Modèles et techniques en programmation parallèle hybride et multi-cœurs

Travail pratique 2

Marc Tajchman

CEA - DEN/DM2S/STMF/LMES

mise à jour le 10/01/2021

Travail pratique 2

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.

Structure du code

Récupérer et décompresser un des fichiers `TP2_MPI.tar.gz` ou `TP2_MPI.zip`.

Se placer dans le répertoire `TP2/PoissonMPI` créé.
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, f)`	construit une variable de type `Scheme` en lui donnant une variable de type `Parameters` et une fonction `f` (second membre de l'équation)
`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:

$$\begin{array}{ccccc} \texttt{imin(0)} & \leq & \texttt{i} & \leq & \texttt{imax(0)} \\ \texttt{imin(1)} & \leq & \texttt{j} & \leq & \texttt{imax(1)} \\ \texttt{imin(2)} & \leq & \texttt{k} & \leq & \texttt{imax(2)} \end{array}$$

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`

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

  ```
  ./build.py
  ```

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

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

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

  Pour voir les options d'exécution possibles, taper `./install/Release/PoissonMPI.exe --help`

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

Version hybride MPI-OpenMP (grain fin)

Récupérer et décompresser un des fichiers

`TP2_MPI_OpenMP_FineGrain_incomplet.tar.gz` ou
`TP2_MPI_OpenMP_FineGrain_incomplet.zip`

Un répertoire `TP2/Poisson_MPI_OpenMP_FineGrain` est créé et contient le code source incomplet de la version MPI OpenMP grain fin.

Attention:

Si vous avez déjà récupéré ce fichier et modifié les sources qui y sont contenues, créez un autre répertoire ailleurs et travaillez dans ce nouveau répertoire, sinon vous écraserez les modifications déjà faites !

- Pour compiler, se placer dans le répertoire `TP2/PoissonMPI_OpenMP_FineGrain` et taper:

  ```
  ./build.py
  ```

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

- Pour exécuter, rester dans le même répertoire et taper (sur une seule ligne):

  ```
  mpirun -n <n> \
    ./install/Release/MPIPoissonOpenMP_FineGrain.exe
    threads=<t>
  ```

  Exemple: à la place de <n>, taper 2 pour exécuter avec 2 processus MPI, et à la place de <t>, taper 3 pour exécuter dans chaque processus MPI, 3 threads.

On fournit aussi un script `submit_hybrid.py` pour lancer les calculs sur la machine Gin (ce script devrait être facilement utilisable sur une machine parallèle gérée par SGE):

```
./submit_hybrid.py -n <n> -t <t>
```

où `<n>` est le nombre de processus MPI, et `<t>` le nombre de threads/processus MPI.

Exécuter le code sur

- 1 processus MPI $\times$ 8 threads
- 2 processus MPI $\times$ 4 threads
- 4 processus MPI $\times$ 2 threads
- 8 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 avantage par rapport au "tout MPI"

Version multithreads avec OpenMP (grain grossier)

Récupérer et décompresser un des fichiers

`TP2_MPI_OpenMP_CoarseGrain_incomplet.tar.gz` ou
`TP2_MPI_OpenMP_CoarseGrain_incomplet.zip`

Un répertoire `TP2/PoissonMPI_OpenMP_CoarseGrain` est créé et contient le code source incomplet de la version OpenMP grain grossier.

Pour compiler et exécuter on utilisera la même procédure que dans le cas OpenMP grain fin.

Dans le parallélisme OpenMP gros grain, on découpe explicitement le domaine de calcul en plusieurs parties, chaque partie est calculée par un thread.

Pour cela, le type `Parameters` possède 2 fonctions supplémentaires utiles pour le // gros grain :

`imin_thread(i,iThread)`	indice des premiers points intérieurs dans la direction i, pour la partie calculée par le thread `iThread`
`imax_thread(i,iThread)`	indice des derniers points sur la frontière dans la direction i, pour la partie calculée par le thread `iThread`

Ajouter une parallélisation OpenMP gros grain, et tester avec les mêmes nombres de processus MPI et threads OpenMP que dans le cas grain fin.

Le découpage de chaque domaine MPI en sous-domaines traités chacun par un thread est fait dans `parameters.cxx` lignes 143-169. Examiner et expliquer l'algorithme.

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)
- autant que possible, les fichiers qui contiennent les sorties écran:
  - si vous avez lancé les calculs sur Gin, les fichiers output_***
  - si vous avez lancé sur une autre machine, les fichiers similaires ou une copie des affichages écran

avant le 22/01/2021.

Envoyez vos fichiers source même s'ils contiennent des erreurs.