

TP MPI N°1 : Prise en main

1. Mise en place de l'environnement MPI :

- Se connecter à etud, puis connecter vous sur frontalhpc via ssh.
- OpenMPI est installé dans le répertoire : /usr/lib64/openmpi
 - o 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.
 - o Si which ne trouve pas de commandes MPI. Il faudra 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:$PATH
    export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib64/openmpi/lib
fi
```
 - o CPATH ou C_INCLUDE_PATH pour les 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 hello.c. Vous pouvez ajouter les options usuelles de compilation de C.
 - o Exécuter le programme avec la commande: \$ sbatch submit_hell_mpi.sh

```
#!/bin/bash

# submit_hello_mpi.sh fichier pour l'exécution de hello_mpi
# Options de sbatch
#SBATCH --partition=part-etud # partition pour tps etudiants
#SBATCH --ntasks=8           # 8 tasks / processus
#SBATCH --cpus-per-task=1     # de 1 thread
#SBATCH job-name=testMPI

# Execution du programme
./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.
- Exécuter le programme sur plusieurs nœuds en ajoutant l'option suivante dans **submit_hello_mpi.sh**
`#SBATCH --nodelist=node27,node29`

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 `MPI_Recv` par `MPI_ANY_SOURCE`, exécuter plusieurs fois le programme et analyser les résultats d'affichage.

Soit N le nombre de processus d'une exécution,

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



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

