UNIVERSITÉ D'ORLÉANS **M1 Info**

Informatique Année **2015 - 2016**

Contrôle continu : Programmation Parallèle - 1h30

Exercice 1. Que fait ce programme ? (5pts)

Expliquez le calcul effectué par ce programme et en particulier que vaut *vec* à la fin sur le processeur *root*.

```
int main ( int argc , char **argv ) {
  int pid, nprocs;
  MPI_Status status;
  MPI_Init (&argc , &argv) ;
  MPI_Comm_rank(MPI_COMM_WORLD, &pid ) ;
  MPI_Comm_size (MPI_COMM_WORLD, &nprocs ) ;

  int n = atoi(argv[1]);
  int root = atoi(argv[2]);

  int* mat = NULL;
  int* vec = new int[n];
  int* mat_local;

  srand(time(NULL)+pid);
  if (pid==root) {
    mat = new int[n*n];
    for (int i=0; i<n*n; i++)
      mat[i] = rand()%10;
    }
    for (int i=0; i<n; i++)
      vec[i] = rand()%10;
  }
  MPI_Bcast(vec,n,MPI_INT,root,MPI_COMM_WORLD);

  int nprocs_gauche = n%nprocs;
  int n_local = n/nprocs;
  int sendptr = 0;
  int recvptr = 0;
  int* sendcounts = NULL;
  int* recvcounts = NULL;
  int* senddispls = NULL;
  int* recvdispls = NULL;

  if (pid == root) {
    sendcounts = new int[nprocs];
    recvcounts = new int[nprocs];
    senddispls = new int[nprocs];
    recvdispls = new int[nprocs];
    for (int i=0; i<nprocs; i++) {
      if (i<nprocs_gauche) {
          sendcounts[i] = (n_local+1)*n;
          senddispls[i] = sendptr;
          sendptr+=sendcounts[i];
          recvcounts[i] = (n_local+1);
          recvdispls[i] = recvptr;
```

```
            recvptr+=recvcounts[i];
         }
         else {
            sendcounts[i] = n*n_local;
            senddispls[i] = sendptr;
            sendptr+=sendcounts[i];
            recvcounts[i] = n_local;
            recvdispls[i] = recvptr;
            recvptr+=recvcounts[i];
         }
   }
   if (pid<nprocs_gauche)
       n_local = n_local+1;

   mat_local = new int[n_local*n];

   MPI_Scatterv(mat,sendcounts,senddispls,MPI_INT,mat_local,n_local*n,MPI_INT,root,MPI_COMM_WORLD);

   int* vec_local = new int[n_local];

   for (int i=0; i<n_local; i++) {
     vec_local[i] = 0;
     for (int j=0; j<n; j++)
       vec_local[i] = vec_local[i] + mat_local[i*n+j]*vec[j];
   }

   MPI_Gatherv(vec_local,n_local,MPI_INT,vec,recvcounts,recvdispls,MPI_INT,root,MPI_COMM_WORLD);

   // Destruction de la mémoire allouée
   // ....
   MPI_Finalize() ;
   return 0 ;
}
```

Exercice 2. Communications (5pts)

A partir du communicateur `MPI_COMM_WORLD` contenant p processeurs, on souhaite construire une topologie cartésienne telle que $p = p_1 \times p_2$

1. Donnez les lignes de code MPI qui permettent de réaliser cette tolopogie. On supposera que p est divisible par p_1 et que $p_2 = \frac{p}{p_1}$.
2. Codez la fonction de communication qui permet à tous les processeurs d'une colonne de la grille d'envoyer une donnée scalaire aux processeurs de sa ligne. Les autres processeurs doivent recevoir également la donnée envoyée. La signature de la fonction est `void envoi_ligne(int* a, int col, MPI_Comm com_grille)` où a est le scalaire à envoyer ou recevoir, col est le numéro de la colonne des processeurs qui envoient a et `com_grille` le communicateur construit précédemment.

Exercice 3. Suite de Syracuse (10pts)

Une suite de Syracuse est une suite telle que $U_0 = x$ avec $x > 0$ et

$$U_n = \begin{cases} \frac{U_{n-1}}{2} & si\ U_{n-1}\ est\ pair \\ 3U_{n-1} + 1 & sinon \end{cases}$$

On souhaite vérifier si un tableau d'entiers de taille N correspond à une suite de Syracuse alors que ce tableau est réparti sur p processeurs.

1. A partir d'un exemple, expliquez le principe de la parallélisation en supposant que le tableau est déjà distribué et que N est divisible par p et que tous les processeurs doivent avoir le résultat du test.
2. Explicitez les fonctions de communications MPI que vous allez utiliser et pourquoi.
3. Ecrivez la fonction correspondante à partir du squelette ci-dessous

```
// tab est le tableau local
// n_local est la taille du tableau en local
bool test_syracuse(int* tab, int n_local)
{
  int pid, nprocs;
  MPI_Comm_rank(MPI_COMM_WORLD, &pid );
  MPI_Comm_size (MPI_COMM_WORLD, &nprocs );

  bool test;

  // A compléter

  return test;

}
```