App4, Programmation Parallèle : TP 1 : OpenMP Tasks

Récupérer le code à compléter sur https://www.lri.fr/~bagneres/index.php?section=teaching&page=2016_App4_prog_parallele
Le projet utilise *CMake* https://fr.wikipedia.org/wiki/CMake pour lancer la compilation des différents tests. Le fichier `README.txt` donne les différentes commandes pour compiler les différents tests (pour les systèmes comme GNU/Linux, sinon : https://www.lri.fr/~bagneres/index.php?section=tools&page=cmake_and_cpp).
Pour lancer un test en particulier : `make && ./test__nom_du_test`

Faire les exécutions à comparer sur la même machine et dans les mêmes conditions. Regarder les caractéristiques du système d'exploitation, du compilateur (version et options) et du processeur (modèle, nombre de cœurs, hyperthread, taille des caches, ... (la commande `cat /proc/cpuinfo` est un bon début)).
Afin d'avoir une idée correcte du temps d'exécution lancer plusieurs fois le programme et prendre la médianne ou le minimum des temps.

1 OpenMP task

Le code suivant créé 5 tâches OpenMP :

```
int t1, t2, t3, t4, t5;
#pragma omp parallel
{
    #pragma omp single nowait
    {
        #pragma omp task
        {
            std::cout << "T1" << std::endl;
            t1 = 1;
        }
        #pragma omp task
        {
            std::cout << "T2" << std::endl;
            t2 = 2;
        }
        #pragma omp task
        {
            std::cout << "T3" << std::endl;
            t3 = 3;
        }
        #pragma omp task
        {
            std::cout << "T4" << std::endl;
            t4 = 4;
            std::cout << "t1 = " << t1 << std::endl;
            std::cout << "t2 = " << t2 << std::endl;
            std::cout << "t3 = " << t3 << std::endl;
            std::cout << "t4 = " << t4 << std::endl;
        }
        #pragma omp task
        {
            std::cout << "T5" << std::endl;
            t5 = 5;
            std::cout << "t5 = " << t5 << std::endl;
        }
    }
}
```

1.1 Que se passe-t-il si on exécute ce programme ?

Note : Exécuter plusieurs fois le programme.

1.2 Ajouter toutes les dépendances. La tâche 4 doit s'exécuter après les tâches 1, 2 et 3 (modifier le fichier `tests/tasks.cpp`).

Note : La tâche suivante `#pragma omp task depend (out:y) depend (in:x)` ne peut s'exécuter que si la tâche qui produit `x` est terminée. Elle produit `y` en sortie.

2 Smith-Waterman

Le code suivant implémente l'algorithme de Smith-Waterman. Cet algorithme permet d'aligner des séquences (les bases nucléiques (A, C, G, T) d'une molécule d'ADN par exemple) :

```
std::vector<char> const a(N); // Contient des bases nucléiques (A, C, G, T)
std::vector<char> const b(M); // Contient des bases nucléiques (A, C, G, T)

hopp::llint const w_match = 2; // hopp::llint est un long long int
hopp::llint const w_mismatch = -1;
hopp::llint const w_gap_a = -1;
hopp::llint const w_gap_b = -1;

hopp::vector2D<hopp::llint> h(a.size() + 1, b.size() + 1, 0);

for (size_t i = 1; i < a.size() + 1; ++i)
{
    for (size_t j = 1; j < b.size() + 1; ++j)
    {
        h[i][j] = std::max(h[i][j], h[i - 1][j - 1] + ((a[i - 1] == b[j - 1]) ?
                  w_match : w_mismatch)); // match / mismatch
        h[i][j] = std::max(h[i][j], h[i - 1][j    ] + w_gap_a); // deletion
        h[i][j] = std::max(h[i][j], h[i    ][j - 1] + w_gap_b); // insertion
    }
}

// Lecture de h pour en déduire l'alignement de score optimal
```

2.1 Pourquoi ce code ne peut pas être directement parallélisé avec OpenMP ?

Note : Regarder les dépendances entre les différentes itérations.

2.2 Ce code peut être parallélisé après un tuilage (*tiling* en anglais). Expliquer pourquoi avec un schéma.

Note : Le tuilage est une transformation classique de boucle afin d'améliorer la localité des données.

2.3 Tuiler le code (compléter la fonction `smith_waterman_tile` du fichier `tests/smith-waterman.hpp`).

Notes : Traiter uniquement les tailles de tuiles qui sont des multiples exacts des tailles de `a` et `b`.
Lancer le programme avec des tailles facilement lisibles : `make && ./test__smith-waterman 4 4 4`

2.4 Pourquoi le code tuilé ne s'exécute pas plus vite que le code séquentiel ?

Note : Regarder la localité des données dans les deux versions.

2.5 Créer une tâche OpenMP pour chaque tuile sans oublier les dépendances (compléter la fonction `smith_waterman_task` du fichier `tests/smith-waterman.hpp`).

Note : Utiliser le "faux" tableau `depends` pour ajouter les dépendances.

2.6 Expliquer le gain obtenu par rapport au gain pouvant être naïvement espéré.