projet3 GLCS : Projet

8/01/2021

Travail en binôme

Le projet sera réalisé en binômes. Vous avez le choix de la personne avec qui vous mettre en binôme, et votre choix devra impérativement être fait et signalé par mail le 8 janvier.

Certaines parties du projet (notées **[individuel (membre 1)]** et **[individuel (membre 2)]**) devront être réalisées par un des deux membres du binôme et seront notées indépendamment. D'autres parties (notées **[commun]**) devront être réalisées en commun par les deux membres du binôme.

Rendu du projet

Date : 28 janvier 2021

Votre projet devra être hébergé sur github. Si vous le souhaitez, vous pouvez rendre votre projet privé, mais dans ce cas, vous devez ajouter l'utilisateur `jbigot` aux membres ayant accès au projet.

Vous devrez rendre votre projet pour le 28 janvier au plus tard. Ce rendu devra être fait sous la forme de deux emails à l'adresse julien.bigot@uvsq.fr :

- un premier mail avec en fichier attaché une archive (format `tar.gz`) de votre projet ne contenant que les fichiers source, **pas d'exécutable ni de répertoire caché `.git` !** (l'archive générée par github convient par exemple très bien),
- un second mail comportant uniquement l'identifiant (SHA-1) du commit correspondant à cette archive.

Vos mails devront être adressés en CC au deux membres du binôme.

Mise en place

Comme pour les TDs 1 et 2, vous devrez au préalable avoir installé `docker` sur votre machine et autorisé votre utilisateur à le lancer. Sous Debian / Ubuntu, on obtient ce résultat avec les commandes :

```
$ sudo apt install docker.io
$ sudo gpasswd -a $USER docker
```

Le code du projet est hébergé dans le projet `jbigot/Projet-GLCS-2020-2021` sur github accessible à l'adresse https://github.com/jbigot/Projet-GLCS-2020-2021. Chaque binôme devra créer son propre projet sur github basé sur ces fichier. Le plus simple est d'utiliser le bouton "Fork" en haut à droite de la page du projet d'origine.

Vous trouverez le fichier d'environnement du projet à l'adresse https://raw.githubusercontent.com/jbigot/glcs_2020-2021/master/glcs-td3-projet.sh. Comme pour les TDs 1 et 2, son exécution peut être un peu lente ; elle télécharge tout l'environnement du projet.

```
$ wget https://raw.githubusercontent.com/jbigot/glcs_2020-2021/master/glcs-td3-projet.sh
$ bash glcs-td3-projet.sh <chemin>/<vers>/projet
```

Cette commande ouvre un shell qui tourne dans un environnement docker (container) qui comporte des bibliothèques, outils, etc... différents de ceux de votre machine personnelle. Contrairement aux TDs 1 et 2, la commande ne crée pas les fichiers du projet. Il est de votre responsabilité de les récupérer en utilisant `git`.

```
$ git clone git@github.com:<MY_USER>/Projet-GLCS-2020-2021.git
```

Si vous préférez, il vous est tout à fait possible de travailler directement sur votre machine personnelle sans utiliser l'environnement fourni. Attention cependant, l'évaluation se déroulera dans l'environnement fourni. **Il est donc impératif de tester votre projet dans cet environnement avant de le rendre !**

1 Description du code

1.1 Équation de la chaleur

Le projet met en œuvre un solveur pour l'équation de la chaleur :

$$\frac{\partial u}{\partial t} = \Delta u \tag{1}$$

Où u est une fonction du temps et de l'espace qui donne la température en un point donné à un instant donné.

Dans notre cas, on travaille dans un espace cartésien 2D. Typiquement, on simule la diffusion de la température sur une plaque métallique très fine. On a donc :

$$\begin{aligned} u \colon \mathbb{R} \times \mathbb{R}^2 &\to \mathbb{R} \\ (t, x, y) &\mapsto u(t, x, y). \end{aligned}$$

et

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}. \tag{2}$$

1.2 Méthode des différences finies

La méthode des différences finies propose une solution pour calculer la valeur d'une dérivée dans un système discrétisé par un pas h suffisamment petit :

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}. \tag{3}$$

La dérivation propre de cette méthode basée sur la formule de Taylor est bien expliquée dans l'article Wikipedia en anglais[1]. On se contentera ici de remarquer que la valeur proposée par la méthode ressemble fortement à la définition de la dérivée :

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}.$$

En choisissant $h' = -h$, on obtient la formule à gauche :

$$\begin{aligned} f'(x) &\approx \frac{f(x+h') - f(x)}{h'} \\ &\approx \frac{f(x-h) - f(x)}{-h} \\ &\approx \frac{f(x) - f(x-h)}{h}. \end{aligned} \tag{4}$$

Pour une dérivée seconde, on peut commencer par appliquer la formule à droite (3) :

$$f''(x) \approx \frac{f'(x+h) - f'(x)}{h},$$

puis à gauche (4) :

$$\begin{aligned} f''(x) &\approx \frac{\frac{f(x+h) - f(x+h-h)}{h} - \frac{f(x) - f(x-h)}{h}}{h} \\ &\approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}. \end{aligned} \tag{5}$$

On obtient ainsi la formule centrée pour la dérivée seconde.

