*, et pour visualiser la solution.

Pour générer et partitionner le maillage, on utilise la commande suivante dans le dossier `benchmark/` :

```
>> gmsh -2 -part 4 mesh.geo
```

où 4 est le nombre de sous-domaines *(à modifier)*. Cette commande génère le fichier de maillage `mesh.msh` dans le dossier `benchmark/`.

Sur les stations de travail de l'ENSTA, la version de gmsh par defaut pose actuellement problème. Utilisez une ancienne version en remplaçant '`gmsh`' par '`gmsh2.14`' dans le code ci-dessus.

Sur le cluster Cholesky, il est nécessaire de charger un module spécifique pour utiliser gmsh.

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

- ***Exécuter le code et visualiser la solution***

Pour lancer le calcul, il faut lancer l'exécutable dans le dossier `benchmark/` avec le nom du maillage en argument:

```
>> mpirun -np 4 ../solver mesh.msh
```

Les calculs de la norme 2 du résidu et de la norme L^2 de l'erreur ne sont pas encore implémentées.

Après l'exécution du code de calcul, il est possible de visualiser la solution si on travaille en local, ou si on télécharge les fichers de sortie qui ont été écrits dans le dossier `benchmark/`. Voici la commande à exécuter :

```
>> gmsh mesh.msh solNum.msh_*
```

Consignes

1. Ajouter et paralléliser :
   - le calcul de la norme 2 du résidu pour le critère d'arrêt de la méthode de Jacobi;
   - le calcul de la norme L^2 de l'erreur à la fin de la résolution du système.
2. Valider le code parallèle en vérifiant la convergence de la norme L^2 de l'erreur pour un problème manufacturé. Prendre par exemple $\alpha = 1$ et $u(\mathbf{x}) = \cos(\pi x/a)\cos(2\pi y/b)$.
3. Implémenter la méthode du gradient conjugué dans une nouvelle fonction. *(Ne dupliquez pas le code complet!)* Paralléliser et valider cette fonction.

Choisir une variante ...

→ *Variante A :*

4. Étudier et comparer la convergence des méthodes de Jacobi et du gradient conjugué sur un benchmark.
5. Étudier la scalabilité faible et la scalabilité forte du code parallélisé sur le cluster cholesky et comparer les performance parallèle des 2 méthodes (Jacobi et gradient conjugué).

→ *Variante B :*

4. Implémenter, paralléliser et valider les méthodes de Jacobi avec relaxation (JOR) et la méthode de la plus grande pente. Étudier et comparer la convergence des différentes méthodes implémentées (Jacobi, JOR, gradient et gradient conjugué).
5. Étudier la scalabilité forte du code parallélisé sur une station de travail de l'ENSTA et comparer la performance parallèle pour 2 méthodes (Jacobi et gradient conjugué).

Quelques recommandations pour les études de scalabilité :

- *Prendre des maillages suffisamment grands. Ces maillages ne devront pas être envoyés aux enseignants, mais les caractéristiques principales doivent être fournies (i.e. pas de maillage et nombre de nœuds du maillage).*
- *Pour l'évaluation des temps de calcul, considérer des temps de calcul par itération. Ignorer les phases d'initialisation, de finalisation et d'écriture de fichiers.*