TP de programmation parallèle avec OpenMP

Université Lorraine - 2022 - 2023

Démarrage

La bibliothèque `OpenMP` est directement intégrée aux compilateurs `C/C++` les plus courants (dont `gcc`). On rappelle que `OpenMP` fonctionne via des *directives* données au *pré-processeur*. Pour que celles-ci soient prises en compte, il faut ajouter une option spécifique au compilateur. Le nom de cette option peut varier d'un compilateur à l'autre, il faut donc se référer au manuel du compilateur utilisé.

Dans notre cas, nous utilisons le compilateur `gcc` et l'option est `-fopenmp`. On a donc la commande de compilation suivante :

```
gcc -fopenmp source.c -o executable
```

Il est également conseillé d'ajouter l'option donnant les alertes de compilation (`-Wall`) ainsi que la bibliothèque mathématique (`-lm`).

Pour faciliter la compilation de vos programmes, il est recommandé d'utiliser un fichier `Makefile` qui permet d'automatiser la compilation de tous les programmes `C` du dossier courant en utilisant la commande `make` dans le dossier où se trouvent les sources.

Une archive est disponible sur le site Arche du cours, qui contient les fichiers sources des différents exercice de ce TP ainsi qu'un `Makefile` pour la compilation automatique.

Exercice 1 : Premières manipulations

Lorsque l'on compile avec `OpenMP`, cela crée une constante nommée `_OPENMP` qui permet de détecter `OpenMP` dans le programme. Cela peut notamment être utile pour inclure ou non l'API `OpenMP`. L'utilisation de cette constante est visible dans notre premier programme `exo_parallel.c`. Celui-ci effectue quelques affichages et crée simplement une région parallèle.

a) Compilez et exécutez le programme. Que constatez-vous ?

b) Recopiez la ligne de compilation en enlevant l'option `-fopenmp`. Que constatez-vous lors de la compilation ? et lors de l'exécution du programme ?

c) Effacez le programme exécutable `exo_parallel` et recompilez avec la commande `make`.
Exécutez le programme en définissant la variable d'environnement du nombre de threads à 12, avec la commande suivante : `OMP_NUM_THREADS=12 ./exo_parallel`.
Que constatez-vous ?

d) Ajoutez l'option `num_threads(3)` à la suite de la directive `parallel` dans le source. Recompilez et exécutez avec et sans la variable d'environnement. Que constatez-vous ?

Exercice 2 : Une première boucle parallèle

Le programme dans le fichier source `exo_boucle.c` effectue une boucle.

a) Compilez et exécutez ce programme. Que peut-on dire des itérations de la boucle ?

b) Ajoutez l'option `schedule(static, 2)` à la directive `for`. Recompilez et exécutez le programme. Quels changements constatez-vous ?

c) Commentez l'option `schedule(static, 2)` ainsi que le `printf` dans la boucle, et modifiez le nombre d'itérations à 1000000. Recompilez et exécutez plusieurs fois le programme. Que constatez-vous quant au cumul calculé ?

d) Afin de corriger le calcul du cumul, ajoutez la directive d'exclusion mutuelle `#pragma omp atomic` juste avant la ligne `cumul++;`.
Recompilez et exécutez plusieurs fois le programme. Que constatez-vous quant au résultat et au temps de calcul ?

e) Commentez la directive `atomic` et ajoutez l'option `reduction(+:cumul)` à la directive `for`. Recompilez et exécuter plusieurs fois le programme.
Que constatez-vous ?

Exercice 3 : Une boucle déséquilibrée

Le programme correspondant au fichier source `exo_equilibre.c` effectue une boucle déséquilibrée, c'est-à-dire dans laquelle les itérations n'ont pas les mêmes temps de calcul.

a) Compilez et exécutez ce programme. Notez bien les temps de calcul.

b) Ajoutez l'option `nowait` à la directive `for`. Recompilez et exécutez le programme. Que constatez-vous ? Expliquez les temps obtenus précédemment.

c) Ajoutez l'option `schedule(dynamic)` à la suite du `nowait`. Recompilez et exécutez le programme. Que constatez-vous ?

Exercice 4 : Parallélisme imbriqué

Dans le programme correspondant au fichier `exo_imbrication.c`, plusieurs régions parallèles imbriquées sont créées et des affichages sont effectués à différents niveaux.

a) Compilez et exécutez ce programme. Que peut-on dire des threads de niveaux 2 et 3 par rapport à ce qui est attendu pour chaque région parallèle ?

b) Ajoutez l'instruction `omp_set_nested(1);` avant la première région parallèle.
Recompilez et exécutez le programme. Que constatez-vous ?

Exercice 5 : Génération de tâches

Le fichier source `exo_taches.c` est une variante de la boucle de calcul déséquilibrée vue précédemment.

a) Compilez et exécutez ce programme. Expliquez les affichages obtenus.

b) Placez la boucle de calcul dans une région `single`, comme indiqué dans le code source suivant :

```
#pragma omp single
{
  int i;

  for(...){
    ...
  }
}
```

Recompilez et exécutez le programme. Que constatez-vous ?

c) Ajoutez une directive `task` juste avant l'appel à la fonction `travail`. Recompilez et exécutez le programme. Expliquez les résultats obtenus.

Exercice 6 : Application à un cas concret

L'objectif est d'utiliser les concepts vus précédemment pour paralléliser le programme séquentiel donné dans le fichier `exo_occurrences.c`.