TP MPI N°1 : Prise en main

1. Matériels disponibles

- Cluster `master0` : CentOS 6.7, Quad-Core AMD Opteron 2352 (8 cores), **OpenMPI 1.8.1**
  Liste des nœuds disponibles : `node0, node1, node2, node3, node5, node10, node11, node12, node13, node15, node17, node20, node21`
- Serveur `etud` : CentOS 6.8, AMD Opteron 6272 (64 cores), **OpenMPI 1.8.1**
  Attention : deux implémentations de MPI (OpenMPI et MPICH) sont installées sur ce serveur. Assurez vous que vous utilisez l'implémentation OpenMPI !

2. Mise en place de l'environnement MPI :

- Se connecter à `etud`, puis connecter vous sur `master0` via `ssh`.
- 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/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 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
```

- MPI utilise `ssh` pour l'exécution des processus distants. Afin d'éviter d'enter le mot de passe à chaque exécution pour chaque nœud concerné, On va utiliser `ssh` **sans passphrase** :
  - A la racine de votre répertoire privé, créer une paire de clés privée/publique **sans passphrase** à l'aide de :
    `$ ssh-keygen -t rsa`
  - Copier/ajouter la clé publique dans le fichier `~/.ssh/authorized_keys` :
    `$ cat id_rsa.pub >> authorized_keys`
  - Les clés est créées et prêtes à être utilisées.
  - Fermer l'accès de votre répertoire privé par les autres pour plus de sécurité
    `$ chmod 700 /home/etud/$USER`
- A la première connexion à un nœud, répondre `'yes'` au processus d'authentification afin qu'il ajoute le serveur courant dans la liste des `known_hosts` du nœud.

3. Programmation : Qui suis je ?

- Reprendre le premier exemple de la présentation du MPI (`hello.c`).

```
#include "mpi.h"
#include <stdio.h>
int main(int argc, char **argv)
{
      int rang, nbProcs;
      MPI_Init( &argc, &argv );
      MPI_Comm_rank( MPI_COMM_WORLD, &rang );
      MPI_Comm_size( MPI_COMM_WORLD, &nbProcs );
      printf( " Hello from proc. %d of %d\n ",
            rang, nbProcs);
      MPI_Finalize();
      return 0;
}
```

- Compiler ce programme avec la commande `mpicc -o hello hello.c`. Vous pouvez ajouter les options usuelles de compilation de C.
    - Exécuter le programme sur le nœud courant : `mpirun -np 4 hello`
    - Ajouter la fonction `MPI_Get_processor_name` qui vous permet de connaître le nom du nœud sur lequel s'exécute un processus.
    - Exécuter le programme sur plusieurs nœuds : `mpirun -np 16 -host node0,node1 hello`
      Attention : pas d'espace dans la liste des nœuds !

4. Programmation : Communication point-à-point

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

```
#include "mpi.h"
#include <stdio.h>
#include <string.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 0 ;
}
```

- 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.

5. Modification du deuxième exemple

Soit `N` le nombre de processus d'une exécution,

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

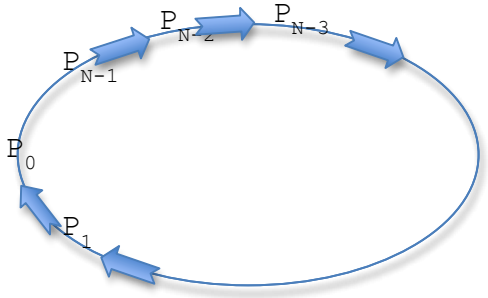

Figure 1. Organisation en anneau des processus

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 prend la forme suivante :

`Bonjour de la part de P1, P2, P3, … PN-1!`

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

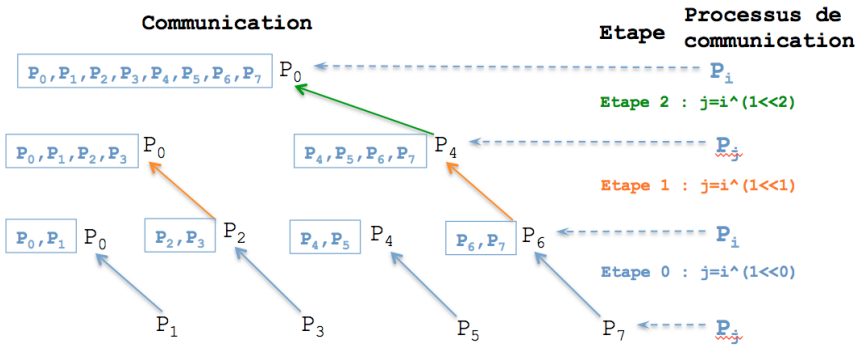

Figure 2. Organisation en arbre binaire des processus