1.3 Équation de la chaleur discrétisée

Pour l'équation de la chaleur 2D (2), on commence par appliquer la formule à gauche (4) pour la dérivée en temps avec $h = d_t$:

$$\begin{aligned} \frac{u(t+d_t, x, y) - u(t, x, y)}{d_t} &\approx \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \\ u(t+d_t, x, y) &\approx d_t * \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) + u(t, x, y) \end{aligned}$$

[1] https://en.wikipedia.org/wiki/Finite_difference_method

On applique ensuite la formule centrée (5) pour les dérivées en espace ($h = d_x$ et $h = d_y$) :

$$u(t+d_t,x,y) \approx u(t,x,y) + d_t \times \left(\frac{u(t,x+d_x,y) - 2u(t,x,y) + u(t,x-d_x,y)}{{d_x}^2} + \frac{u(t,x,y+d_y) - 2u(t,x,y) + u(t,x,y-d_y)}{{d_y}^2} \right).$$

On discrétise u en temps avec un pas d_t et en espace (d_x, d_y) pour obtenir sa version discrétisée $U_t(i,j)$:

$$\begin{aligned} U_{t+}(i,j) &= u(t+d_t, jd_x, id_y) \\ &= U_t(i,j) + d_t * \left(\frac{U_t(i+1,j) - 2U_t(i,j) + U_t(i-1,j)}{{d_y}^2} + \frac{U_t(i,j+1) - 2U_t(i,j) + U_t(i,j-1)}{{d_x}^2} \right). \end{aligned}$$

C'est cette équation qui est implémentée par la fonction `FinitediffHeatSolver::iter` du code qui vous est fourni avec U_{t+} stocké dans la variable `next` et U_t dans dans la variable `cur`. On parle d'un *stencil* car le calcul des valeurs à chaque coordonnée dépend des valeurs aux coordonnées voisines du pas de temps précédent.

1.4 Distribution de domaine et conditions aux limites

Le code est parallélisé avec MPI. Le domaine global est distribué en 2D dans l'ensemble des processus comme illustré sur la figure 1. Chaque processus est en charge d'un bloc de données identifié par le code couleur sur la figure. De plus, comme on est dans le cadre d'un code *stencil*, il est nécessaire pour calculer les valeurs aux frontières du domaine de définir des valeurs juste à l'extérieur du domaine. Dans le code on implémente des condition aux limites de Dirichlet en fixant ces valeurs à l'extérieur du domaine qui ne sont jamais modifiées. Dans le code fourni, la fonction `FixedConditions::initial_condition` positionne les conditions suivantes :

- conditions initiales : $U_0 = 0$ sur tout le domaine,
- conditions au limites en haut, en bas et à droite : conditions de Dirichlet, $U_t(i,j) = 0$,
- conditions au limites à gauche : conditions de Dirichlet, $U_t(i,0) = 2097.152$,

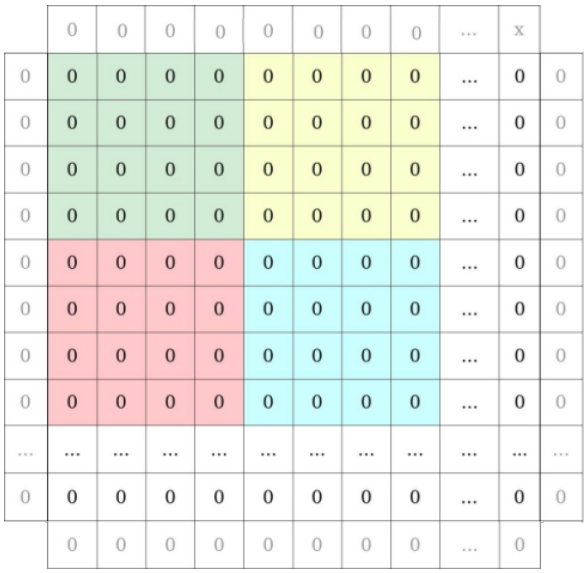

FIG. 1 : Domaine global distribué ; un processus (rang) MPI est en charge des données colorées en vert, un autre de celles en jaune, un autre de celles en rouge, un autre de celles en bleu, etc.

