Tâches en OpenMP

Copiez le répertoire `/net/cremi/rnamyst/etudiants/pmg/TP3` sur votre compte.

1 Deux tâches

Modifiez le programme `deux-taches.c` pour qu'il exécute deux tâches en parallèle : chaque tâche écrira un mot et le numéro du thread qui l'exécute.

2 For en tâches

Modifiez le programme `for-en-taches.c` afin que les indices soient distribués au moyen de tâches tout en conservant un comportement globalement similaire.

3 Taskwait

Analyser le comportement du programme `task-wait.c`. Pour simplifier, on pourra lancer ce programme avec 4 threads.

4 Tâches et durée de vie des variables locales

Expliquez le comportement du programme `task.c` pour les quatre cas de figure :

1. taskwait et nowait décommentés ;
2. taskwait décommenté et nowait commenté ;
3. taskwait commenté et nowait décommenté ;
4. taskwait et nowait commentés.

5 Suite du TP2 (TSP) : parallélisation à l'aide de tâches OpenMP

Dupliquer le répertoire source initial pour paralléliser l'application à l'aide de tâches. Au niveau du main() il s'agit de créer une équipe de threads et de faire en sorte qu'un seul thread démarre l'analyse. Au niveau de la fonction tsp lancer l'analyse en faisant en sorte de ne créer des tâches parallèles que jusqu'au niveau `grain`. Deux techniques d'allocation mémoire sont à comparer :

1. allocation dynamique : un tableau est alloué dynamiquement et initialisé avant la création de la tâche - ce tableau sera libéré à la fin de la tâche ;
2. allocation automatique : le tableau est une variable locale allouée et initialisée dans la tâche - il est alors nécessaire d'utiliser la directive taskwait après avoir créé toutes les tâches filles.

Comparer les performance obtenues par les deux approches sur le cas 15 villes et seed 1234 pour des grains variant de 1 à 9. Comparer à celles obtenues à l'aide des techniques *imbriquées* et *collapse*.

Relever ensuite le(s) meilleur(s) grain(s) pour 2, 4, 6, 12 et 24 threads. Calculer les accélérations obtenues.