**UNIVERSITÉ D'ORLÉANS** **M1 Info**

**Informatique** Année **2019 - 2020**

*Contrôle continu : Programmation Parallèle - 1h30*

---

**Exercice 1. Questions de cours (5pts)**

1. Questions liées à la bibliothèque **MPI**

   (a) Soit le tableau global `tab=[6 5 8 1 4 2 9 3 7]` réparti sur 3 processeurs (`tab_local` sur chaque processeur), quel est le résultat $res$ sur le processeur $pid = root$ de l'opération

   ```
   MPI_Reduce(tab_local,res,3,MPI_INT,MPI_SUM,root,MPI_COMM_WORLD)
   ```

   (b) Soit un tableau de 21 entiers initialisé sur le processeur `root` sachant qu'on dispose de 4 processeurs. Si on exécute une diffusion par

   ```
   MPI_Scatterv(tab,sendcounts, displs,MPI_INT,tab_recu,n,MPI_INT,root, MPI_COMM_WORLD)
   ```

   - Quels sont les processeurs qui exécutent cette commande ?
   - Quelle est la valeur de n sur le dernier processeur ?
   - Quel est le contenu de `sendcounts` et quel(s) processeur(s) le définit ?

2. Questions liées à la programmation par directives **OpenMP**

   (a) Donnez une définition d'une section critique

   (b) Soit le programme suivant, quelle est la directive à rajouter et avant quelle ligne pour le paralléliser ?

```
1 #include <omp.h>
2 #define N 100000000
3 void main (){
4         int A = 0;
5         int* tab = new int[N];
6         // ... initialisation de tab (hors sujet)
7         for (int i=0; i<N; i++)
8                 A += tab[i];
9 }
```

**Exercice 2. Vérification d'une solution Sudoku en mémoire distribuée (8 pts)**

Une grille de Sudoku généralisée est une matrice qui peut être vue comme un ensemble de blocs de taille $k \times k$ contenant tous les nombres de 1 à $k$ si c'est bien une solution Sudoku. La matrice globale $S$ est donc de taille $k^2 \times k^2$ et on a également une décomposition possible de $S$ comme $k \times k$ blocs de taille $k \times k$. On souhaite paralléliser la vérification qu'une grille est bien une grille Sudoku correcte soit que chaque bloc, ligne, colonne contiennent exactement une fois chaque valeur de $1$ à $k$.

1. Pour la vérification des blocs, quelle type de distribution choisiriez vous pour cette matrice $S$ potentiellement très grande sachant qu'initialement seul le processeur `root` a accès à cette matrice (génération ou lecture dans un fichier) ? Quelles hypothèses faut-il faire sur le nombre de processeurs afin d'obtenir une répartition des données équilibrées (même quantité sur chaque processeur) ? Et quelle organisation des processeurs choisiriez vous ?

2. Comment peut-on paralléliser efficacement cette vérification ? Expliquez sans donner de code mais à partir d'un schéma ou d'un algorithme en pseudo langage.

3. Donnez les routines MPI nécessaires en les justifiant par rapport à l'organisation des processeurs, aux étapes de la parallélisation et/ou de communications. Vous pourrez vous appuyer sur un exemple ($k$ et $nprocs$ fixés) et préciser les paramètres de ces routines par rapport à cet exemple.

4. Que faudrait-il ajouter pour faire la vérification complète ? Donnez les étapes supplémentaires sans les détailler.

**Exercice 3. Diffusion en temps $O(log_2(nprocs))$ (7 pts)**

Soit la fonction `Diffusion(int *val)` qui diffuse le scalaire **val** défini sur le processeur `0`. Cette diffusion doit être réalisée en temps $O(log_2(nprocs))$ où $nprocs$ est le nombre de processeurs. Sachant que vous ne pouvez utiliser que les fonctions send et recv bloquantes expliquez comment effectuer cette diffusion et donnez le code correspondant.

```
1
2  void Diffusion(int *val) {
3          int pid, nprocs;
4          MPI_Comm_rank(MPI_COMM_WORLD, &pid ) ;
5          MPI_Comm_size (MPI_COMM_WORLD, &nprocs ) ;
6
7          // ....
8
9
10         }
11
12 #include <mpi.h>
13
14 int main ( int argc , char **argv )
15 {
16   int pid, nprocs;
17   MPI_Init (&argc , &argv) ;
18
19
20   int val;
21
22   if (pid==0)
23         val = 10;
24
25   Diffusion(val);
26
27         // sur tous les processeurs, val a pour valeur 10.
28
29   MPI_Finalize( ) ;
30   return 0 ;
31 }
```