Travaux pratiques : introduction à MPI

Frédéric Guinand, Moustafa Nakechbandi, IUT du Havre - Université du Havre
email:Frederic.Guinand@univ-lehavre.fr, moustafa.nakechbandi@univ-lehavre.f
2007/2008

1 Introduction

Dans les séances de TP que nous allons voir, le langage choisi est C, utilisé, conjointement aux primitives offertes par la biblioth`eque de communication par échange de message MPI.

Condition pour exécuter un code MPI

Exécuter un programme MPI nécessite l'inclusion des fichiers d'entête dans lesquels se trouvent les définitions des différentes primitives de MPI.

2 Sujet

En vous aidant des informations données en annexe, notamment de la présentation de MPI, vous allez programmer une application de type maître/esclave.
Le rôle du maître est de générer et fournir des tableaux d'entiers aux esclaves qui effectuent l'addition des éléments du tableau et renvoient la somme obtenue au maître.

3 Commandes de base

Il existe 6 commandes de base :
– MPI _Init qui initialise un bloc de calcul utilisant MPI.
– MPI_Finalize qui termine un bloc de calcul MPI.
– MPI_COMM size qui détermine le nombre de processus qui seront engagé dans un calcul.
– MPI_COMM rank qui détermine l'identificateur d'un processus.
– MPI_Send primitive d'envoi de messages.
– MPI_Recv primitive de reception de messages.

3.1 Exemple de base

3.1.1 Edition

L'exemple de base qui ne fait rien consiste simplement à initialiser une phase de calcul MPI et à la terminer :

```
#include <stdio.h>
#include <mpi.h>
int main(int argc,char **argv) { MPI Init(&argc,&argv)
printf("bonjour la compagnie ! !") ;
MPI Finalize()
}
```

3.1.2 Compilation

```
mpicc basics1.c –o b1
```

3.1.3 Exécution

Avant de pouvoir exécuter un programme MPI, il faut lancer le moteur d'exécution de MPI. Lorsque l'outil LAM est installé sur les machines, cela se fait avec la commande `lamboot`.

L'exécution se fait avec la commande `mpirun`. Cette commande prend deux arguments, le nombre de processus qui devront exécuter le code et le nom du programme. Dans notre cas, si on désire que le programme b1 soit exécuté par 4 processus :

```
mpirun –np 4 b1
Bonjour la compagnie !!!
Bonjour la compagnie !!!
Bonjour la compagnie !!!
Bonjour la compagnie !!!
```

3.2 Connaître son environnement

Pour que chaque processus prenne connaissance de son environnement d'exécution, il faut qu'il obtienne son propre identifiant et le nombre de processus qui exécutent lemême programme que lui.
Les primitives MPI_COMM size() et MPI_COMM rank() sont faites pour répondre à ces questions.

Ci-dessous un nouveau programme qui permet à chaque processus d'indiquer qui il est dans le groupe de processus
qui ont été générés pour la réalisation de la tˆache courante.

```
#include <stdio.h>
#include <mpi.h>
int main(int argc,char **argv) { int moi,lesautres ;
MPI_Init(&argc,&argv)
MPI_COMM rank(MPI_COMM WORLD,&moi) ;
MPI_COMM size(MPI_COMM WORLD,&lesautres) ;
printf("bonjour la compagnie, je suis %d et il y a %d processus impliqués !
!",moi,lesautres) ;
MPI Finalize()
}
```

Compilation puis exécution :

```
> mpicc basics2.c –o b2
> mpirun –np 4 b2
bonjour la compagnie, je suis 0 et il y a 4 processus impliqués !!
bonjour la compagnie, je suis 1 et il y a 4 processus impliqués !!
bonjour la compagnie, je suis 2 et il y a 4 processus impliqués !!
bonjour la compagnie, je suis 3 et il y a 4 processus impliqués !!
```

3.3 Communiquer en mode point-à-point

Supposons qu'un processus p1 veuille communiquer un résultat à un processus p2, il va utiliser un mode de communication de type point-à-point, c'est-à-dire un envoi de message simple. Les primitives utilisées pour ce type de communication sont :

- **MPI_Send**(void *,int, MPI Datatype, int, int, MPI Comm) :
  1. adresse du buffer d'émission,
  2. nombre d'éléments,
  3. type des éléments,
  4. destinataire (il s'agit de son rang),
  5. tag (permet de différencier les messages entre eux),
  6. communicateur.
- **MPI_Recv**(void *, int, MPI Datatype, int, int, MPI Comm, MPI Status*) :
  1. adresse du buffer de reception,
  2. nombre d'éléments,
  3. type des éléments,
  4. source, rang du processus dont provient le message,
  5. tag, permet de différencier les messages à la reception,
  6. status : état de la communication. Peut être ignoré avec MPI STATUS IGNORE

Exemple :

```
#include<stdio.h>
#include "mpi.h"
#define N 10
/*
* exemple de communication en mode point-à-point.
* L'un des processus envoi un message à l'ensemble
* des autres, à tour de rôle.
*/
void generationAleatoire(int *,int, int, int);
void afficher(int *,int);
int additionner(int *,int);
int main(int argc, char** argv) {
int Tablo[N];
int Tab2[N];
```

```
int moi,nbprocs,lesautres;
int tag = 4444;
MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD,&moi);
MPI_Comm_size(MPI_COMM_WORLD,&nbprocs);
if(moi == 0) {
      generationAleatoire(Tablo,N,100,1000);
      afficher(Tablo,N);
      printf("%d : je vais envoyer \n",moi);
      for(lesautres = 1;lesautres<nbprocs;lesautres++) {
            MPI_Send(Tablo, // buffer d'envoi
            N, // nombre d'éléments
            MPI_INT, // type des éléments
            lesautres, // rang du processus d'envoi
            4444, // tag
            MPI_COMM_WORLD); // communicateur
            printf("%d : envoi à %d \n",moi,lesautres);
      }
} else {
      printf("%d : je vais recevoir \n",moi);
      MPI_Recv(Tab2, // buffer de reception
      N, // nombre d'éléments
      MPI_INT, // type des éléments
      0, // rang de l'émetteur
      4444, // tag
      MPI_COMM_WORLD, // communicator
      MPI_STATUS_IGNORE); // status
      printf("%d : somme %d \n",moi,additionner(Tab2,N));
}
MPI_Finalize();
}

/*
* générateur aléatoire de nombres entiers compris entre min et max
*/
void generationAleatoire(int *T, int taille, int min, int max) {
      int i;
      srand(time(NULL));
      for(i=0;i<taille;i++) {  T[i] = min+rand()%(max-min);  }
}

/*
* procédure qui affiche le contenu du tableau
*/
void afficher(int *T, int taille) {
      int i;
      for(i=0;i<taille;i++) { printf("%d : %d \n",i,T[i]); }
}

/*
* fonction qui effectue l'addition des éléments du tableau d'entiers.
*/
int additionner(int *T, int taille) {
      int i;
      int somme = 0;
      for(i=0;i<taille;i++) { somme+=T[i]; }
      return somme;
}
```

Exercice
Modifier le dernier exemple pour faire un programme permettant de calculer la somme de 3N éléments :

- Le maître envoie aux esclaves 3 tableaux.
- Les esclaves calculent puis envoient au maître les sommes partielles
- Le maître reçoit les sommes partielles de tous les esclaves et effectue la somme globale puis affiche le résultat.

Ce TP est noté, le programme de l'exercice précédent (la somme globale) doit être envoyer à la fin de ce TP à l'adresse suivante **:** nakech@free.fr avec comme objet : `cr_tp2_gb_votre_nom`.