Modèles et techniques en programmation parallèle hybride et multi-cœurs

Projet 1

Marc Tajchman

CEA - DEN/DM2S/STMF/LDEI

mise à jour le 15/12/2025

Projet 1

On part d'un code parallélisé avec MPI qui calcule une solution approchée du problème suivant :

Chercher u*:* $(x,t) \mapsto u(x,t)$*, où* $x \in \Omega = [0,1]^3$ *et* $t \geq 0$*, qui vérifie :*

$$\begin{aligned} \frac{\partial u}{\partial t} &= \Delta u + f(x,t) \\ u(x,0) &= g(x) && x \in \Omega \\ u(x,t) &= g(x) && x \in \partial\Omega, t > 0 \end{aligned}$$

où f *et* g *sont des fonctions données.*

Le code utilise des différences finies pour approcher les dérivées partielles et découpe Ω en $n_0 \times n_1 \times n_2$ subdivisions.

Récupérer et décompresser un des fichiers `Projet1.tar.gz` ou `Projet1.zip`.

Les archives contiennent 2 répertoires Projet1/PoissonMPI et Projet1/PoissonMPI_OpenMP.

Chaque fois que vous développerez une nouvelle version, il faudra copier le répertoire Projet1/PoissonMPI_OpenMP dans un nouveau répertoire (par exemple : Projet1/PoissonMPI_OpenMP_grain_fin).

Vous modifierez uniquement les fichiers dans le nouveau répertoire, le répertoire d'origine Projet1/PoissonMPI servira de référence.

Structure du code

Le code est réparti en plusieurs fichiers principaux dans le sous-répertoire `src`:

main.cxx:	programme principal: initialise, appelle le calcul des itérations en temps, affiche les résultats
scheme(.hxx/.cxx):	définit le type `Scheme` qui calcule une itération en temps
values(.hxx/.cxx):	définit le type `Values` qui contient les valeurs approchées à un instant donné
parameters(.hxx/.cxx):	définit le type `Parameters` qui rassemble les informations sur la géométrie et le calcul

Fonctions du type `Scheme` :

`Scheme(P)`	construit une variable de type `Scheme` en lui donnant une variable de type `Parameters`
`iteration()`	calcule une itération (la valeur de la solution à l'instant suivant)
`variation()`	retourne la variation entre 2 instants de calcul successifs
`synchronize()`	copie les valeurs sur le bord d'un domaine vers les domaines voisins
`getOutput()`	renvoie une variable de type `Values` qui contient les dernières valeurs calculées
`setInput(u)`	rentre dans `Scheme` les valeurs initiales

Sur chacun des p sous-domaines de Ω (chaque sous-domaine est géré par un processus MPI)

Fonctions du type `Parameters` :

`imin(i)`	indice des premiers points intérieurs dans la direction i pour le sous-domaine courant
`imax(i)`	indice des derniers points intérieurs dans la direction i pour le sous-domaine courant
`imin_global(i)`	indice des premiers points intérieurs dans la direction i
`imax_global(i)`	indice des derniers points intérieurs dans la direction i
`dx(i)`	dimension d'une subdivision dans la direction i
`xmin(i)`	coordonnée minimale de Ω dans la direction i
`xmax(i)`	coordonnée maximale de Ω dans la direction i

`itmax()`	nombre d'itérations en temps
`dt()`	intervalle de temps entre 2 itérations
`neighbour(k)`	indice des sous-domaines voisins (1-2 à gauche ou à droite suivant X) (3-4 en arrière ou en avant suivant Y) (5-6 en bas ou en haut suivant Z) -1 si pas de voisin sur un côté du sous-domaine
`rank()`	indice du processus (= nombre de processus)
`size()`	nombre du processus (= nombre de sous-domaines)
`freq()`	fréquence de sortie des résultats intermédiaires (nombre d'itérations entre 2 sorties)

Pour un processus MPI P : les points de calcul à l'intérieur du sous-domaine Ω_p ont des indices (i, j, k) tels que:

```
imin(0)  ≤  i  ≤  imax(0)
imin(1)  ≤  j  ≤  imax(1)
imin(2)  ≤  k  ≤  imax(2)
```

Pour un processus MPI P : les points sur la frontière du sous-domaine $\partial\Omega$ ont des indices (i, j, k) tels que:

```
i = imin(0)-1  ou  i = imax(0)+1
j = imin(j)-1  ou  j = imax(1)+1
k = imin(k)-1  ou  k = imax(2)+1
```

Point frontière du sous domaine = point au bord du sous-domaine voisin ou point frontière du domaine global.

Fonctions du type `Values` pour le sous-domaine courant :

`init()`	initialise les points du domaine à 0
`init(f)`	initialise les points du domaine avec la fonction $f : (x, y, z) \mapsto f(x, y, z)$
`boundaries(g)`	initialise les points de la frontière avec la fonction $g : (x, y, z) \mapsto g(x, y, z)$
`v(i,j,k)`	si `v` est de type Values, la valeur au point d'indice (i, j, k)
`v.swap(w)`	si `v` et `w` sont de type `Values`, échange les valeurs de `v` et `w`

Compiler et exécuter le code de référence

- Pour compiler, se placer dans le répertoire `PoissonMPI` et taper:

  `./build.py`

  (si cela ne marche pas, taper `python ./build.py`).

- Pour exécuter sur N processus, rester dans le même répertoire et taper:

  `mpirun -n N ./install/Release/PoissonMPI.exe`

Pour voir les options d'exécution possibles, taper

`./install/Release/PoissonMPI.exe -h`

Noter les valeurs obtenues et les temps de calcul affichés, ils serviront de référence pour évaluer les autres versions.

Versions hybrides MPI-OpenMP (grain fin et grain grossier)

Créer deux versions du code, une pour la version hybride MPI + OpenMP grain fin et une pour la version MPI + OpenMP grain grossier.

Pour exécuter les codes hybrides sur N processus et T threads, taper (sur une seule ligne) :

```
mpirun -n N ./install/Release/PoissonMPI_OpenMP.exe
     --threads T
```

Si vous travaillez sur une machine avec (au moins) 8 cœurs, exécuter le code sur

- 1 processus MPI $\times$ 1 thread
- 1 processus MPI $\times$ 8 threads
- 2 processus MPI $\times$ 4 threads
- 4 processus MPI $\times$ 2 threads
- 8 processus MPI $\times$ 1 thread

Si non, exécuter le code sur

- 1 processus MPI $\times$ 1 thread
- 1 processus MPI $\times$ 4 threads
- 2 processus MPI $\times$ 2 threads
- 4 processus MPI $\times$ 1 thread

Et comparer les résultats et les temps de calcul.

Remarque

On rappelle que le nombre de processus MPI et de threads OpenMP est trop petit ici pour que la programmation hybride apporte un réel avantage par rapport au "tout MPI"

Envoyez par mail à marc.tajchman@cea.fr :

- une description du travail réalisé (1-2 pages maximum)
- le code source, avec vos modifications, dans une archive (n'envoyez pas les répertoires `build` et `install` qui contiennent des binaires, et riquent d'empêcher le message d'arriver)
- autant que possible, les fichiers qui contiennent les sorties écran

avant le 21/01/2026 minuit.

Envoyez vos fichiers source même s'ils contiennent des erreurs (ils peuvent ajouter des points à la note).