TP 3 : Initiation à OpenMP

L'objectif de cette séance est d'apprendre à utiliser les commandes de base d'OpenMP, en particulier celles destinées à paralléliser des boucles de calcul et à gérer les variables. Les codes sont disponibles sur le site Internet du cours : https://perso.ensta-paris.fr/~modave/SIM203/

Exercice 1. Définition du nombre de threads

Cet exercice est le `helloworld` *d'OpenMP. Bienvenue dans le monde du calcul parallèle !*

Le nombre de threads peut être défini de plusieurs façons (avec la clause `num_threads`, avec la fonction `omp_set_num_threads` ou la variable d'environnement `OMP_NUM_THREADS`).

1. Lorsque le nombre de threads est défini de plusieurs façons dans un même code, mais avec des valeurs différentes, quelle valeur sera utilisée pour la section parallèle ? Testez en partant du code `helloworld1.c`.
2. Que devient le nombre de threads par défaut après la fin de la section parallèle ? Utilisez la fonction `omp_get_num_threads()` pour vérifier.

Exercice 2. Région parallèle et portée des variables

Dans cet exercice, on s'intéresse aux aspects "mémoire". Vous devez simplement jouer avec un code pour apprendre le fonctionnement des variables privées/partagées.

Au sein d'une région parallèle, une variable peut être partagée par les threads (variable *partagée*) ou avoir une valeur différente pour chaque thread (variable *privée*). Si la variable est déclarée avant une région parallèle, différentes clauses peut être utilisées pour définir le type de variable.

1. Exécutez et modifiez le code `variables.c` pour comprendre l'effet des clauses `private`, `firstprivate` et `shared` sur les valeurs des différentes variables avant/pendant/après la région parallèle.
2. Une variable peut-elle être listée dans plusieurs clauses ?
3. Quel est le rôle de `default(none)` ? Peut-on retirer une des variables des listes ?

Exercice 3. Parallélisation de boucles et opération réduction

Dans cet exercice, on s'intéresse aux aspects "calcul" en manipulant un code bien connu.

On teste la parallélisation des boucles et l'opération de réduction avec OpenMP sur le code de différences finies utilisé pendant le deuxième TP (`diffusionIcc.c`).

1. Reprennez la meilleure implémentation du programme `diffusionIcc.c` et ajoutez une directive de compilation pour paralléliser le calcul.
2. Vérifiez la parallélisation et la vectorisation en lisant le rapport de compilation.
3. Ajoutez une boucle pour calculer la valeur maximale du champs à tout instant. Parallélisez la boucle en utilisant la directive `parallel for` avec la clause `reduction(max,...)`.
4. Étudiez le temps de calcul en faisant varier le nombre de threads *(1, 2, 4, 6, 8)*.

Exercice 4. Code dgemm – Vectorisation et parallélisation

À votre tour. Accélérez votre code !

1. Reprenez votre code dgemm du premier TP *(version avec des boucles, sans appel à une librairie BLAS extérieure)*. Vérifiez la vectorisation en lisant le rapport de compilation.
2. Ajoutez une directive de compilation pour paralléliser le calcul. Étudiez le temps de calcul en faisant varier le nombre de threads *(1, 2, 4, 6, 8)*.

Références

– OpenMP 3.1 API C/C++ Syntax Quick Reference Card
https://www.openmp.org/wp-content/uploads/OpenMP3.1-CCard.pdf
– Intel® C++ Compiler Classic Developer Guide and Reference – OpenMP* Pragmas Summary
https://www.intel.com/content/www/us/en/develop/documentation/cpp-compiler-developer-guide-and-reference/top/optimization-and-programming-guide/openmp-support/openmp-pragmas-summary.html