TP2 Calcul Haute Performance : OpenMP

Exercice 1 : Produit matrice vecteur en OpenMP

Ecrire un programme parallèle qui calcule le produit d'une matrice et d'un vecteur :

$$Ax = b$$

en utilisant OpenMP. Comparer les performances en augmentant le nombre de threads pour un problème de taille fixe. Que pensez-vous des performances ?

Exercice 2 : Calcul de Pi

Proposez un programme qui parallélise une approximation de la valeur de Pi en utilisant OpenMP.

Exercice 3 : La conjecture de Goldbach

Le but de cet exercice est de démontrer l'intérêt de l'utilisation de la clause *SCHEDULE* de OpenMP pour améliorer les performances d'un code parallélisé.

Dans le monde de la théorie des nombres, d'après la conjecture de Goldbach: chaque nombre pair supérieur à deux est la somme de deux nombres premiers.

Dans cet exercice, vous allez écrire un code pour tester cette conjecture. Le code permet de trouver le nombre de paires de Goldbach pour un nombre pair donné i (c'est à dire le nombre de paires de nombres premiers P_1, P_2 tels que $P_1 + P_2 = i$) pour $i = 2 \dots 8000$. Le coût de calcul est proportionnel à $i^3/2$, donc pour obtenir une performance optimale avec plusieurs threads la charge de travail doit être distribuée en utilisant intelligemment la clause *SCHEDULE* de OpenMP.

Code séquentiel

On vous demande de:

1. Ecrire une fontion *isprime* qui teste la primalité d'un nombre donné.
2. Ecrire une fonction *goldbach* qui permet de trouver le nombre de paires de nombre premiers pour un nombre pair donné.
3. Dans la fonction *main*
   (a) Initialiser le tableau *numpairs* à zéro.
   (b) Modifier le contenu du tableau *numpairs* en appelant la fonction *goldbach*. Le tableau *numpairs* contiendra alors le nombre de paires de Goldbach pour chaque nombre pair jusqu'à 8000.
   (c) Une fois le tableau *numpairs* a été rempli avec le nombre de paires de Goldbach, tester que la conjecture de Goldbach est vraie, c'est à dire que pour chaque nombre pair supérieur à 2 il existe au moins une paire de Goldbach.

Le résultat obtenu doit être en adéquation avec l'exemple suivant:

Nombre pair	Nombre de paires de Goldbach
2	0
802	16
1602	53
2402	37
3202	40
4002	106
4802	64
5602	64
6402	156
7202	78

Parallélisation du code

On vous demande de:

1. Paralléliser le code séquentiel en utilisant OpenMP (ajouter les directives nécessaires autour de la boucle appelant la fonction *goldbach*) sans préciser d'ordonnancemment.
2. Vérifier que votre code trouve la bonne solution en utilisant plusieurs threads.
3. Afin d'accélérer la vitesse de calcul, utiliser un ordonnancement d'abord statique *schedule(static)* puis dynamique *schedule(dynamic)*, en précisant la taille du bloc. Récapitulez tous les résultats dans un tableau et analysez-les. Pour mieux comprendre l'effet des différentes méthodes d'ordonnancement vous pourriez afficher le thread traitant chaque itération.
4. Utiliser un ordonnancement guidé *schedule(guided)*, analyser l'effet de cet ordonnancement sur les performances de votre programme.

A votre avis, l'implémentation séquentielle proposée pour cet excercice est-elle pertinente ?