TD OpenMP

OpenMP pas à pas

Télécharger l'archive sous moodle TD_OpenMP, la désarchiver dans votre espace.

1- Hello world

a. Compiler et exécuter le programme hello.cpp. Ouvrir le fichier hello.cpp : ceci est la version standard.
b. Ouvrir le fichier hello_par_1.cpp. Les directives OpenMP ont été ajoutées.
   i. Compiler avec *g++ hello_par_1.cpp -o hello_par_1 –fopenmp*
   ii. Commenter l'exécution
c. Ouvrir le fichier hello_par_2.cpp
   i. Compiler ce fichier
   ii. Comparer l'exécution et le code avec le programme précédent.
   iii. Quelles sont les similitudes de comportement avec fork() ?

2- Premiers algo

a. Compiler le fichier algo_simple_1.cpp puis l'exécuter.
   i. Comparer les résultats obtenus avec les autres groupes
   ii. Que se passe-t-il ?
   iii. Ajouter la clause « reduction(+:sum) » à la directive « #pragma omp parallel » (la ligne devient : #pragma omp parallel reduction(+:sum) »
   iv. Recompiler et exécuter. Changements observés ?
b. Analyser le code du fichier algo_simple_2.cpp
   i. Combien de fois est exécutée la boucle for ?
   ii. Combien de fois est exécutée l'instruction « cout << "Hello World from thread = "?
   iii. Peut-on prévoir le découpage de la boucle ***for*** ?
   iv. Suite à l'exécution, conclusion ?
c. En cumulant les 2 précédents exemples, l'algo_simple_3.cpp tire profit de la mise en concurrence.
   i. Cette approche garanti-t-elle le résultat ?
   ii. Combien faut-il de thread pour obtenir le résultat le plus rapidement ?
   iii. Quel impact a le changement du type double à float (changer le *typedef* de ***DataType*** de double à float, recompiler !)

3- Multiplication Matricielle (organiser vous entre vous !)

a. Algorithme naif de multiplication, mais avec une mise en concurrence OpenMP sur les 3 boucles en for imbriquées : matmul_par.cpp.
b. A compiler et à exécuter.
c. Donner les temps pour 1, 2, 3, 4, 6, 8, 16, 32, 64 et 128 threads.
d. La valeur optimale est-elle la même que précédemment ?
e. Idem avec le type float ?

4- Approximation de pi : 3.1415926535….

a. Le fichier pi_loop.cpp permet le calcul approché de pi.
b. Quelles sont vos conclusions sur le temps, le nombre de thread et la précision ?
c. Re compiler avec *g++ pi_loop.cpp -o pi_loop –fopenmp* ***–O3***
d. Quelles sont vos nouvelles conclusions?