TP MPI N°1 : Prise en main

1. **Mise en place de l'environnement MPI** :
   - On travaille toujours sur `turing`.
   - OpenMPI est installé dans le répertoire : `/usr/lib64/openmpi`
     - Vérifier l'accès des commandes MPI. Vous pouvez utiliser la commande `which`. Par exemple : `which mpicc` donne `/usr/lib64/openmpi/bin/mpicc`. Pour connaître les informations sur la version d'OpenMPI installée, utiliser la commande : `ompi_info`.
     - Si `which` ne trouve pas de commandes MPI. Il faudra alors modifier votre `.bashrc` en ajoutant les lignes suivantes :

```
if ! (which mpicc>/dev/null 2>&1) && [ -d /usr/lib64/openmpi ]
then
    export PATH=/usr/lib64/openmpi/bin:$PATH
    export LD_LIBRARY_PATH=/usr/lib64/openmpi/lib:$LD_LIBRARY_PATH
fi
```

     - Vous pouvez aussi modifier `CPATH` ou `C_INCLUDE_PATH` pour l'inclusion de fichiers d'entête.

2. **Programmation : Qui suis je ?**
   - Reprendre le premier exemple de la présentation du MPI (`hello_mpi.c`).

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

int main(int argc, char **argv)
{
      int  rang=-1, nbprocs=0;

      MPI_Init( &argc, &argv );
      MPI_Comm_rank( MPI_COMM_WORLD, &rang );
      MPI_Comm_size( MPI_COMM_WORLD, &nbprocs );

      printf( " Hello from process %d of %d\n ",
            rang, nbprocs);

      MPI_Finalize();
      return EXIT_SUCCESS;
}
```

   - Compiler ce programme avec la commande `mpicc –o hello_mpi hello_mpi.c`. Vous pouvez ajouter les options usuelles de compilation de C.
     - Exécuter le programme avec la commande : $ `mpiexec -np 8 ./hello_mpi`
     - Ajouter la fonction `MPI_Get_processor_name(processor_name, &namelen);` et la fonction `cpu_id=sched_getcpu();(<sched.h>)` qui vous permet de connaître le nom du nœud et le numéro de core sur lequel s'exécute un processus.

3. Programmation : Communication point-à-point

- Reprendre le deuxième exemple du cours, qui porte le nom « `p2p.c` ».

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

int main(int argc, char **argv)
{
      int        rang, nbprocs, dest=0, source, etiquette = 50;
      MPI_Status statut;
      char       message[100];

      MPI_Init( &argc, &argv );
      MPI_Comm_rank( MPI_COMM_WORLD, &rang );
      MPI_Comm_size( MPI_COMM_WORLD, &nbprocs );

      if ( rang != 0 ) {
         sprintf(  message, "Bonjour de la part de P%d!\n" , rang
         );
         MPI_Send( message, strlen(message)+1, MPI_CHAR,
                   dest, etiquette, MPI_COMM_WORLD );
      }
      else
         for ( source=1; source<nbprocs; source++ ) {
             MPI_Recv( message, 100, MPI_CHAR, source,
                       etiquette, MPI_COMM_WORLD, &statut );
             printf( "%s", message );
         }

      MPI_Finalize();
      return EXIT_SUCCESS ;
}
```

- Compiler le programme, puis l'exécuter.

- Remplacer le paramètre `source` de la fonction `MP_Recv` par `MPI_ANY_SOURCE`, exécuter plusieurs fois le programme et analyser les résultats d'affichage.

4. Modification du deuxième exemple « p2p.c »

Soit N le nombre de processus d'une exécution,

- On demande de les organiser en anneau comme dans la Figure 1.

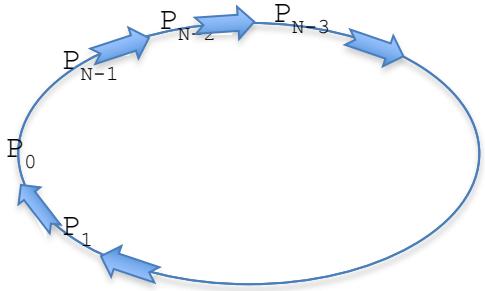

Figure 1. Organisation en anneau des processus

Le message de P_{N-1} est envoyé à P_{N-2}, concaténé au message de P_{N-2}, puis P_{N-2} envois le message à P_{N-3}, ainsi de suite jusqu'à P_0. P_0 reçoit le message de P_1 et l'afficher dans le terminal. Le message affiché prendra la forme suivante :

`Bonjour de la part de` P_1, P_2, P_3, … P_{N-1}!

- Refaire ces communications avec un arbre binaire avec $N=2^n$, comme montre la figure 2. (optionnel)

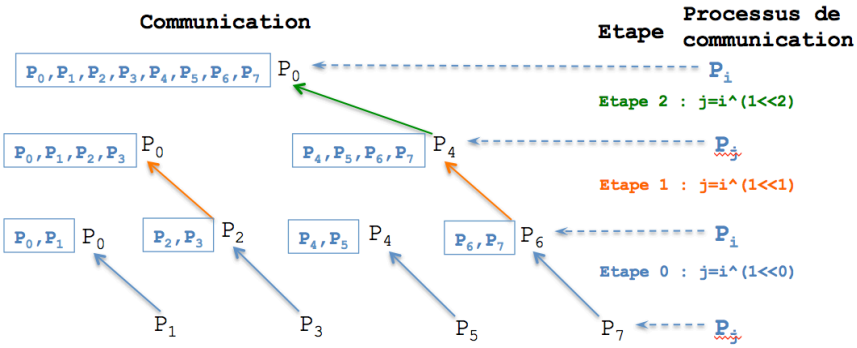

Figure 2. Organisation en arbre binaire des processus