UNIVERSITÉ D'ORLÉANS **Programmation Parallèle**

M1 Info Année **2025 - 2026**

Série de Travaux Dirigés : **1** *- L'API MPI et la parallélisation*

Exercice 1. Lecture de codes

A partir des codes suivants expliquez ce que fait le programme. Vous pouvez donner un exemple en indiquant ce qu'il se passe pour chaque processus. On supposera qu'on a exécuté le programme avec `mpirun -np 4`.

1. La fonction `CommAComprendre`

```
void CommAComprendre(int n, int* in, int* out)
{
  int pid, nprocs;
  MPI_Comm_rank(MPI_COMM_WORLD, &pid );
  MPI_Comm_size (MPI_COMM_WORLD, &nprocs );
  int* tabtmp = new int[n/2];
  int pid_envoi = (pid+1)%nprocs;
  int pid_recu = (pid-1+nprocs)%nprocs;
  if (pid%2==0){
    MPI_Send(in+n/2,n/2,MPI_INT,pid_envoi,34,MPI_COMM_WORLD);
    MPI_Recv(out,n/2,MPI_INT,pid_recu,34,MPI_COMM_WORLD,MPI_STATUS_IGNORE);}
  else {
    MPI_Recv(out,n/2,MPI_INT,pid_recu,34,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
    MPI_Send(in+n/2,n/2,MPI_INT,pid_envoi,34,MPI_COMM_WORLD);}
  pid_envoi = (pid-1+nprocs)%nprocs;
  pid_recu = (pid+1)%nprocs;
  if (pid%2==0){
    MPI_Send(in,n/2,MPI_INT,pid_envoi,34,MPI_COMM_WORLD);
    MPI_Recv(out+n/2,n/2,MPI_INT,pid_recu,34,MPI_COMM_WORLD,&status);  }
  else {
    MPI_Recv(out+n/2,n/2,MPI_INT,pid_recu,34,MPI_COMM_WORLD,&status);
    MPI_Send(in,n/2,MPI_INT,pid_envoi,34,MPI_COMM_WORLD); }
}
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 nlocal = 2*atoi(argv[1]);
  int* tablocal = new int[nlocal];
  int* tabres = new int[nlocal];
  srand(time(NULL)+pid);
  for (int i=0; i<nlocal; i++)
    tablocal[i] = rand()%10;
  CommAComprendre(nlocal,tablocal,tabres);
  delete[] tablocal;
  delete[] tabres;
  MPI_Finalize() ;
  return 0 ; }
```

2. Fonction Mystère

```
void Mystere(int* ptr_a, int root) {
```

```
    int pid, nprocs;
    MPI_Comm_rank(MPI_COMM_WORLD, &pid);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Status status;

    int nb_etape = ceil((log((double) nprocs) / log(2.0)));
    for (int i=0; i<nb_etape; i++) {
        int size = pow(2,i);
        if ((pid-root+nprocs)%nprocs<size) {
            int pid_dest = (pid+size+nprocs)%nprocs;
            if ((pid-root+nprocs)%nprocs+size<nprocs)
                MPI_Ssend(ptr_a, 1, MPI_INT, pid_dest, 1, MPI_COMM_WORLD);
            }
        else if ((pid-root+nprocs)%nprocs<pow(2,i+1)){
            int pid_src = (pid - size + nprocs)%nprocs;
            MPI_Recv(ptr_a,1,MPI_INT,pid_src,1,MPI_COMM_WORLD,&status);
        }
    }
}
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 root = atoi(argv[1]); // le processeur root
    int a;
    if (pid==root) {
        srand(time(NULL));
        a = rand() % 10;
    }
    Mystere(&a,root);
    MPI_Finalize();
    return 0;
}
```

Exercice 2. Réduction sur un tableau

1. Calcul du maximum d'un tableau d'entiers

   Complétez et modifiez le code ci-dessous afin de répartir le calcul du maximum d'un tableau d'entiers sur plusieurs processus. Un processus **root** unique dispose du résultat final.

```
int fn_max(int n, int* tab) {
    int max = tab[0];
    for (int i=1; i<n; i++)
        if (tab[i]>max)
            max = tab[i];
    return max;
}
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 ) ;
```

```
14
15    int n = atoi(argv[1]);
16    int root = atoi(argv[2]);
17
18
19    int* tab;
20    if (pid==root) {
21        tab = new int[n];
22        srand(time(NULL));
23        for (int i = 0; i < n; i++)
24            tab[i] = rand() % 1000;
25    }
26
27    int max;
28
29    // ??????
30
31
32    if (pid==root)
33        cout << "max final = " << max << endl;
34
35
36   if (pid==root)
37      delete[] tab;
38    MPI_Finalize() ;
39    return 0 ;
40  }
```

2. Calcul du maximum et de la position de ce maximum dans un tableau d'entiers

   Modifiez le code précédent pour l'adapter à la recherche du maximum d'un tableau et de sa position dans ce tableau. Vous pourrez utiliser le type `MPI_2INT` et l'opération `MPI_MAXLOC`.

Exercice 3. Une valeur approchée de π

La valeur de π peut être calculée par l'intégrale $\int_0^1 f(x)$ avec $f(x) = \frac{4}{1+x^2}$. Cette intégrale peut être approchée par $\pi \approx \sum_{i=0}^{n} f(x_i)\Delta x$ où $\Delta x = \frac{1}{n}$ et $x_i = \frac{1}{2n} + i\frac{1}{n}$

Plus n est grand plus on se rapproche de π. Proposez une parallélisation du calcul de π pour permettre de choisir un n très grand.

Exercice 4. Une suite de Syracuse

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

$$U_i = \left| \begin{array}{ll} \frac{U_{i-1}}{2} & si\ U_{i-1}\ est\ pair \\ 3U_{i-1} + 1 & sinon \end{array} \right.$$

On souhaite vérifier si un tableau d'entiers U de taille n correspond à une suite de Syracuse alors que ce tableau est initialement sur le processeur `root`.

1. Décrivez le principe de la parallélisation en indiquant un ordre de grandeur de la complexité parallèle. Vous pouvez faire un schéma sur un exemple.

2. Est-il nécessaire d'avoir des communications au-delà de la distribution initiale de U.

3. Quelles sont les fonctions MPI que vous allez utiliser (en justifiant) ?