**UNIVERSITÉ D'ORLÉANS** **M1 Info**

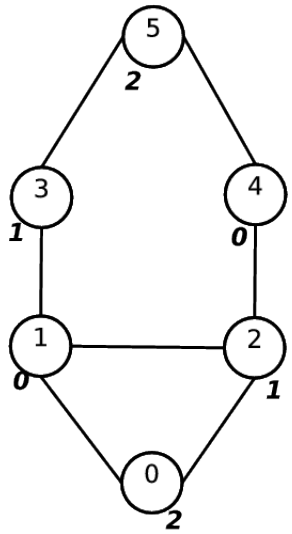
**Informatique** Année **2016 - 2017**

*Contrôle continu : Programmation Parallèle - 1h30*

---

**Exercice 1. Coloriage de Graphe, Algorithme de Brélaz (10pts)**

L'objectif de cet exercice est de proposer une parallélisation pour les architectures à mémoire distribuée de l'algorithme de Brélaz qui permet de colorier un graphe. Pour cela, un graphe sera représenté par une matrice d'adjacence qui sera distribuée à chaque processeur impliqué dans le calcul.




| | | | | | |
|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | 0 |

Comme illustré par l'exemple précédent la matrice d'adjacence $M$ est définie par $M_{ij} = 1$ s'il existe une arête entre le nœud $i$ et le nœud $j$ et $M_{ij} = 0$ sinon. On ne considérera que les graphes non dirigés et connexes. Pour le problème de la coloration le coût de l'arête n'a pas d'importance donc on le considérera toujours égal à $1$.

Le problème de la coloration consiste à affecter une couleur à chaque nœud du graphe de telle manière que deux nœuds adjacents soient coloriés par des couleurs différentes. Si on suppose qu'on numérote les couleurs de 0 à $n_c - 1$ où $n_c$ est le nombre de couleurs nécessaires, les numéros en bold-italic sur le graphe précédent illustre une coloration du graphe.

L'algorithme de Brélaz est un algorithme itératif. Dans une phase d'initialisation, on choisit le nœud de plus haut degré où le degré est le nombre d'arêtes connectées au nœud. On affecte la couleur 0 à ce premier nœud (en cas d'égalité on choisit au hasard parmi les nœuds de plus haut degré). Ensuite les étapes sont

1. calculer pour chaque nœud non colorié le nombre $n_{vc}$ de voisins coloriés,
2. choisir le nœud $n_{ac}$ ayant le plus grand $n_{vc}$ et en cas d'égalité choisir le nœud ayant le plus fort degré,
3. affecter au nœud $n_{ac}$ la plus petite couleur disponible et non utilisée par ses voisins.

Pour paralléliser cet algorithme on va distribuer les lignes de la matrice d'adjacence afin de répartir le calcul de $n_{vc}$ à chaque itération de l'algorithme. Il est à noter qu'en distribuant par ligne la matrice d'adjacence, chaque processeur gère une partie des nœuds du graphe mais que pour chaque nœud il connaît les arêtes auxquelles ce nœud est connecté.

- (5pts) Expliquez le principe de la parallélisation en supposant que la matrice d'adjacence de taille $n \times n$ est déjà distribuée sur chaque processeur. On appellera `graphe_local` la partie de la matrice d'adjacence affectée à chaque processeur et on supposera que $n$ est divisible par le nombre de processeurs (`graphe_local` est de taille $\frac{n}{nprocs} \times n$). Le résultat de votre parallélisation est un tableau `noeuds_colores` de taille $n$ tel que `noeuds_colores`$[i]$ est la couleur affectée au nœud $i$.
- (2pts) Explicitez les fonctions de communications MPI que vous allez utiliser et pourquoi.
- (3pts) Écrivez la fonction suivante qui implémente l'algorithme de Brélaz en parallèle

```
void Brelaz(int n_local, int n, int* graphe_local, int* noeuds_colores)
```

Vous pouvez utiliser des petites fonctions supplémentaires que vous n'avez pas besoin de coder si vous expliquez correctement ce qu'elles font. Par contre les communications doivent être explicites.

Quelques indications qui peuvent vous aider :

- Le tableau `noeuds_colores` peut ne pas être distribué mais mis à jour sur tous les processeurs à chaque itération
- Pour manipuler les degrés ou le nombre de voisins associés à chaque nœud vous pouvez utiliser une structure

```
typedef struct {
  int degre_couleur;
  int noeud;
} struct2int;
```

- Le type `MPI_2INT` existe et l'opération `MPI_MAXLOC` permet de manipuler à la fois le calcul du max et la position du max.

### Exercice 2. Calcul d'histogramme (6 points)

On veut écrire un programme parallèle pour produire un histogramme d'intensité à partir d'une image en niveau de gris représentée par une matrice de taille $n \times m$ contenant des valeurs comprises entre 0 et 255. L'histogramme sera représenté par un tableau de taille 256 où le nombre d'occurrences du nombre $k$ dans l'image (du niveau de gris) sera donné par l'élément à la position $k$ du tableau. Le problème est que chaque *thread* peut modifier les mêmes zones mémoires.

1. En utilisant OpenMP, proposez une première version de l'algorithme ou chaque *thread* calcule un histogramme localement, dans une première étape, avant de les fusionner en un unique histogramme global, dans une seconde étape.
   - Vous devez paralléliser les deux étapes.
   - Pour la première étape, vous ne devez pas utiliser de section critiques ou d'opérations atomiques. (3 points)
2. Proposer une seconde version de l'algorithme en une seule étape, où tous les *threads* écrivent directement dans un unique tableau histogramme partagé. (3 points)

### Exercice 3. Algorithme de Brelaz en mémoire partagée (4 points)

Dans cet exercice, vous allez reprendre l'algorithme de Brelaz, en mémoire partagée, avec OpenMP. On considérera la matrice d'adjacence globale et donc partagée par tous les *threads* OpenMP.

Décrivez (avec ou sans code) comment vous proposez de paralléliser avec OpenMP l'algorithme présenté Exercice 1. Avez-vous besoin de structures de données supplémentaires (globales/locales?)? Quelles directives OpenMP allez vous utiliser? Pourquoi?