UNIVERSITÉ D'ORLÉANS **M1 Info**
Informatique Année **2011 - 2012**

Contrôle continu : Calcul Intensif

Exercice 1. Calcul de Pi en parallèle

Soit le programme suivant qui permet d'approcher la valeur de π comme la valeur de $\frac{1}{4n}\sum_{i=1}^{n} f(\frac{i}{n}) + f(\frac{n-1}{n})$.

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <iomanip>

using namespace std;

double fx(double x)
{
  double res =(double)4.0/(1+x*x);
  return res;
}

double SommePartielle(int indmin, int indmax, int n)
{

  double somme = 0;
  double x=0;
  for (int i=indmin; i<=indmax; i++) {
    x = (double)i/(double)n;
    somme += fx(x);
  }
  return somme;
}

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 taille = atoi(argv[1]);

  int n = taille*nprocs;
  double MaSomme = SommePartielle(pid*taille ,(pid+1)*taille -1,n);

  double Pi = 0;
  MPI_Reduce(&MaSomme, &Pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

  if (pid==0) {
    Pi = 0.25*(Pi + fx(((double)taille -1.0)/(double)taille))/(double)taille;
    cout << "PI=" << setprecision (15) << Pi << endl;
  }
  MPI_Finalize() ;
  return 0 ;
}
```

Listing 1: Un premier programme MPI pour calculer π.

1. Nous souhaitons que l'utilisateur rentre lui même la valeur du paramètre *taille* sur le processeur 0. Donnez la ligne à rajouter dans le programme ci-dessous pour que le calcul continue à fonctionner.

```
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 taille = 0;
  if (pid==0) {
    cout << "Donner la taille de la somme par processeur " << endl;
    cin >> taille;
  }
  //LIGNE A RAJOUTER ICI

  int n = taille*nprocs;
  // ETC ... Code identique

}
```

Listing 2: Lecture de taille.

2. Proposez également une méthode pour ne pas utiliser la fonction *MPI_Reduce* et complétez alors le programme suivant

```
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_Status status;
  int taille = atoi(argv[1]);

  int n = taille*nprocs;
  double MaSomme = SommePartielle(pid*taille ,(pid+1)*taille -1,n);

  double Pi = 0;
  //RAJOUTER ICI LE CODE REMPLACANT MPI_Reduce

  if (pid==0) {
    Pi = 0.25*(MaSomme + fx(((double)taille -1.0)/(double)taille))/(double)taille;
    cout << "PI=" << setprecision (15) << Pi << endl;
  }
  MPI_Finalize() ;
  return 0 ;
}
```

Listing 3: Dans réduction.

Exercice 2. Que fait ce programme ?

Donnez en quelques mots le rôle du programme ci-dessous. Explicitez la valeur de *recu* pour chaque processeur numéroté dans *MPI_COMM_WORLD*.

```
#include <stdio.h>
#include "mpi.h"

using namespace std;

int main (int argc, char* argv[])
{
  int nb, pid;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD,&nb);
  MPI_Comm_rank(MPI_COMM_WORLD,&pid);

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

  dims[0] = 3;
  dims[1] = 3;


  periods[0] = 1;
  periods[1] = 1;
  periods[2] = 1;

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

  int Coords[2];
  MPI_Cart_coords(Grid2D,pid,2,Coords);

  int val = pid+100;
  int recu = 0;
  int previous, next;
  int err = MPI_Cart_shift(Grid2D,1,1,&previous,&next);

  MPI_Request request;

  MPI_Isend(&val,1,MPI_INT,next,150,MPI_COMM_WORLD,&request);
  MPI_Recv(&recu,1,MPI_INT,previous,150,MPI_COMM_WORLD,MPI_STATUS_IGNORE);

  cout << "pid = " << pid << " recu = " << recu << endl;

  MPI_Finalize();
}
```

Listing 4: Programme à interpréter.