On retrouve ce problème si on s'intéresse au domaine local d'un processus tel qu'illustré sur la figure 2. Pour calculer les valeurs à la limite du domaine local, il faut accéder aux valeurs calculées par le processus voisin.

Pour gérer ces deux cas, on alloue un tableau plus grand que le domaine local. Le domaine inclue des zones fantômes. Pour les domaines au centre du domaine global, ces zones fantômes servent à stocker les valeurs calculées par les processus voisins communiquées par MPI. Pour les domaines à la limite du domaine global, ces zones fantômes stockent les valeurs au limites.

Cette distribution de domaine est implémentée par les classes `Distributed2DField` qui représente un bloc local de données et `CartesianDistribution2D` qui identifie la distribution du domaine local au sein du domaine global. La classe `Distributed2DField` expose des "vues" (*view*) sous la forme de fonctions membres :

- `full_view` donne accès aux données avec un système de coordonnées qui commence à 0 au début des zones fantômes,
- `noghost_view` donne accès aux données avec un système de coordonnées qui commence à 0 après les zones fantômes et n'expose que le domaine local,
- `ghost_view` donne accès aux données avec un système de coordonnées qui commence à 0 au début de chaque zone fantôme et donne accès à cette zone fantôme spécifique.

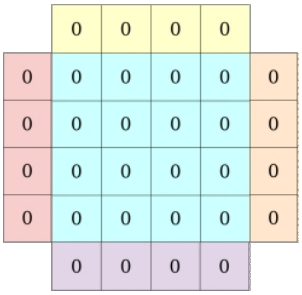

FIG. 2 : Domaine local (bleu) incluant des zones fantômes pour accéder aux valeurs calculées par le voisin de gauche (rouge), droite (orange), du haut (jaune) et du bas (violet).

La fonction `sync_ghosts` synchronise les zones fantômes entre processus voisins. Les communications MPI nécessaires à la synchronisation sont illustrées sur la figure 3.

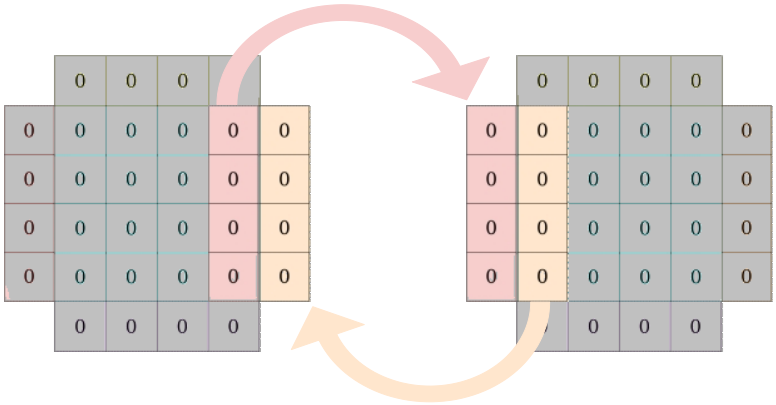

FIG. 3 : Communications pour synchroniser les zones fantômes entre deux voisins. Le domaine de gauche envoie sa colonne la plus à droite (en rouge) qui forme la zone fantôme de gauche du processus à droite. En échange et symétriquement, celui-ci envoie sa colonne la plus à gauche (orange) qui forme la zone fantôme de droite.

2 Objectif du projet

2.1 Prise en main [commun]

Étudiez le code pour en comprendre la logique globale. Exécutez le code sur un petit exemple pour en comprendre et vérifier le résultat, par exemple avec les paramètres suivant :

- `Nb_iter` = 10
- `height` = 4
- `width` = 8
- `process_height` = 1 ou 2
- `process_width` = 1 ou 2
- `delta_t` = 0,125
- `delta_x` = 1
- `delta_y` = 1

```
$ mpirun -n 2 ./simpleheat 10 4 8 1 2 0.125 1 1
```

2.2 Rapport [commun]

Vous devez rendre un rapport qui traite deux aspects.

- La première partie doit décrire le fonctionnement du code fourni, en particulier concernant la synchronisation des zones fantômes. Elle doit expliquer dans le détail le fonctionnement de la fonction qui vous est fournie dans ce but : `Distributed2DField::sync_ghosts`. Elle doit décrire la classe `CartesianDistribution2D` et expliquer en quoi consiste le concept de communicateur MPI cartésien utilisé dans cette classe.
- La seconde partie du rapport doit décrire les modifications que vous avez effectuées dans le code. Elle doit expliquer leur fonctionnement et décrire d'éventuelles limitations ou problèmes de vos modifications. Elle doit identifier et justifier les choix d'implémentation que vous avez fait.

