ANALYSE TP

EXERCICE 1

Pour minimiser les défauts de cache lors de la parallélisation du programme , il est important de s’assurer que chaque thread accède à des parties distinctes des tableaux A,B et C. Dans ce code la directive « #pragma omp parallel for » indique au compilateur de paralléliser la boucle for sur plusieurs threads. L a clause « shared(A,B,C) » indique que les tableaux doivent etre partagés entre tous les threads et la clause « schedule(static) » spécifie la répartition statique des itérations de la boucle for sur les threads.En utilisant cette approche ,chaque thread accède à une partie différente des tableaux A,B et C, minimisant ainsi les défauts de cache et améliorant les performances.

EXERCICE 2

VERSION 1 : Réduction explicite

Dans cette version, la réduction est effectuée de manière explicite en utilisant la clause « reduction » de OpenMP. Dans ce code, la directive « pragma omp parallel for » indique au compilateur de paralléliser la boucle for sur plusieurs threads ; La clause « reduction(+ :sum) » indique que la somme doit etre réduite à travers tous les threads en utilisant l’opération « + ».

VERSION 2 : Réduction implicite

Dans cette version, la réduction est effectuée de manière explicite en utilisant la fonction  « omp\_get\_thread\_num»  pour accéder à une variable locale pour chaque thread .Dans ce code, la directive « pragma omp parallel  » indique au compilateur de créer plusieurs threads  ;la directive « #pragma omp » for indique que la boucle for doit etre parallélisée sur plusieurs threads, et chaque thread accumule la somme locale dans la variable « local\_sum ». Ensuite, la clause   « #pragma omp critical » est utilisée pour s’assurer que l’accumulation dans la variable globale « sum » est effectuée de manière atomique pour éviter les conflits.

Il est important de noter que la performance de ces deux approches dépendra de la taille du tableau et du nombre de threads disponibles sur le système.

EXERCICE 3

Chaque thread exécute une partie de la boucle for extérieure en itérant sur un sous-ensemble de lignes de la matrice A et accumule le résultat dans la variable C. Les threads accèdent à des parties différentes de la matrice A et du vecteur B, minimisant ainsi les conflits de mémoire. Il est important de noter que la performance de cette approche dépendra de la taille de la matrice et du nombre de threads disponibles sur le sytème.

EXERCICE 4

Les variables « chunk\_size ; start ; end » sont utilisées pour diviser la boucle for entre les threads. Le calcul de « chunk\_size », « start » et « end » est effectué en utilisant la fonction OpenMP « omp\_get\_thread\_num ()» pour obtenir l’identifiant du thread courant et « omp\_get\_num\_threads() » pour obtenir le nombre total de threads dans la section parallèle. Il est important de noter que la performance de cette approche dépendra de la taille du tableau A et du nombre de threads disponibles sur le système.

EXERCICE 5

Comme dans l’exercice 5 Les variables « chunk\_size ; start ; end » sont utilisées mais cette fois pour diviser la boucle for extérieure entre les threads. Le calcul de «  chunk\_size », « start » et « end » est effectué en utilisant la fonction OpenMP « omp\_get\_thread\_num ()» pour obtenir l’identifiant du thread courant et « omp\_get\_num\_threads() » pour obtenir le nombre total de threads dans la section parallèle . Il est important de noter que la performance de cette approche dépendra de la taille du tableau A et du nombre de threads disponibles sur le système.