TP 3. Programmation multi-threads (2)

Préparation

Récupérer l'archive compressée `TP3.tar.gz` sur le site web

`https://perso.ensta-paris.fr/~tajchman/TP3`

et extraire les fichiers qui sont contenus dans cette archive :

```
cd <repertoire dans votre espace de travail>
tar xvfz TP3.tar.gz
```

Se placer dans le répertoire `TP3` :

```
cd TP3
```

et préparer les compilations dans les points suivants en tapant :

```
./build.sh
```

Chaque fois que vous modifiez un fichier source, tapez cette commande pour mettre à jour les exécutables.

1 Exemple OpenMP "grain fin"

Le fichier `src/sinus_fine/sinus.cxx` contient la version OpenMP "grain fin" obtenue à la fin du TP précédent.

Demonstration 1.

Comparer les temps d'exécution de la version séquentielle et de cette version en tapant les commandes

```
time -p ./install/sinus_seq 40000
time -p OMP_NUM_THREADS=3 ./install/sinus_fine 40000
```

Expliquer pourquoi on n'atteint pas le facteur de diminution du temps calcul espéré.

2 Exemple OpenMP "grain fin (peut-être) amélioré"

Question 1.

Ajouter aux pragmas OpenMP "for" de la version "grain fin", l'option

`schedule(static, n/12).`

Compiler, exécuter et comparer les temps d'exécution avec les versions précédentes.

Faire varier le paramètre `n/12`.

Remplacer dans l'option schedule, le paramètre `static` par `dynamic`.

Refaire les comparaisons en temps d'exécution.

3 Exemple OpenMP "grain grossier"

On parle d'OpenMP grain grossier, quand on crée les threads (par un pragma omp parallel), mais quand c'est le programmeur qui répartit le travail entre les threads :

```
#pragma omp parallel
{
  int ith = omp_get_thread_num();
  int nth = omp_get_num_threads();
  int nDebut = ...
  int nFin = ...
  int i;
  for (i=nDebut; i<nFin; i++) {
    ...
  }
}
```

Chaque thread calcule ses valeurs de `nDebut` et `nFin`. Faire attention que la réunion des intervalles `[nDebut, nFin[` doit être égale à l'ensemble de indices de la boucle du programme séquentiel.

Question 2.

Introduire dans le fichier `src/sinus_coarse_1/sinus.cxx` des directives (pragma) OpenMP "grain grossier".

Comparer les temps d'exécution de la version "grain fin" et de cette version en tapant les commandes

```
OMP_NUM_THREADS=3 time -p ./install/sinus_fine 40000
OMP_NUM_THREADS=3 time -p ./install/sinus_coarse_1 40000
```

On a déjà présenté la classe Timer pour mesurer le temps passé dans un groupe d'instructions :

```
#include "timer.hxx"
...
#pragma omp parallel
{
  int ith = omp_get_thread_num();
  int nth = omp_get_num_threads();
  Timer T;
  T.start();
    // groupe d'instructions à mesurer
  T.stop();
  std::cerr << "thread " << ith
      << " temps " << T.elapsed() << std::endl;
}
```

Question 3.

Mesurer dans le code OpenMP gros grain, le temps d'exécution de chaque thread en utilisant la classe Timer

Comparer les temps d'exécution de la version "grain fin" et de cette version en tapant les commandes

```
OMP_NUM_THREADS=3 time -p ./install/sinus_fine 40000
OMP_NUM_THREADS=3 time -p ./install/sinus_coarse_1 40000
```

4 Exemple OpenMP "grain grossier" avec équilibrage de charge

Le calcul du sinus en utilisant un développement de Taylor a été volontairement ralenti pour accentuer la différence de temps calcul de $x \mapsto \sin x$ pour différentes valeurs de x.
Il s'en suit que les threads ne prennent pas le même temps de calcul suivant la plage des valeurs de x qui leur sont attribuée (et qui est la même que dans le cas "grain fin"), voir le fichier `src/sin.cxx`.
Dans cette version, on utilise un algorithme d'équilibrage de charge entre les différents threads.

Demonstration 2.

Examiner le fichier `src/sinus_coarse_2/charge.cxx` qui contient cet algorithme et le fichier `src/sinus_coarse_2/sinus.cxx` qui l'utilise.

Exécuter plusieurs fois la commande

```
OMP_NUM_THREADS=3 time -p ./install/sinus_coarse_2 40000
```

Chaque exécution tente d'améliorer les temps calcul en adaptant la répartition de charge de mieux en mieux (si possible).

Remarque

(Optionel) L'algorithme d'équilibrage de charge utilisé ici n'est pas optimal. Vous êtes encouragés à l'étudier et à l'améliorer.

5 Parallélisation du (mini-)code avec le modèle OpenMP "grain grossier"

Le répertoire `code/PoissonOpenMP_FineGrain` contient la version du code parallélisé par de l'OpenMP "grain fin" (tel qu'obtenu à la fin du TP précédent).

Question 4.

Dans le répertoire `PoissonOpenMP_CoarseGrain`, modifier les fichiers source pour obtenir une version avec de l'OpenMP "gros grain" (sans équilibrage de charge).

Les fichiers source, que vous avez modifiés, seront à fournir avant le début de la séance suivante et vous seront renvoyés commentés et notés. La note obtenue pourra améliorer la note finale suivant les modalités précisées au premier cours.

6 Parallélisation en utilisant le concept de tâches OpenMP

Demonstration 3.

Examiner le fichier `src/exemple_tasks/main_seq.cxx`.

Cet exemple est difficile à paralléliser avec OpenMP. Expliquer pourquoi.

Le modèle de programmation par tâches OpenMP a été conçu pour répondre à ce type de situation.

Demonstration 4.

Examiner le fichier `src/exemple_tasks/main_tasks.cxx`.

Compiler et exécuter. Comparer les temps calcul avec la version séquentielle.

Pensez à vérifier que les résultats sont bien les mêmes dans les deux cas.