TP ALGOREP

E. Renault

Conseil : Beaucoup d'algorithmes vous sont demandés dans ce TP, il est recommandé de faire un algorithme par fichier.

1 Préliminaires

MPI est une norme *de facto* pour la programmation parallèle le par passage de messages qui a pour objectif d'être portable et d'obtenir de bonnes performances. Plusieurs implémentations de MPI existent, nous utiliserons ici open-mpi [1]. Pour installer MPI utilisez votre gestionnaire de packets préféré ou bien installez le en local, en suivant les instructions suivantes :

1. `https://www.open-mpi.org/nightly/v4.1.x/`
2. ./configure –prefix=somepath
3. make
4. make install

Vous devez maintenant obtenir deux exécutables dans somepath/bin/

- **mpicc** : le compilateur pour MPI
- **mpirun** : le lanceur d'exécutable MPI

En considérant le fichier C suivant vous pouvez maintenant l'exécuter en faisant :

- mpicc hello.c
- mpirun ./a.out

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

int main(int argc, char* argv[])
{
    int rank, size, len;
    char version[MPI_MAX_LIBRARY_VERSION_STRING];

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Get_library_version(version, &len);
    printf("Hello, world, I am %d of %d, (%s, %d)\n", rank, size, version, len);
    MPI_Finalize();

    return 0;
}
```

Par defaut mpi utilise le nombre de coeurs disponibles sur la machine. Si nous voulons utiliser plus de coeurs (en local ou en distribué) il préciser à mpirun l'option *-np nbprocesses* et un fichier hostfile via l'option *-hostfile hostfile*. Ce fichier permet de déclarer des capacité (virtuelles ou non) accessibles en local ou en SSH. Par exemple le fichier hostfile suivant, déclare que localement, la machine peut faire tourner 25 processus.

```
localhost slots=25
```

Notez que par défaut, MPI ne construit que les bindings pour le C. Si vous souhaitez utiliser le C++ ou le python il faut le préciser à configure.

1. `https://www.open-mpi.org`

2 Ping-Pong 2 Processus

Dans cette section nous souhaitons en sorte que deux processus se répondent l'un affichant ping l'autre affichant pong. Avant toute chose, pensez à regarder l'excellente documentation présente ici : `https://www.open-mpi.org/doc/current/` et notamment MPI_Init, MPI_Finalize, MPI_Comm_rank, MPI_Comm_size, MPI_Send, MPI_Ssend et MPI_Recv. Pour réaliser ce ping-pong, commencez par désigner un processus qui commencera (celui de rang 0 par exemple), et pensez à la terminaison de votre algorithme.

3 Topologies

Nous souhaitons maintenant simuler différentes topologies. Pour cela, nous supposerons que le processus de rang 0 initie toujours le calcul. Vos algorithmes doivent pouvoir fonctionner et ce, quelque soit le nombre de processus spécifiés

- Écrivez un algorithme permettant de simuler l'envoi d'un message autour d'un anneau virtuel. A chaque réception, le processus affichera son identifiant. Indice, utilisez les identifiants pour simuler un anneau.
- Simulez une topologie centralisée dans laquelle un message est envoyé à un noeud, puis remonte vers le maÎtre qui le redispatche ensuite vers un autre noeud. Quand tous les noeuds ont reçus un message l'algorithme s'arrête. Pensez a afficher un message à chaque réception/envoi de message.
- Simulez une topologie hiérarchique dans laquelle la racine initie un message qui va faire un parcours en profondeur de "l'arbre" représenté par la topologie. Pensez a afficher un message à chaque réception/envoi de message.

4 Horloges

Reprenez l'exercice précédent en y ajoutant dans un premier temps les horloges de Lamport, puis celles de Mattern. Vous les afficherez a chaque réception de message.

5 Ping-Pong multi processus

Nous reprenons maintenant le premier exercice et nous le généralisons. Pensez à regarder les émissions/réceptions non-bloquantes MPI_Isend, MPI_Irecv.

- Chaque processus tire un nombre aléatoire. Nous appelons ce nombre *nbcoups*.
- Maintenant, chaque processus va communiquer ce nombre avec tous les autres processus.
- Les processus (deux à deux) se mettent d'accord sur le nombre de coups devant être joués.
- Toutes parties sont ensuite démarrées et lorsque la dernière s'arrête le programme s'arrête.

Pour aller plus loin :

- Assurez vous que toutes les parties ne commencent que lorsque tous les processus sont arrivés à un consensus deux à deux.
- Un processus ne peut maintenant jouer qu'une partie à la fois. Modifiez votre code en conséquence.
- Organisez maintenant vos processus sur une topologie hiérarchique et simulez un tournois (le dernier qui tape la balle gagne). Pensez que les processus ne peuvent communiquer qu'avec leurs parents et/ou enfants.