**UNIVERSITÉ D'ORLÉANS** **M1 Info**
**Informatique** **2013 - 2014**

*Session 1: Programmation Parallèle - 2h*

---

**Exercice 1. PRAM : Calcul matriciel (4pts)**

Soit $A$ une matrice de taille $n \times n$, donnez un algorithme PRAM qui calcule $B$ la transposée de $A$ ($B = A^T$) et le nombre $M$ d'éléments non nuls de la matrice. $n$ est supposé connu par tous les processeurs. Indiquez le nombre de processeurs utilisés ainsi que la notation pour les numéroter. Précisez également le type de la machine PRAM utilisée.

**Exercice 2. MPI : Fonctions de communication (8pts)**

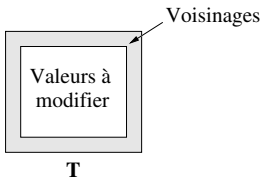
Soit un tableau à deux dimensions $T = (t_{ij})_{0 \leq i,j \leq n-1}$, pour les simulations basées sur les automates cellulaires, l'élément $(i, j)$ de $T$ sera mis à jour en fonction de $t_{ij}$ et des voisins de $t_{ij}$. Lorsque ce tableau est distribué sur $p$ processeurs, il est nécessaire de définir des fonctions d'échanges qui permettent à chaque processeur de recevoir les valeurs dont il a besoin pour effectuer toutes les mises à jour des valeurs de $T$ en local.

1. Supposons que $T$ est distribué par ligne sur $p$ processeurs tel que $p$ divise $n$. Supposons que chaque valeur est mise à jour en fonction de ses $4 \times x$ voisins ($x$ à gauche, $x$ à droite, $x$ en haut et $x$ en bas) et de manière circulaire. Ecrivez la fonction `EchangeGhostCol` suivante sans utiliser de topologie

```
void EchangeGhostLigne(float* T, int n, int x) {
  int pid; // numéro du processeur
  int p; // nombre de processeurs de MPI_COMM_WORLD
  MPI_Comm_rank(MPI_COMM_WORLD, &pid);
  MPI_Comm_size(MPI_COMM_WORLD, &p);

  // A compléter
}
```

   le tableau $T$ est supposé de taille suffisante pour recevoir les voisinages (les ghosts) :




2. Même chose mais cette fois-ci le tableau $T$ est distribué par bloc sur $p = p_1 \times p_2$ processeurs avec chaque bloc de taille $\frac{n}{p_1} \times \frac{n}{p_2}$ où $p_1$ et $p_2$ divisent $n$.

```
void EchangeGhostBloc(float* T, int n, int x, int p1, int p2) {
  int pid; // numéro du processeur
  int p; // nombre de processeurs de MPI_COMM_WORLD
  MPI_Comm_rank(MPI_COMM_WORLD, &pid);
  MPI_Comm_size (MPI_COMM_WORLD, &p);
  int n1 = n/p1;
  int n2 = n/p2;
  // A compléter
}
```

   On supposera également que $x$ est toujours plus petit que $\frac{n}{p_1}$ et $\frac{n}{p_2}$.

   Pensez à commenter vos codes en précisant par exemple la sémantique des fonctions MPI que vous utilisez.

**Exercice 3. MPI2 : Ferme de PC (8pts)**

L'objectif de cet exercice est d'implémenter en MPI2 une ferme de PC sur le principe suivant:

- Le maître possède une liste de tâches à effectuer (numérotées de 0 à N-1) et un nombre de processeurs $P$ à sa disposition
- Au démarrage le maître lance les $P$ instances des esclaves.
- Ensuite, il attribue à chaque esclave le même nombre de tâches ($\pm 1$).
- Les esclaves effectuent leurs tâches dans l'ordre des numéros de tâches. A chaque fois qu'un esclave a terminé une tâche, il doit l'indiquer au maître.
- Si un esclave a terminé toutes ses tâches, il doit l'indiquer au maître
- Régulièrement le maître vérifie que tous les esclaves sont bien occupés, si ce n'est pas le cas, il transfert une des tâches non encore effectuée (ni commencée) à chaque esclave libre.

Dans cet exercice on va considérer que chaque tâche est juste l'entier qui l'identifie et exécuter la tâche $i$ consiste à appeler la fonction $f(i)$ où $f$ est définie par ailleurs. Le maître devra connaître pour chaque tâche, le processeur à qui il l'a attribuée, son état (terminée ou non). Il devra aussi connaître pour chaque processeur s'il est occupé ou non. Chaque esclave devra connaître la liste des tâches qui lui sont attribuées (en sachant distinguer celles déjà effectuées et celles à faire). Toutes les communications entre le maître et les esclaves (quelque soit le sens) devront s'effectuer suivant le principe du *one-sided communication*. Pour simplifier, on va considérer que chaque esclave ne pourra jamais avoir plus de `MAXTASKS` tâches d'attribuées.

Ecrivez le code du maître et des esclaves.