Deuxième série d'exercices sur OpenMP
Boucle orpheline, Combinaison derectives parallel et for, et information sur l'environnement OpenMP

Compléments de cours

1. **Examiner/compiler/exécuter le code des exemples suivants :**
   - **omp_master.c**
   - **omp_nowait.c**
   - **omp_single.c**
   - **omp_schedule.c**

2. Examiner/compiler/exécuter le code de l'exemple orphan(réduction de boucle parallèle orpheline)

L'exemple suivant calcule un produit scalaire, cependant il diffère des exemples précédant car la construction de la boucle parallèle est orpheline, elle est contenue dans un sous programme en dehors de la portée de la région parallèle du programme principal

a. Après avoir examiné le code, compiler puis exécuter

```
gcc -fopenmp omp_orphan.c -o orphan
./orphan | sort
```

b. Noter le résultat, cet exemple on le verra plus tard dans l'analyse des bugs (omp_bug6)

3. Examiner/compiler/exécuter le code de l'exemple matmult (multiplication de matrice)

Cet exemple effectue une multiplication de matrices en distribuant les itérations de l'opération de multiplication entre les threads disponibles.

a. Après avoir examiné le code, compiler puis exécuter

```
gcc -fopenmp omp_omp_matmult.c -o matmult
./matmult
```

b. Examiner la sortie, ce programme montre quel thread qui travaille chaque itération et le résultat final
c. Exécuter le programme à nouveau mais cette fois ci, trier la sortie pour voir clairement quel thread qui exécute quel itération.

```
./matmult  | sort | grep Thread
```

d. Est-ce que la boucle d'itération suit la directive d'ordonnancement (STATIC, CHUNK) dans code de multiplication de matrice ?

4. Examiner/compiler/exécuter le code de l'exemple Combined Parallel Loop Reduction omp_reduction.c

a. Après avoir examiné le code, compiler puis exécuter

```
gcc -fopenmp omp_reduction.c -o reduction
./reduction
```

b. Que peut-on dire de la variable resul

5. Examiner/compiler/exécuter le code de l'exemple de combinaison des directive parallel et for avec reduction

c. Après avoir examiné le code, compiler puis exécuter

```
gcc -fopenmp omp_omp_parallel_for.c -o parallel_for
./parallel_for
```

d. Que peut-on dire de la variable resul

6. Examiner/compiler/exécuter le code de l'exemple avec la directive threadprivate

a. Après avoir examiné le code, compiler puis exécuter

```
gcc -fopenmp omp_threadprivate.c -o threadprivate
./threadprivate
```

a. Examiner la sortie, ce programme montre quel thread qui travaille chaque itération et le résultat final
b. Examiner la valeur des variables privées de chaque thread
c. Peut on les comparer à des variables privées

```
1st Parallel Region:
Thread 0:   a,b,x= 0 0 1.000000
Thread 2:   a,b,x= 2 2 3.200000
Thread 3:   a,b,x= 3 3 4.300000
Thread 1:   a,b,x= 1 1 2.100000
**********************************
Master thread doing serial work here
**********************************
2nd Parallel Region:
Thread 0:   a,b,x= 0 0 1.000000
Thread 3:   a,b,x= 3 0 4.300000
Thread 1:   a,b,x= 1 0 2.100000
Thread 2:   a,b,x= 2 0 3.200000
```

7. Obtenir les informations d'environnement

a. Ecrire un programme simple affiche des informations sur votre environnement OpenMP.
b. Utiliser les procédures et fonctions appropriées dans le thread principal pour afficher ce qui suit :
    - Le nombre de processors disponibles
    - Le nombre de threads utilisés
    - Le nombre maximum de threads disponibles
    - Si vous êtes dans une région parallèle
    - Si le thread dynamique est activé
    - Si le parallélisme imbriqué est pris en charge

Remarque : SI vous avez besoin d'aide, vous pouvez consulter le fichier de variables d'environnement omp_getEnvInfo