UNIVERSITÉ D'ORLÉANS **M1 Info**

Informatique Année **2017 - 2018**

Contrôle continu : Programmation Parallèle - 1h30

Exercice 1. Questions (5pts)

Vous devez répondre brièvement aux questions suivantes

1. (1pt) Si vous disposez d'une machine de type PRAM (mémoire partagée et nombre illimité de processeurs) quelle est la complexité du tri par induction (tri par comptage) que vous avez vu en TD ?
2. (2pts) Expliquez à quoi correspond le parallélisme de données ? Est-il possible d'utiliser ce parallélisme pour les architectures à mémoire distribuée et/ou à mémoire partagée ?
3. (2pts) Expliquez ce qu'est une condition pour les Pthreads.

Exercice 2. Multiplication matrice/vecteur (5pts)

Le code ci-dessous implémente une multiplication Matrice - Vecteur où A est un vecteur de taille m, B une matrice de taille $m \times n$ et C un vecteur de taille n avec $A = B * C$.

```
void mxv_row(int m, int n, double *A, double *B, double *C)
{
  int i, j;
  for (i=0; i<m; i++) {
    A[i] = 0.0;
    for (j=0; j<n; j++)
      A[i] += B[i*n+j]*C[j];
  }
}
```

1. (3pts) Modifiez ce code séquentiel pour le paralléliser avec les Pthreads.
2. (2pts) Même question avec OpenMP.

Exercice 3. Graphe par matrice d'adjacence(10pts)

Pour cet exercice, il ne s'agit pas d'écrire du code mais d'expliquer la parallélisation pour une architecture à mémoire distribuée et de donner les fonctions MPI qui seraient utiles à la mise en œuvre. Les explications peuvent d'être données sous forme de schéma sur un exemple en explicitant les paramètres choisis.

On représente un graphe orienté par une matrice d'adjacence A de taille $n \times n$ où A_{ij} est la valeur de l'arête du sommet i vers le sommet j ($A_{ij} = 0$ indique que les sommets ne sont pas reliés). On suppose cette matrice est déjà distribuée sur p processeurs.

1. Quelle est la complexité pour trouver l'arête de poids minimal ? Décrivez le principe de la parallélisation sachant que le résultat doit être disponible sur un processeur `root`. Donnez les fonctions MPI qui seront utiles pour la mise en œuvre.
2. Si on suppose que le graphe représente désormais un graphe de dépendances entre tâches (le sommet i représente la tâche T_i à exécuter). Le tri topologique suivant permet d'ordonnancer les tâches en indiquant celles qui peuvent s'exécuter en paralléle.

```
1: procedure TRI TOPOLOGIQUE(A la matrice d'adjacence, n la taille)
2:     for all i ≤ n do
3:         C_i = nombre d'arcs entrants dans i
4:     end for
5:     S_1 = l'ensemble des sommets sans prédécesseurs
6:     while S_1 ≠ ∅ do
7:         S_2 = ∅
8:         Afficher les sommets de S_1 (les tâches correspondantes peuvent s'exécuter en paralléle)
9:         for all i ∈ S_1 do
10:            for all j telque A_ij ≠ 0 do
11:                C_j = C_j − 1
12:                if C_j==0 then
13:                    S_2 = S_2 ∪ j
14:                end if
15:            end for
16:        end for
17:        S_1 = S_2
18:    end while
19: end procedure
```

Décrivez une parallélisation possible de cet algorithme. Le graphe est toujours représenté par sa matrice d'adjacence qui est distribuée par lignes sur les p processeurs. Donnez les fonctions MPI qui seraient nécessaires à la mise en œuvre de votre parallélisation sachant que pour la partie affichage seul un processeur `root` l'effectue. On rappelle que la matrice A est déjà distribuée sur les p processeurs.