05

MPI

Inria

C'est quoi ?

C'est quoi ?

Interface de programmation pour architecture à mémoire partagée ou distribuée

C'est quoi ?

Interface de programmation pour architecture à mémoire partagée ou distribuée

Principe

C'est quoi ?

Interface de programmation pour architecture à mémoire partagée ou distribuée

Principe

On lance plusieurs instances du même programme (processus)

C'est quoi ?

Interface de programmation pour architecture à mémoire partagée ou distribuée

Principe

On lance plusieurs instances du même programme (processus)

Ces processus sont indépendants les uns des autres.

C'est quoi ?

Interface de programmation pour architecture à mémoire partagée ou distribuée

Principe

On lance plusieurs instances du même programme (processus)

Ces processus sont indépendants les uns des autres.

Pour communiquer, ils doivent s'envoyer des messages (MPI=Message Passing Interface)

C'est quoi ?

Interface de programmation pour architecture à mémoire partagée ou distribuée

Principe

On lance plusieurs instances du même programme (processus)

Ces processus sont indépendants les uns des autres.

Pour communiquer, ils doivent s'envoyer des messages (MPI=Message Passing Interface)

Statut des variables

C'est quoi ?

Interface de programmation pour architecture à mémoire partagée ou distribuée

Principe

On lance plusieurs instances du même programme (processus)

Ces processus sont indépendants les uns des autres.

Pour communiquer, ils doivent s'envoyer des messages (MPI=Message Passing Interface)

Statut des variables

Les processus étant indépendants, toutes les variables sont privées

C'est quoi ?

Interface de programmation pour architecture à mémoire partagée ou distribuée

Principe

On lance plusieurs instances du même programme (processus)

Ces processus sont indépendants les uns des autres.

Pour communiquer, ils doivent s'envoyer des messages (MPI=Message Passing Interface)

Statut des variables

Les processus étant indépendants, toutes les variables sont privées

Pour qu'un processus A connaisse la valeur d'une variable d'un processus B, il faut que B lui envoie un message

Commande mpiexec ou mpirun :

```
mpiexec -n 4 echo 'hello'
```

Sortie

```
hello
hello
hello
hello
```

Code (hello.F90)

```
program main
  use mpi_f08
  implicit none
  integer :: rang,a
  integer :: nb_procs
call MPI_INIT()
call MPI_COMM_SIZE(MPI_COMM_WORLD,nb_procs)
call MPI_COMM_RANK(MPI_COMM_WORLD,rang)
print*, "hello, mon rang est ", rang
call MPI_FINALIZE()
end program main
```

Compilation

```
mpif90 -o exec.out hello.F90
```

Execution

```
mpirun -n 4 exec.out
```

Sortie

```
 hello, mon rang  est          1
 hello, mon rang  est          0
 hello, mon rang  est          2
 hello, mon rang  est          3
```

Pour envoyer un message du processus I au processus J

```
call MPI_SEND(envoi,taille,MPI_TYPE,J,tag,MPICW)
```

Pour recevoir un message du processus I sur le processus J

```
call MPI_RECV(reception,taille,MPI_TYPE,I,tag,MPICW,statut)
```

Description des commandes

MPI_TYPE indique le type de données des vecteurs envoi et réception (entier, réel, réel double, etc...)

taille indique la taille des vecteur envoi et réception

Sauf cas très particulier, taille et MPI_TYPE doivent être les mêmes dans les deux appels.

TAG est un entier qui permet de contrôler la communication. Il doit être le même dans les deux appels.

MPICW (MPI_COMM_WORLD) et statut sont deux objets utilisés pour les communications, on ne les décrira pas ici.

Communications bloquantes

Une communication est bloquante quand l'exécution du processus est bloquée tant que la communication n'est pas terminée.

MPI_SEND/MPI_RECV sont utilisées pour des communications bloquantes.

Communications non bloquantes

Une communication est non-bloquante quand les processus peuvent effectuer d'autres opérations en attendant la fin de la communication.

On utilise MPI_ISEND et MPI_IRECV pour des communications non bloquantes

Communications bloquantes avec envoi synchrone : MPI_SSEND

Le communication est synchrone quand le processus qui envoie attend que le processus qui reçoit soit prêt pour effectuer la communication.

MPI_SSEND est utilisé quand les données à envoyer sont de très grande taille.

Inconvénient : risque fort d'interblocage

Communications bloquantes avec envoi bufferisé : MPI_BSEND

Un envoi est bufférisé quand le processus qui envoie place la donnée dans un buffer en attendant que le processus qui reçoit soit prêt pour. Le processus qui envoie peut effectuer d'autres opérations en attendant mais pas le processeur qui reçoit.

MPI_BSEND est utilisé pour limiter les risques d'interblocage

Inconvénient : risque de saturation mémoire pour des données

Communications bloquantes avec envoi standard : MPI_SEND

Avec MPI_SEND, le programme "choisit" d'utiliser MPI_SSEND ou MPI_BSEND en fonction des données à communiquer

Inconvénient : on ne maîtrise pas ces choix

Réception

Dans tous les cas, on utilise la réception bloquante MPI_SEND.

Intérêt

Les communications non bloquantes permettent de recouvrir les temps de communication par des calculs. On peut alors mieux équilibrer les charges de calcul entre les processus.

Inconvénient

Difficulté de mise en oeuvre.

Dans la suite du cours, on se concentrera sur les communications bloquantes.

Problème

Le processus 0 arrive sur un MPI_SEND, il veut faire un envoi synchrome avec le processus 1. Il attend donc que le processus 1 arrive sur le MPI_RECV.

Le processus 1 arrive sur un MPI_SEND, il veut faire un envoi synchrome avec le processus 0. Il attend donc que le processus 0 arrive sur le MPI_RECV.

Les deux processus sont donc bloqués au niveau du MPI_SSEND

Solution

Remplacer les MPI_SSEND par des MPI_BSEND

Problème

Le processus 0 envoie dans un buffer un message pour le processus 1, il attend un message du processus 2

Le processus 1 envoie dans un buffer un message pour le processus 2, il attend un message du processus 0

Le processus 2 envoie dans un buffer un message pour le processus 0, il attend un message du processus 1

Les trois processus sont donc bloqués au niveau du MPI_RECV

Solution

Faire d'abord tous les envois puis toutes les réceptions (interblocage2_solution.F90)

MPI_BCAST

Envoi d'un valeur d'un processus proc à tous les autres

```
call MPI_BCAST(envoi,taille,MPI_INTEGER,proc,MPICW)
```

MPI_BARRIER

Barrière pour synchroniser tous les processus. Aucun processus ne peut passer tant que tous ne sont pas arriver

```
call MPI_BARRIER(MPI_COMM_WORLD)
```

MPI_REDUCE

Opération de réduction sur tous les processus vers un processus.

```
call MPI_REDUCE(valeur,somme,1,MPI_INTEGER,MPI_SUM,0,MPICW)
```

Ajoute les "valeurs" de chaque processus dans la variable somme sur le processus 0.

MPI_ALLREDUCE

Opération de réduction sur tous les processus vers un processus.

```
call MPI_ALLREDUCE(valeur,somme,1,MPI_INTEGER,MPI_SUM,MPICW
```

Ajoute les "valeurs" de chaque processus dans la variable somme sur le processus 0.

Le cours de l'Idris

http://www.idris.fr/media/formations/mpi/idrismpi.pdf