Cours Calcul Intensif - MPI

Hélène Coullon , Sophie Robert, Sébastien Limet

LABORATOIRE D'INFORMATIQUE FONDAMENTALE D'ORLEANS

UNIVERSITE D'ORLEANS

24 octobre 2012

Rappels : Modèle de programmation séquentiel

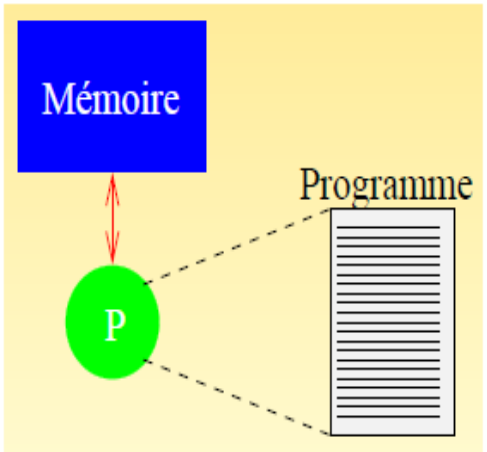
Principes

- Le programme est exécuté par un seul processus
- Toutes les variables et constantes du programme sont allouées dans la mémoire centrale
- Un processus s'exécute sur un processeur physique de la machine

Programme en C

```
#include <stdio.h>
int main(int argc, char** argv)
{
    printf("Bonjour à tous,\n");
    return 0;
}
```

Architectures



Rappels : Modèles de programmation parallèle

?

Rappels : Modèles de programmation parallèle

Programmation pour systèmes à mémoire partagée

- Multi-threads
- OpenMP

Difficultés

- Synchronisations
- Accès concurrents

Rappels : Modèles de programmation parallèle

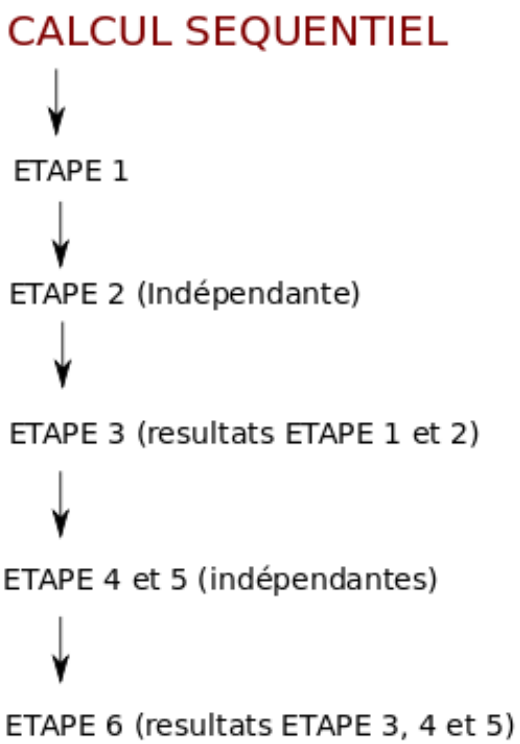
Machines à mémoire distribuée

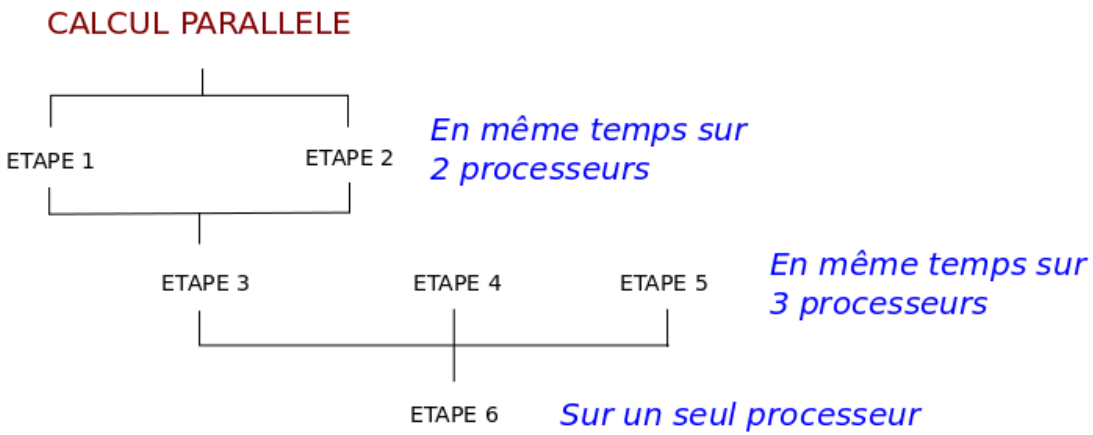
- Chaque processeur possède sa propre mémoire locale
- Le processeur exécute ses instructions sur ses données
- Un processeur ne peut pas avoir accès à la mémoire des autres

