UNIVERSITÉ D'ORLÉANS **M1 Info**

Informatique **2014 - 2015**

Session 1: Programmation Parallèle - 2h

Exercice 1. QuickSort (5pts)

Décrivez sans donner l'implémentation l'algorithme de tri QuickSort sur une architecture à mémoire distribuée. Vous pourrez illustrer cette description à partir d'un exemple avec $n = 14$ la taille du tableau à trier et $nprocs = 4$ le nombre de processeurs disponible.

Exercice 2. Parallélisation du tri par induction (8pts)

Soit l'algorithme suivant qui réalise un tri appelé tri par induction

```
for i from 1 to n do
  p = 1
  for j from 1 to n do
    if (E(j)<E(i)) then
      p = p + 1
    end if
  end for
  T(p) = E(i)
end for
```

Soit E un tableau d'entiers tous distincts, on souhaite paralléliser la première boucle for (indice i) en répartissant le calcul des positions p sur $nprocs$ processeurs. Initialement le tableau E est réparti sur les nprocs processeurs. De plus à la fin du tri, on souhaite avoir le tableau T également réparti sur les nprocs processeurs :

1. (4pts) Donnez un exemple pour $n = 14$ et $nprocs = 4$ du tri par induction paralléle en illustrant par un schéma le principe de la parallélisation.
2. (1pt) Par rapport à ces étapes, explicitez les fonctions de communications MPI que vous allez utiliser et pourquoi.
3. (3pts) A partir de la trame ci-dessous proposez une implémentation MPI. Pour certains calculs (des indices par exemple), vous pouvez utiliser une fonction que vous définirez clairement sans l'implémenter.

```
int main ( int argc , char **argv ) {
  int pid, nprocs;
  MPI_Init (&argc , &argv) ;
  MPI_Comm_rank(MPI_COMM_WORLD, &pid ) ;
  MPI_Comm_size (MPI_COMM_WORLD, &nprocs ) ;
  int n_local = atoi(argv[1]);
  int root = atoi(argv[2]);
  srand(time(NULL)+pid);
  int* E_local = new int[n_local];
  generation_elts_distincts(E_local,n_local); // fonction pour générer un tableau initial
  delete[] E_local;
  MPI_Finalize() ;
  return 0 ;
}
```

Exercice 3. Le jeu de la vie (7pts)

Cet exercice se base sur l'énoncé du projet du 12 décembre mais en supposant que les ghosts sont de taille 1 et que le voisinage est constitué des 8 voisins (droite, gauche, bas, haut et sur les diagonales).

- Donnez l'implémentation de la fonction de profil

```
bool shift_diagonal1_haut(void* send, int count, MPI_Datatype datatype, void* recv, MPI_Comm Comm)
```

Comm est le communicateur d'une topologie cartésienne de $p_1 \times p_2$ processeurs et la diagonale 1 désigne la diagonale allant du bas à gauche vers le haut à droite.

Désormais, vous disposez de toutes les fonctions suivantes qui effectuent les échanges où Comm est le communicateur d'une topologie cartésienne de $p_1 \times p_2$ processeurs.

```
bool shift_droite(void* send, int count, MPI_Datatype datatype, void* recv,  MPI_Comm Comm);
bool shift_gauche(void* send, int count, MPI_Datatype datatype, void* recv, MPI_Comm Comm);
bool shift_haut(void* send, int count, MPI_Datatype datatype, void* recv,  MPI_Comm Comm);
bool shift_bas(void* send, int count, MPI_Datatype datatype, void* recv, MPI_Comm Comm);
bool shift_diagonal1_haut(void* send, int count, MPI_Datatype datatype, void* recv, MPI_Comm Comm);
bool shift_diagonal1_bas(void* send, int count, MPI_Datatype datatype, void* recv, MPI_Comm Comm);
bool shift_diagonal2_haut(void* send, int count, MPI_Datatype datatype, void* recv, MPI_Comm Comm);
bool shift_diagonal2_bas(void* send, int count, MPI_Datatype datatype, void* recv, MPI_Comm Comm);
```

Soit le programme principal ci-dessous qui réalise le jeu de la vie

```
int main ( int argc , char **argv )
{
  int pid, nprocs;
  MPI_Init (&argc , &argv) ;
  MPI_Comm_rank(MPI_COMM_WORLD, &pid ) ;
  MPI_Comm_size (MPI_COMM_WORLD, &nprocs ) ;
  MPI_Comm COMM_CART;
  srand(time(NULL)+pid);
  int n = atoi(argv[1]);
  int m = atoi(argv[2]);
  int nbiter = atoi(argv[3]);
  int percent = atoi(argv[4]);
  int root = atoi(argv[5]);

  int* nbprocs_dims = new int[2];
  nbprocs_dims[0] = 0;
  nbprocs_dims[1] = 0;
  int* period = new int[2];
  period[0] = 1;
  period[1] = 1;

  MPI_Dims_create(nprocs, 2, nbprocs_dims);
  MPI_Cart_create(MPI_COMM_WORLD, 2, nbprocs_dims, period, false, &COMM_CART);
  int* mycoords = new int[2];
  MPI_Cart_coords(COMM_CART, pid, 2, mycoords);

  int* jeu = NULL;
  if (pid==root) {
    jeu = new int[n*m];
    // fonction de génération d'une grille avec percent% de cellules à 1 et le reste à 0
    generation_int(n,m,jeu,percent);
```

```
    for (int i=0; i<n*m; i++)
      cout << jeu[i] << " ";
    cout << endl;
  }

  int n_local = n/nbprocs_dims[0];
  int m_local = m/nbprocs_dims[1];
  int* jeu_local = new int[n_local*m_local];

  // le processeur root distribue la grille initiale par bloc
  distribution_int(jeu,n,m,nbprocs_dims,jeu_local,root,COMM_CART);

  int* grille = new int[(n_local+2)*(m_local+2)];
  for (int i=0; i<(n_local+2)*(m_local+2); i++)
    grille[i] = 0;

  for (int i=0; i<n_local; i++)
    for (int j=0; j<m_local; j++)
      grille[(i+1)*(m_local+2)+1+j] = jeu_local[i*m_local+j];

  for (int i=0; i<nbiter; i++) {
    echange_ghost_int(grille, n_local, m_local, COMM_CART); // A IMPLÉMENTER
    evolution(grille,n_local,m_local); // application des règles d'évolution des éléments de la grille
  }

  for (int i=0; i<n_local; i++)
    for (int j=0; j<m_local; j++)
      jeu_local[i*m_local+j] = grille[(i+1)*(m_local+2)+1+j];

  // Rassemblement des blocs de la grille sur le processeur root
  rassemblement_int(jeu,n,m,nbprocs_dims,jeu_local,root,COMM_CART); // A IMPLÉMENTER

  if (pid==root) {
    cout << "jeu final" << endl;
    for (int i=0; i<n; i++)
      for (int j=0; j<m; j++)
         cout << jeu[i*m+j] << " ";
     cout << endl;
  }

  delete[] nbprocs_dims;
  delete[] period;
  delete[] mycoords;
  if (jeu!=NULL)
    delete[] jeu;

  MPI_Finalize();

  return 0 ;
}
```

- Donnez l'implémentation des deux fonctions `echange_ghost_int` et `rassemblement_int`
- Modifiez le programme principal pour qu'il permette à un programme MPI esclave de récupérer une itération sur 10 de l'évolution de la grille pour afficher les valeurs correspondantes.