Session 1: Calcul Intensif

Exercice 1. Analyse d'un programme PRAM

Expliquez ce que fait le programme PRAM ci-dessous sachant qu'il travaille sur une matrice initiale M_1 de taille n^2 déjà en mémoire. De plus, ce programme suppose une grille de processeurs numérotés (i, j, k_1, k_2) tels que $0 \leq i, j \leq n-1$ et $-1 \leq k_1, k_2 \leq 1$. Détaillez pour chacune des trois phases la complexité si c'est possible ou une idée de la complexité, sur quels types de machine PRAM elles fonctionnent et enfin analysez les défauts du programme par rapport à ce qu'il devrait faire.

Les déclarations ne sont pas indiquées dans le programme. Mais les tableaux tab et ind sont des variables globales de taille $n \times n \times 9$. Les matrices calculées M_2 et M_3 sont aussi des variables globales de taille $n \times n$.

```
C1[9]={1,2,3,8,0,4,7,6,5}
C2[9]={5,6,7,4,0,8,3,2,1}
if (j!=0 & i!=0) then
//Phase 1
  k3 = k1+k2+2*(k1+2)
  gw(k3,ind[i,j,k3])
  gr(M1[i+k1,j+k2],a)
  gw(a,tab[i,j,k3])
  for h from 1 to log2(8)
    if k3<8/2**h-1 then
      gr(tab[i,j,2*k3],x)
      gr(tab[i,j,2*k3+1],y)
      if (x<=y) then
        gw(x,tab[i,j,k3])
        gw[ind[i,j,2*k3],ind[i,j,k3]) //petit raccourci PRAM
      else
        gw(y,tab[i,j,k3])
        gw[ind[i,j,2*k3+1],ind[i,j,k3]) //petit raccourci PRAM
      end if
    end if
  end for
  //Attention 9 n'est pas une puissance de 2
  if (k3==0) then
    gr(tab[i,j,8],x)
    gr(tab[i,j,0],y)
    if (x<=y) then
      gw(C1[8],M2[i,j]) //toujours le petit raccourci PRAM
    else
      gr(ind[i,j,0],a)
      gr(C1[a],b)
      gw(b,M2[i,j])
    end if
  end if
// Phase 2
  if (k3==0)
    gr(M2[i,j],b)
    if (b==0) then
      M3[i,j]=i*n+j
    else
      M3[i,j]=-1
    end if
  end if
// Phase 3
  Repeat
    if (k1!=0 & k2!=0) then
      gr(M3[i,j],a)
      if (a!=-1) then
```

```
        gr(M2[i+k1,j+k2],b)
        gr(C2[k3],c)
        if (b == c) then
          gw(M3[i,j],M3[i+k1,j+k2]) // encore un petit raccourci PRAM
        end if
        exit
      end if
    else
      exit
    end if
  end Repeat
end if
```

Quelques indications :

- Les correspondances entre les valeurs de k_1, k_2 et k_3 sont les suivantes

(k_1, k_2)	k_3
(-1,-1)	0
(-1,0)	1
(-1,1)	2
(0,-1)	3
(0,0)	4
(0,1)	5
(1,-1)	6
(1,0)	7
(1,1)	8

- Pour dérouler l'algorithme, essayez de fixer la valeur de i et j. Par exemple soit un extrait de la matrice $M1$:

$$M_1 = \begin{matrix} 12 & 5 & 8 & 15 & \dots \\ 6 & 13 & 5 & 7 & \dots \\ 7 & 4 & 12 & 7 & \dots \\ 6 & 13 & 12 & 12 & \dots \\ \dots & \dots & \dots & \dots & \end{matrix}$$

1. si vous considérez $i = j = 2$ essayez de calculer $M_2[2, 2]$,
2. si vous considérez $i = 2$, $j = 1$ essayez de calculer $M_2[2, 1]$,
3. enfin si vous considérez $i = 2$, $j = 1$ quelle valeur prendra $M_3[2, 2]$?

Exercice 2. Le jeu de la vie en MPI

On souhaite réaliser le jeu de la vie sur une grille de cellules distribuée sur une grappe de calculs.

Le jeu de la vie qu'on souhaite implémenter est un automate cellulaire qui consiste en une grille de $cellules$ pouvant chacune prendre à un instant donné un $état$ parmi deux (vivante ou morte). L'état d'une cellule au temps t est fonction de l'état au temps $t-1$ des **4** cellules voisines (on ne tient pas compte des cellules voisines en diagonale). À chaque nouvelle unité de temps, les mêmes règles sont appliquées pour toutes les cellules de la grille, produisant une nouvelle $génération$ de cellules dépendant entièrement de la génération précédente.

L'évolution de l'état d'une cellule dépend de l'état de ses quatre voisins :

- une cellule morte à l'étape $t-1$ et ayant exactement 2 voisins sera vivante à l'étape suivante ;
- une cellule vivante à l'étape t ne restera vivante que si 1 ou 2 voisins le sont à l'étape $t-1$.