2.3 Ajout de fonctionnalités au code

Vos modifications devront respecter l'architecture du projet proposé. Elles devront minimiser les modifications dans le code déjà fourni. Pour chaque nouvel ensemble de fonctionnalités, vous devrez fournir un nouvel exécutable les utilisant et vous devrez vous assurer que l'exécutable fourni continuent de fonctionner.

Vous veillerez à garder à l'esprit les objectifs de génie logiciel étudiés dans le cours. Vous serez évalués sur l'ensemble de ces critères, incluant notamment (en plus de la correction de la solution) :

- utilisation du gestionnaire de versions,
- qualité de la documentation et des commentaires, essayez de commenter l'interface de vos fonctions avec le format Doxygen, comme dans le code fourni,
- propreté et structuration du code,
- réutilisabilité et réutilisation de code,
- performances,
- etc.

2.3.1 Modification du système d'écriture des données [individuel (membre 1)]

Cette modification devra être fournie sous la forme d'une bibliothèque indépendante, comme `heatlib` *et* `simpleui`*.*

L'objectif est de fournir une nouvelle classe pour remplacer l'écriture des données sur la console mise en œuvre par la classe `ScreenPrinter`. En effet, `ScreenPrinter` propose une implémentation fragile inefficace et qui ne peut pas assurer que l'ordre des données sur la console soit bien respectée.

Vous devrez à la place fournir une nouvelle classe qui stocke les données au format HDF5. Les itérations auxquelles stocker les données devraient être configurable par l'utilisateur. Une ou plusieurs nouvelles options de configuration devront pour ça être ajoutées à la classe `Configuration`. Vous permettrez d'utiliser les données ainsi écrites comme conditions initiales d'une nouvelle simulation en proposant aussi un remplaçant pour la classe `FixedConditions`.

2.3.2 Modification du système de configuration [individuel (membre 2)]

Cette modification devra être fournie sous la forme d'une bibliothèque indépendante, comme `heatlib` *et* `simpleui`*.*

L'objectif est de remplacer la classe `CommandLineConfig` qui analyse les options passées sur la ligne de commande. En effet, la ligne telle que demandée est longue, peu lisible et l'ajout de nouveaux paramètres est rendue complexe par l'impossibilité d'avoir des valeurs par défaut.

Vous proposerez une nouvelle implémentation qui supporte des paramètres par défaut et qui permet d'avoir plus de lisibilité sur les paramètres que chaque valeur défini. Il est conseillé pour cele de baser votre nouvelle version au choix sur la bibliothèque `Boost.Program_options`[2] ou `Boost.PropertyTree`[3]. Vous pouvez utiliser une autre bibliothèque de votre choix, mais dans ce cas, pensez à bien préciser dans votre rapport la bibliothèque à installer.

2.3.3 Post-traitement en ligne des données [commun]

Cette modification devra être fournie sous la forme d'une bibliothèque indépendante, comme `heatlib` *et* `simpleui`*. En cas de manque de temps, vous devrez finir les deux modifications précédentes avant d'effectuer celle-ci.*

L'objectif de cette dernière fonctionnalité est d'ajouter une nouvelle classe permettant d'effectuer un post-traitement simple des données avant de les écrire. Pour réduire la quantité de données à écrire et ainsi pouvoir effectuer cette opération plus fréquemment sans surcharger le système de fichier on va appliquer une réduction aux données. Vous fournirez simplement une nouvelle classe qui calcule la moyenne globale des données avant de l'écrire au format HDF5. Comme pour les données brutes, vous déléguerez l'écriture à une classe dédiée, distincte de la classe responsable du calcul de la moyenne.

2.4 Soutenance [commun]

Vous devrez présenter votre rapport sous la forme d'une présentation de 15 minutes + 5 de questions qui aura lieu le 29 janvier. Vous vous appuierez sur des transparents pour votre présentation. Cette présentation devra reprendre le contenu de votre rapport et proposer une explication rapide du code en se focalisant particulièrement sur les fonctions d'échange de données et sur vos modifications. Vous prêterez attention à ce que le temps de présentation soit équilibré entre les membres du binôme.

[2] https://www.boost.org/doc/libs/1_74_0/doc/html/program_options.html
[3] https://www.boost.org/doc/libs/1_74_0/doc/html/property_tree.html