TP Programmation Parallèle : OpenMP Tasks

Exercice 1 : Compte d'occurrences dans un tableau en parallèle

Allouer un tableau A de taille N et initialiser le tableau avec des valeurs aléatoires entre 0 et V en parallèle (à l'aide de la fonction rand_r(unsigned int *seedp)). Vous pouvez commencer par des valeurs $N = 1M$ et $V = 100$, puis modifier ces valeurs pour tester la performance de votre programme.

Dans un premier temps, on ecrit une fonction $recherche\_occurrences(A, cle)$ qui rend le nombre d'occurrences de la valeur cle dans A. Faire le code en séquentiel, puis le paralleliser avec #pragma omp for adequatement. La fonction $recherche\_occurrences$ contiendra un #pragma omp for orpheline. On créera une région parallèle plutôt la où on appelle cette fonction dans main.

En suite, réaliser une version parallèle à l'aide de OpenMP Tasks. Il faudrait définir une variable $task\_size$ qui précise la taille de chaque tâche, puis créer des tâches de manière que chaque tâche parcourt $task\_size$ elements contigus du tableau A. Tester la performance avec de differentes valeurs de $task\_size$. Quelle est la valeur optimale? Finalement, on compare la pérformance pour les valeurs $V = 10$ et $V = 1000$. Que constate-t-on?

Exercice 2 : Merge sort en parallèle

Dans cet exercise, on va réaliser le tri d'un tableau en parallèle à l'aide de l'algorithme "merge sort". Pour un tableau $A[N]$ donné, il s'agit de trier les sous-tableaux $A[1 \dots N/2]$ et $A[N/2{+}1 \dots N]$ récursivement, en suite appliquer une fonction $merge(A, 1, N/2, N, B)$ qui fusionne ces deux sous-tableaux triés dans un tableau auxiliaire $B[N]$. Copier B dans A finalise l'algorithme.

Implementer la fonction $merge(A, i, j, k, B)$ qui prend un tableau A dont sous-tableaux $A[i \dots j]$ et $A[j + 1 \dots k]$ sont triés, et fusionne les deux dans le sous-tableau $B[i \dots k]$. Une fois que cette fonction est réalisé, implementer la fonction $mergesort(A, i, j)$ qui tri le sous-tableaux $A[i \dots j]$ avec de façon récursive à l'aide de l'algoritme détaillé précédemment.

Une fois que l'implementation fonctionne, paralleliser-le avec des taches tenant en compte que les sous-tableaux peuvent être trié independemment, mais $merge$ doit êtré réalisé en séquentiel. Pour l'éfficacité, mettre un seuil min_task_size tel que les tableaux dont taille est inferieure à cela sont triés entièrement en séquentiel.

Exercice 3 : Produit Matrice Vecteur avec des tâches

Ecrire un programme parallèle qui calcule le produit d'une matrice de taille $N \times N$ et d'un vecteur de taille N:

$$Ax = b$$

en utilisant OpenMP.

Allouer la matrice et l'initialiser de façon orientée par les lignes (c'est à dire, les premières N éléments correspondent à la première ligne $A(0, :)$, puis la deuxième ligne $A(1, :)$, etc) tel que $A(i, j) = i + j$. Allouer les vecteurs x et b de taille N, et initialiser le vector x à 1 et b à 0. En suite, effectuer la multiplication $b = Ax$ ligne-par-ligne.

Une fois que vous avez le code séquentiel, parallèlisez-le (chaque ligne peut être initialié indepen-demment) et tester l'accéleration avec de differents nombres de threads pour un problème de taille

fixe. Cette fois-ci on va effectuer la parallélisation à l'aide de OpenMP Tasks. Créer des taches pour effectuer la multiplication de $task\_size$ lignes.

Bonus: Utiliser une topologie des taches tel que chaque tâche effectue la multiplication d'un bloc du A de taille $task\_size \times task\_size$.