Travaux pratiques de

« Programmation hybride MPI-OpenMP »

Code fourni

Poisson : code de résolution approchée de l'équation de la chaleur sur un segment (dimension 1 d'espace), carré (dimension 2 d'espace) ou un cube (dimension 3 d'espace).

Le problème considéré est celui de l'équation de la chaleur :

$$-\frac{du}{dt}=\sum_{k=1}^{k=dim}\frac{d^2u_k}{dx_k^2}$$

avec conditions aux limites de Dirichlet et condition initiale $u(x,t=0)=f(x)$ où f est donné.

Le code est fourni :

- en version séquentielle (PoissonSeq) complète,
- en version «multi-threads » à mémoire partagée en utilisant OpenMP (PoissonOpenMP), à compléter
- en version « multi-processus » à mémoire distribuée en utilisant MPI (PoissonMPI), en utilisation un découpage du domaine complet en sous-domaines, chacun géré par un processus MPI et avec des échanges d'informations entre les sous-domaines voisins, à compléter.

Installation

Au préalable, taper les commandes :

```
export VTK_DIR=/home/t/tajchman/local/lib/cmake/vtk-6.3
export LD_LIBRARY_PATH=/home/t/tajchman/local/lib:${LD_LIBRARY_PATH}
```

Pour installer chacune des 3 versions :

- décompresser l'archive correspondante (`PoissonSeq.tgz, PoissonOpenMP.tgz, PoissonMPI.tgz` selon la version),
- entrer dans le répertoire de la version (`PoissonSeq, PoissonOpenMP, PoissonMPI` selon la version),
- créer le sous-répertoire vide « `build` » et se placer dans ce sous-répertoire,
- taper :

```
cmake ..
```

- taper :

```
make
```

Utilisation

Pour exécuter une version du code, se placer dans le répertoire `build` de cette version et tapez la commande suivante :

- `./PoissonSeq n=200 m=200 p=200 it=100 out=5`
  (cas de la version séquentielle)
- `./PoissonOpenMP n=200 m=200 p=200 it=100 out=5 threads=3`
  (cas de la version OpenMP)
- `mpirun -n 2 ./PoissonMPI n=200 m=200 p=200 it=100 out=5`
  (cas de la version MPI)

Le paramètre « `n=` » permet de spécifier le nombre de points de discrétisation suivant x (« `m=` » suivant y et « `p=` » suivant z). Pour changer la taille du problème, modifier les entiers qui suivent « `n=` » (resp. « `m=` », « `p=` »)

Dans la ou les versions utilisant OpenMP, le paramètre « `threads=` » permet de spécifier le nombre de threads à utiliser.

Dans la ou les versions utilisant MPI, le nombre de processus est un paramètre du lanceur « mpirun »

L'option « -h » ou « --help » affiche la liste des paramètres possibles pour chaque version du code.

Le nombre de pas de temps est fixé par défaut à 10 et le code ne sauve aucun résultat. Ceci est modifiable par l'utilisateur (se référer à ce qu'affiche l'option -h ci-dessus).

Lors de l'éxecution d'un cas avec le paramètre out=, il y a un affichage graphique et un répertoire est créé dont le nom dépend des paramètres entrés par l'utilisateur (par exemple « `results_n_NxMxP_p_P1xP2xP3_date` ») qui contient une copie de l'affichage et des sauvegardes du code.

Fichiers sources

Dans le répertoire de base de chaque version se trouvent les fichiers sources C/C++ :

```
main.cpp                      (programme principal)
Poisson.cpp/.hpp              (calcul)
Poisson_Parameters.cpp/.hpp (gestion des parametres)
values.cpp/.hpp               (valeurs de calcul)
plot.cpp/.hpp                 (affichage graphique)
```

```
timer.cpp/.hpp              (temps d'execution)
osutils.cpp/.hpp            (utilitaires)
version.hpp
```

ainsi qu'un fichier de configuration de la compilation par cmake (`CMakeLists.txt`).

Les fichiers sources contiennent

- `main.cpp` :
  le programme principal qui appelle la fonction d'initialisation, lance la boucle en temps, appelle les fonctions d'affichage
- `Poisson_Parameters.cpp`:
  la classe C++ qui interprète les paramètres de la commande tapée par l'utilisateur et initialise la simuation. Cette classe est utilisée dans le reste du code quand on a besoin d'une information (nombre de subdivisions, découpage en sous-domaines, etc)
- `Poisson.cpp` :
  les fonctions de calcul (schéma en temps : $u(t_n) \rightarrow u(t_{n+1})$), échange d'information aux frontières des sous-domaines)
- `values.cpp` :
  classe C++ qui contient les inconnues du problème (discrétisation de u), permet de récupérer/modifier les composantes et sauve aussi les valeurs sur fichier (pour visualisation éventuelle), effectue les échanges entre sous-domaines (dans le cas des version MPI)
- `timer.cpp` :
  classe C++ qui permet de mesurer le temps de calcul utilisé pour exécuter un groupe d'instructions du code

Test de la version séquentielle

Se placer dans le répertoire build de la version séquentielle.

Exécuter le code avec différentes tailles de problèmes, par exemple

```
100 x 100 x 100
400 x 400 x 400
1000 x 1000 x 1
1000 x 1 x 1000
1 x 1000 x 1000
```

Pour les 3 derniers exemples, comparez les temps de calcul

Tests sur les versions parallèles

Pendant les modifications et la mise au point des versions OpenMP et MPI, on conseille dans un premier temps de lancer la tests sur une machine locale (celle où vous travailler).

Pour les mesures de performances, vous pouvez lancer les exécutions sur une machine de calcul à laquelle vous avez accès (cluster ou autre).

Travail sur la version OpenMP

1. Examiner le code source de cette version, identifier le ou les parties du code où l'utilisation de OpenMP pourrait améliorer le temps calcul.

2. Faire une copie du code OpenMP initial. Utiliser sur cette copie, la programmation OpenMP type « fin grain » (voir cours).

3. Tester à nouveau sur les mêmes tailles que celles utilisées sur la version séquentielle. Faire varier le nombre de threads (1,2, …, nombre de cœurs sur la machine de calcul).

   Ne pas oublier de faire un test avec 1 seul thread

4. Expliquez les résultats.

5. Faire une copie du code OpenMP initial. Utiliser sur cette copie, la programmation type « gros grain » (voir cours)

6. Refaire les tests et comparer.

Travail sur la version MPI

1. Le code source de la version MPI est incomplet (rechercher les lignes « a modifier ... » dans le code source).

2. Tester sur les mêmes tailles de problèmes que dans les points précédents, en spécifiant différents nombres de processus (1, 2, 3, …)

   N'oubliez pas de tester le cas de la version MPI sur 1 processus.

Création de versions hybride MPI-OpenMP

On envisagera ici que le cas où seul le thread principal fait des appels MPI

1. Combiner la version MPI avec la version OpenMP « fin grain » pour produire une version hybride.

2. Faire de même en combinant la version MPI avec la version OpenMP «gros grain ».

3. Tester en faisant varier à la fois le nombre de processus et le nombre de threads