UNIVERSITÉ D'ORLÉANS **M1 Info**
Informatique **2019 - 2020**

Examen: Programmation Parallèle - 2h

Exercice 1. Questions diverses (4pts)

1. Soit une matrice T représentant un Modèle Numérique de Terrain (MNT) de taille $N \times M$ où chaque élément T_{ij} mémorise la hauteur du terrain.

   (a) On souhaite définir une nouvelle matrice M initialisée à T avec chaque élément M_{ij} pour $0 < i < N-1$ et $0 < j < M-1$ défini par

   $$M_{ij} = \min_{-1 \leq l_1, l_2 \leq 1} T_{(i+l_1)(j+l_2)}$$

   i. Quel type de calcul effectue-t-on ?

   ii. Si on distribue le terrain en bandes horizontales de taille $n \times M$ sur $nprocs$ processeurs tel que $n = \frac{N}{nprocs}$ quel est la taille des ghosts nécessaires sur chaque processeur (N divisible par $nprocs$)?

   iii. Même question si on distribue le terrain en blocs de taille $n \times m$. On supposera que $nprocs = p_1 \times p_2$ tel que $n = \frac{N}{p_1}$ et $m = \frac{M}{p_2}$ (N divisible par p_1 et M divisible par p_2).

   (b) Pour la répartition du terrain en bandes horizontales avec toujours les mêmes hypothèses quelle fonction collective MPI utiliseriez vous pour distribuer T en supposant qu'un processeur *root* est le seul à avoir le terrain complet.

2. Corrigez le programme ci-dessous afin qu'il calcule réellement une estimation du nombre Π

```
1
2  int main (int argc, char* argv[]){ // 2 arguments attendus
3    long i;
4    long dansDisque = 0;
5    double pi, x, y;
6    double r = 1.0; // rayon du cercle.
7
8    long nb_essais = atoi(argv[1]);
9    int nthreads = atoi(argv[2]);
10
11 #pragma omp parallel num_threads(nthreads)
12   {
13 #pragma omp for
14     for(i=0; i<nb_essais; i++) {
15       x = mrandom();
16       y = mrandom();
17       if (( x*x + y*y) <= r*r)
18           dansDisque++;
19     }
20   }
21   pi = 4.0 * ((double)dansDisque/(double)(nb_essais));
22 }
```

Exercice 2. Que fait ce programme ? (4pts)

```
1  void Mystere(int* ptr_a, int root) {
2      int pid, nprocs;
3      MPI_Comm_rank(MPI_COMM_WORLD, &pid);
4      MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
5      MPI_Status status;
6
7      int nb_etape = ceil((log((double) nprocs) / log(2.0)));
8      for (int i=0; i<nb_etape; i++) {
9          int size = pow(2,i);
10         if ((pid-root+nprocs)%nprocs<size) {
11             int pid_dest = (pid+size+nprocs)%nprocs;
12             if ((pid-root+nprocs)%nprocs+size<nprocs)
13                 MPI_Ssend(ptr_a, 1, MPI_INT, pid_dest, 1, MPI_COMM_WORLD);
14             }
15         else if ((pid-root+nprocs)%nprocs<pow(2,i+1)){
16             int pid_src = (pid - size + nprocs)%nprocs;
17             MPI_Recv(ptr_a,1,MPI_INT,pid_src,1,MPI_COMM_WORLD,&status);
18         }
19     }
20 }
21 int main(int argc, char **argv) {
22     int pid, nprocs;
23     MPI_Init(&argc, &argv);
24     MPI_Comm_rank(MPI_COMM_WORLD, &pid);
25     MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
26     int root = atoi(argv[1]); // le processeur root
27     int a;
28     if (pid==root) {
29         srand(time(NULL));
30         a = rand() % 10;
31     }
32     Mystere(&a,root);
33     MPI_Finalize();
34     return 0;
35 }
```

1. Quel est le rôle de la fonction *Mystere* ?
2. Quelle est sa complexité ?

Exercice 3. Sudoku encore et toujours en MPI (6pts)

Une grille de Sudoku généralisée est une matrice S de taille $k^2 \times k^2$ qui se décompose en $k \times k$ blocs de taille $k \times k$ où chaque bloc contient tous les nombres de 1 à k^2. 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^2.

La distribution de la grille est fixée. Le processeur *root* connaissant la grille complète la décompose en bandes horizontales de taille $k \times k^2$.

1. Si on dispose de $nprocs$ processeurs tel que $nprocs = k$ la distribution de la grille consiste à affecter chaque bande horizontale à un processeur.

   Donnez l'algorithme qui vous permet de vérifier que la grille est bien une grille Sudoku. Vous pouvez suivre les étapes suivantes

(a) Comment procéder pour la vérification des lignes ?

(b) Comment procéder pour la vérification des blocs ?

(c) Comment procéder pour la vérification des colonnes sachant que vous ne devez pas réunir une colonne sur un processeur (vous pouvez utiliser un tableau qui permet de savoir si un élément est présent ou pas) ?

(d) Comment le résultat global de la vérification est-il calculé ?

(e) Quelles sont les fonctions MPI nécessaires à la mise en œuvre ? Justifiez.

2. Que doit-on changer si désormais on dispose de $nprocs$ processeurs tel que $nprocs < k$ et $nprocs$ divise k ?

Exercice 4. Séparation pair/impair en OpenMP(6pts)

A partir d'un tableau T de taille n, on souhaite construire un tableau T' contenant à gauche les éléments pairs de T et à droite les éléments impairs. Par exemple si $T = \{1, 2, 3, 4, 5, 6\}$ alors $T' = \{2, 4, 6, 1, 3, 5\}$. Pour accélérer le calcul de T' on souhaite écrire un programme OpenMP le plus efficace possible

1. Décrivez le principe de la parallélisation à l'aide d'un schéma ou en pseudo-langage
2. Écrivez le programme OpenMP correspondant à votre algorithme. La génération du tableau T n'est pas à écrire.
3. Quelle est la complexité de votre parallélisation ?