UNIVERSITÉ D'ORLÉANS **Programmation Parallèle**
M1 ARIAS Année **2026-2027**

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`

```
1  void CommAComprendre(int n, int* in, int* out)
2  {
3    int pid, nprocs;
4    MPI_Comm_rank(MPI_COMM_WORLD, &pid );
5    MPI_Comm_size (MPI_COMM_WORLD, &nprocs );
6    int* tabtmp = new int[n/2];
7    int pid_envoi = (pid+1)%nprocs;
8    int pid_recu = (pid-1+nprocs)%nprocs;
9    if (pid%2==0){
10     MPI_Send(in+n/2,n/2,MPI_INT,pid_envoi,34,MPI_COMM_WORLD);
11     MPI_Recv(out,n/2,MPI_INT,pid_recu,34,MPI_COMM_WORLD,MPI_STATUS_IGNORE);}
12   else {
13     MPI_Recv(out,n/2,MPI_INT,pid_recu,34,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
14     MPI_Send(in+n/2,n/2,MPI_INT,pid_envoi,34,MPI_COMM_WORLD);}
15   pid_envoi = (pid-1+nprocs)%nprocs;
16   pid_recu = (pid+1)%nprocs;
17   if (pid%2==0){
18     MPI_Send(in,n/2,MPI_INT,pid_envoi,34,MPI_COMM_WORLD);
19     MPI_Recv(out+n/2,n/2,MPI_INT,pid_recu,34,MPI_COMM_WORLD,&status);  }
20   else {
21     MPI_Recv(out+n/2,n/2,MPI_INT,pid_recu,34,MPI_COMM_WORLD,&status);
22     MPI_Send(in,n/2,MPI_INT,pid_envoi,34,MPI_COMM_WORLD); }
23 }
24 int main ( int argc , char **argv )
25 {
26   int pid, nprocs;
27   MPI_Init (&argc , &argv) ;
28   MPI_Comm_rank(MPI_COMM_WORLD, &pid ) ;
29   MPI_Comm_size (MPI_COMM_WORLD, &nprocs ) ;
30   int nlocal = 2*atoi(argv[1]);
31   int* tablocal = new int[nlocal];
32   int* tabres = new int[nlocal];
33   srand(time(NULL)+pid);
34   for (int i=0; i<nlocal; i++)
35     tablocal[i] = rand()%10;
36   CommAComprendre(nlocal,tablocal,tabres);
37   delete[] tablocal;
38   delete[] tabres;
39   MPI_Finalize() ;
40   return 0 ; }
```

2. Fonction Mystère

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

```
2       int pid, nprocs;
3       MPI_Comm_rank(MPI_COMM_WORLD, &pid);
4       MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
5       MPI_Status status;
6
7       int nb_etape = ceil((log((double) nprocs) / log(2.0)));
8       for (int i=0; i<nb_etape; i++) {
9           int size = pow(2,i);
10          if ((pid-root+nprocs)%nprocs<size) {
11              int pid_dest = (pid+size+nprocs)%nprocs;
12              if ((pid-root+nprocs)%nprocs+size<nprocs)
13                  MPI_Ssend(ptr_a, 1, MPI_INT, pid_dest, 1, MPI_COMM_WORLD);
14              }
15          else if ((pid-root+nprocs)%nprocs<pow(2,i+1)){
16              int pid_src = (pid - size + nprocs)%nprocs;
17              MPI_Recv(ptr_a,1,MPI_INT,pid_src,1,MPI_COMM_WORLD,&status);
18          }
19      }
20  }
21  int main(int argc, char **argv) {
22      int pid, nprocs;
23      MPI_Init(&argc, &argv);
24      MPI_Comm_rank(MPI_COMM_WORLD, &pid);
25      MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
26      int root = atoi(argv[1]); // le processeur root
27      int a;
28      if (pid==root) {
29          srand(time(NULL));
30          a = rand() % 10;
31      }
32      Mystere(&a,root);
33      MPI_Finalize();
34      return 0;
35  }
```

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.

```
1  int fn_max(int n, int* tab) {
2      int max = tab[0];
3      for (int i=1; i<n; i++)
4          if (tab[i]>max)
5              max = tab[i];
6      return max;
7  }
8  int main ( int argc , char **argv )
9  {
10   int pid, nprocs;
11   MPI_Init (&argc , &argv) ;
12   MPI_Comm_rank(MPI_COMM_WORLD, &pid ) ;
13   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  }
```

bonus 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

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

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) ?