TME 1 - Initiation à MPI avec OpenMPI

Contexte

- ✯ MPI (The Message Passing Interface), est une norme définissant une bibliothèque de fonctions, utilisable avec les langages C, C++ et Fortran. Elle permet d'exploiter des ordinateurs distants ou multiprocesseur par passage de messages.
- ✯ Cette interface considère un environnement totalement distribué où les processus ne partagent pas de mémoire.
- ✯ Elle est très utilisée dans les calculs haute performance utilisant plusieurs machines et dans l'expérimentation des algorithmes répartis.

Pré-requis : Langage C et pthread de POSIX

Exercice(s)

Exercice 1 – Configuration

Question 1

Commencez par vérifier la distribution de MPI qui est installée sur votre machine :

```
bash-3.2$ which mpicc
/usr/local/bin/mpicc
```

Question 2

La construction de l'exécutable nécessite une édition dynamique de liens avec les bibliothèques de `openmpi`. Il faut donc préciser l'emplacement de ces bibliothèques. Ceci peut être fait en ajoutant dans votre fichier `~/.bashrc` les lignes :

```
LD_LIBRARY_PATH=/usr/local/lib
export LD_LIBRARY_PATH
```

Question 3

`openmpi` utilise `ssh` pour que les différentes machines de l'application puissent converser. Pour éviter de taper votre mot de passe à chaque fois que vous lancez un programme MPI, créez , si ce n'est déjà fait, un couple de clés ssh grâce à la commande ssh-keygen et ajoutez-la aux clés autorisées.

```
ssh-keygen -q -N '' -f ${HOME}/.ssh/id_rsa
cat ${HOME}/.ssh/id_rsa.pub >> ${HOME}/.ssh/authorized_keys
```

Question 4
Vérifiez en tapant la commande :

```
ssh localhost
```

Question 5
Pour compiler un programme mpi vous devez :
- vous assurer que le fichier source inclut le fichier mpi.h (directive **`#include`** `<mpi.h>`)
- utiliser la commande *mpicc* au lieu de *gcc*

Pour exécuter un programme mpi vous devez utiliser la commande *mpirun* au lieu de lancer directement votre exécutable. La commande

```
mpirun -np 5 mon_programme
```

permet de lancer le programme mpi *mon_programme* avec 5 processus sur la machine locale (localhost). L'option *-np* permet de paramétrer le nombre de processus voulus.

Pour prendre en compte différentes machines il faut lister le nom DNS ou l'adresse IP de ces machines dans un fichier. L'option à ajouter est *–hostfile*. Ainsi la commande :

```
mpirun -np 5 --hostfile my_hostfile mon_programme
```

permet de lancer le programme mpi *mon_programme* avec 5 processus, répartis sur l'ensemble des machines listées dans le fichier *my_hostfile*.

Pour commencer nous utiliserons uniquement la machine locale.

Cette configuration est valable pour tous les TME MPI que vous ferez par la suite. Il n'est pas nécessaire de refaire ceci à chaque fois.

Exercice 2 – Hello World

Cet exercice a pour but de coder votre premier programme MPI et de s'assurer que votre machine est correctement configurée.

Question 1
Programmez et exécutez un premier programme MPI où chaque processus affiche :

```
"Processus <rank_processus> sur <nb_process_total> : Hello MPI"
```

Prenez un nombre de processus égal à 5.

Exercice 3 – Hello Master

Dans cet exercice, nous allons manipuler des primitives de communication *MPI_Send* et *MPI_Recv*.

Question 1
Que fait le programme suivant ?

```c
#include <stdio.h>
#include <string.h>
#include <mpi.h>

#define MASTER 0
#define SIZE 128

int main(int argc, char **argv){
  int my_rank;
  int nb_proc;
  int source;
  int dest;
  int tag =0;
  char message[SIZE];

  MPI_Status status;

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &nb_proc);
  MPI_Comm_rank(MPI_COMM_WORLD,&my_rank);

  if(my_rank !=MASTER){
    sprintf(message, "Hello Master from %d", my_rank);
    dest = MASTER;
    MPI_Send(message, strlen(message)+1, MPI_CHAR,dest,tag,MPI_COMM_WORLD);

  }else{
    for(source=0;source < nb_proc;source++){
      if(source != my_rank){
       MPI_Recv(message, SIZE, MPI_CHAR, source, tag, MPI_COMM_WORLD, &status);
       printf("%s\n", message);
      }
    }
  }

  MPI_Finalize();

  return 0;
}
```

Question 2
Programmez, compilez et exécutez ce programme en faisant varier le nombre de processus. Que remarquez-vous au niveau des affichages ?

Question 3
Remplacez la variable $source$ dans le *MPI_Recv* par l'identificateur `MPI_ANY_SOURCE`. Faites le test plusieurs fois de suite. Que se passe-t-il ? Expliquez.

Exercice 4 – Hello Neighbor

Question 1
Ecrivez un programme tel que chaque processus envoie une chaîne de caractères à son successeur (le processus $(rang + 1)\ mod\ nb\_proc$), puis reçoit un message de son prédécesseur et l'affiche ensuite.

Question 2
Une fois que votre programme fonctionne, remplacez *MPI_Send* par la primitive d'envoi synchrone *MPI_Ssend*. Que se passe-t-il ?

Question 3
Tout en gardant un thread par processus, proposez une solution pour résoudre le problème souligné dans la question précédente. Programmez-la et vérifiez que le problème est résolu.

Exercice 5 – L'algorithme de Chang & Roberts (1979)
L'algorithme de Chang & Roberts réalise une élection sur un anneau unidirectionnel où les communications sont asynchrones. Le principe est le suivant :

- l'élection peut être initiée par plusieurs processus ;
- un processus initiateur envoie un jeton portant son identité à son successeur ;
- lorsqu'un processus reçoit un jeton :
  — s'il n'est pas encore initiateur, il ne peut plus le devenir : il passe dans l'état battu et transmet le jeton ;
  — s'il est initiateur, il ne transmet le jeton que si celui-ci porte une identité supérieure à la sienne.

Question 1
Quel est le processus qui gagne l'élection ? Comment le sait-il ?

On veut implémenter une version de cet algorithme qui inclut une annonce de résultat : lorsque l'algorithme se termine, tous les processus connaissent l'identité du processus élu. D'autre part, pour conserver un certain indéterminisme, chaque processus décide aléatoirement s'il est initiateur ou non en exécutant le code suivant :

```
srand(getpid());
initiateur = rand()%2;
```

Ce choix laisse un risque (faible) de n'avoir aucun processus initiateur : dans ce cas, on interrompt l'exécution et on recommence...

Question 2
En vous inspirant de votre réponse à la première question de l'exercice précédent pour simpuler un réseau en anneau, implémentez l'algorithme sous MPI, en affichant

- l'identité des processus initiateurs,
- les opérations de destruction de jeton,
- le résultat de l'élection sur tous les sites.

Question 3
Exécutez plusieurs fois le programme pour vérifier que les résultats sont corrects quand le nombre et l'identité des initiateurs varient.