On suppose que la grille est "fermée" et que les voisins du bord gauche de la grille sont les cellules du bord droit etc.

Proposez une implémentation MPI du jeu de la vie sachant que

1. Vous disposez de $P = p \times p$ processeurs avec p divisant n sachant que la grille est de taille $n \times n$.
2. Les cellules de la grille sont initialement distribuées sur la grappe de calculs par bloc de taille $\frac{n}{p} \times \frac{n}{p}$.
3. Vous devez utiliser une topologie cartésienne 2D pour manipuler les numéros de processeurs ayant besoin de communiquer.

4. Vous devez utiliser des fonctions de communications bloquantes comme MPI_Ssend et MPI_Recv pour les échanges de données.

5. Pour la gestion des bords haut/bas/droite/gauche, vous pouvez définir des fonctions que vous spécifierez très précisément mais dont vous pourrez ne pas donner l'implémentation. Ces fonctions ne doivent pas contenir de communications.

```
#include <mpi.h>

//Initialisation aleatoire d'une grille de taille sizexsize.
void Initialisation(bool* G, int size);
//Pour definir l'etat d'une cellule d'indice i, j dans un tableau
// a une dimension de longueur nxn.
bool EstVivante(bool* G, int i, int j, int n);

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 p = (int)sqrt(nprocs);

  int nbstep = atoi(argv[1]);
  int sizelocal = atoi(argv[2]);
  int n = p*sizelocal;

  // les ?? sont a remplacer par les bonnes valeurs

  bool* grille = (bool*)malloc(sizeof(bool)*???*???);
  Initialisation(???,???);

  MPI_Comm Grid2D;
  int dims[2];
  int periods[2];

  dims[0] = ??;
  dims[1] = ??;


  periods[0] = ??;
  periods[1] = ??;

  MPI_Cart_create(MPI_COMM_WORLD,2,dims,periods,1,&Grid2D);

  for (int i=0; i<nbstep; i++) {
    // A completer
  }


  MPI_Finalize() ;
  return 0 ;
}
```

Listing 1: Trame du programme MPI du jeu de la vie.

Exercice 3. Le calcul des k-moyennes

L'objectif de cet exercice est de proposer une implémentation en MPI de l'algorithme des k-moyennes vu en cours. A partir des contraintes ci-dessous, proposez une implémentation MPI réalisant cet algorithme. Vous terminerez par une analyse de votre implémentation : avantages/inconvénients. De plus n'hésitez pas à commenter votre code.

1. Chaque nœud de la grappe a accès au fichier contenant la base de données des clients représentés par n flottants.

2. Vous pouvez utiliser une fonction `void lecture(int start, int nbclient, float* clients)` qui permet de lire le fichier prédéfini des clients à partir du client numéro $start$ et d'initialiser le tableau une dimension $clients$ de taille $nbclient \times n$.

3. Vous pouvez utiliser une fonction `int dist(int numeroclient, int n, float *k, float* clients)` qui calcule le numéro du centre de la classe à laquelle appartient le client de numéro `numeroclient`. Le tableau k à une dimension et de taille $k \times n$ contient les k centres des classes.

4. Vous ne pouvez utiliser que les fonctions de communication MPI_Bcast et MPI_Reduce avec l'opération MPI_Sum.

```
#include <mpi.h>

void lecture (int start, int nbclient, float* clients); //Ne pas ecrire

int dist (int numeroclient, int n, float* k, float* clients); // Ne pas ecrire

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 nbclienttotal = atoi(argv[1]); // Nombre de clients dans la base de donnees
  int k = atoi(argv[2]); // Le nombre de classes
  int nbstep = atoi(argv[3]); // Le nombre d'iteration souhaite
  int n = atoi(argv[4]); // Le nombre de flottants representant un client

  int nbclient = floor((float)nbclienttotal/(float)nprocs);
  if (pid==nprocs-1)
    nbclient = nbclienttotal-(nprocs-1)*nbclient;

  float* clients = (float*) malloc(???); // ??? A completer.
  float* lescentres = (float*) malloc(???); // ??? A completer

  int* classe = (int*) malloc(???); // ??? A completer
  //telle que classe[i] est le numero du centre de la classe a laquelle le client i appartient.

  lecture(pid*nbclient+1,nbclient,clients);

  // Initialisation des centres
  // A Completer si necessaire.

  for (int i=0; i<nbstep; i++) {
    // A completer
  }

  MPI_Finalize() ;
  return 0 ;
}
```

Listing 2: Trame du programme MPI des k-moyennes.

Rappel : L'algorithme des k-moyennes consiste, à partir d'une base de données de clients par exemple, à construire k classes par calcul progressif des centres de ces classes. Les centres initiaux peuvent être choisis aléatoirement parmi les clients. Ensuite chaque client est affecté à la classe dont le centre est le plus proche. A la fin d'une itération on calcule le nouveau centre d'une classe à partir de la moyenne des clients qui la constitue.