Difficultés

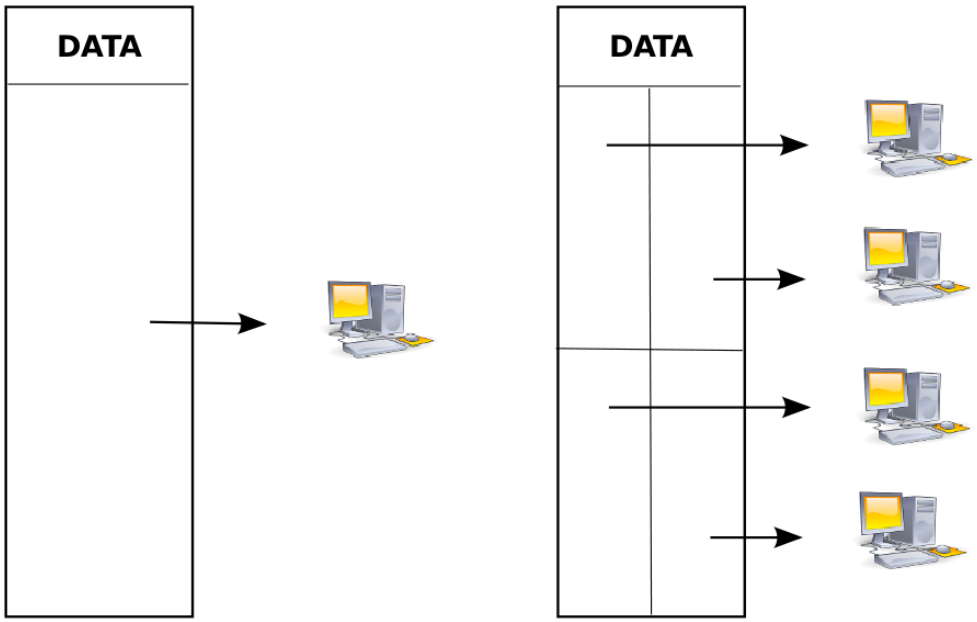
- Pas de problèmes d'accès concurrents
- Problèmes liés aux échanges de messages
- Reflexion différente du séquentiel

Parallélisme de tâches





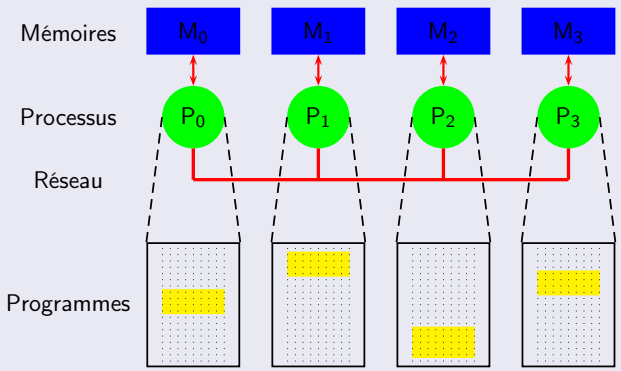
Parallélisme de données



Modèle de programmation par échange de messages

Principes

- Le programme est écrit dans un langage classique (Fortran, C, C++,...)
- Chaque processus exécute éventuellement des parties différentes d'un programme
- Toutes les variables du programme sont privées et résident dans la mémoire locale allouée à chaque processus
- Une donnée est échangée entre deux ou plusieurs processus via un appel à un sous-programme particulier

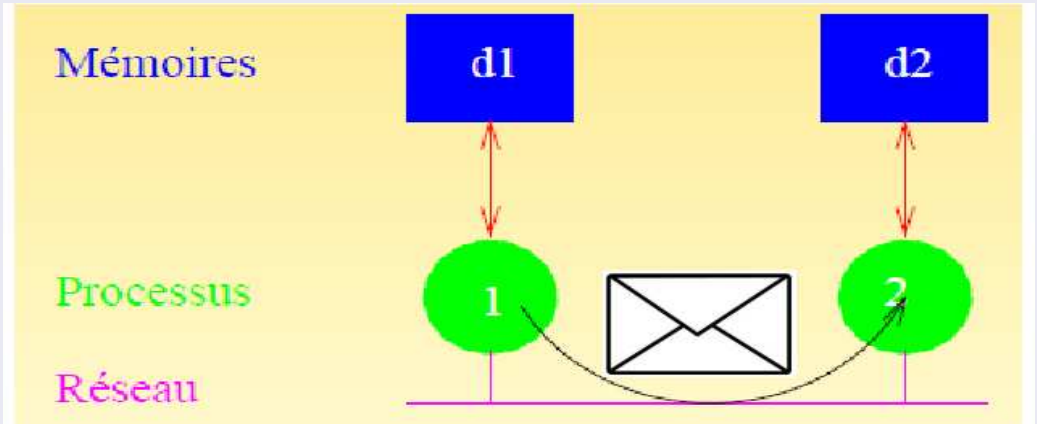


Message à échanger

Un message est constitué de paquets de données.

1. L'entête du message
   - l'identificateur du processus émetteur
   - le type de données (int, float, double, char, struct, ...)
   - la longueur du message
   - l'identificateur du processus récepteur
2. Les données à transmettre

Echange du message

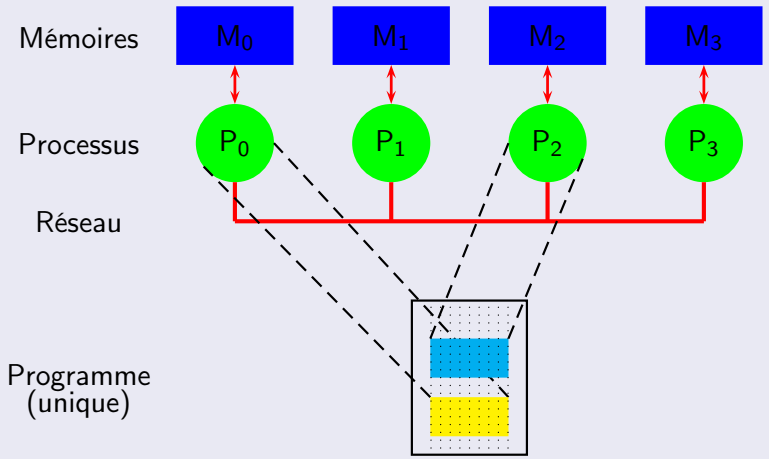


(a) Message (b) Échange d'un message entre deux processus

SPMD - **S**ingle **P**rogram **M**ultiple **D**ata

Principes

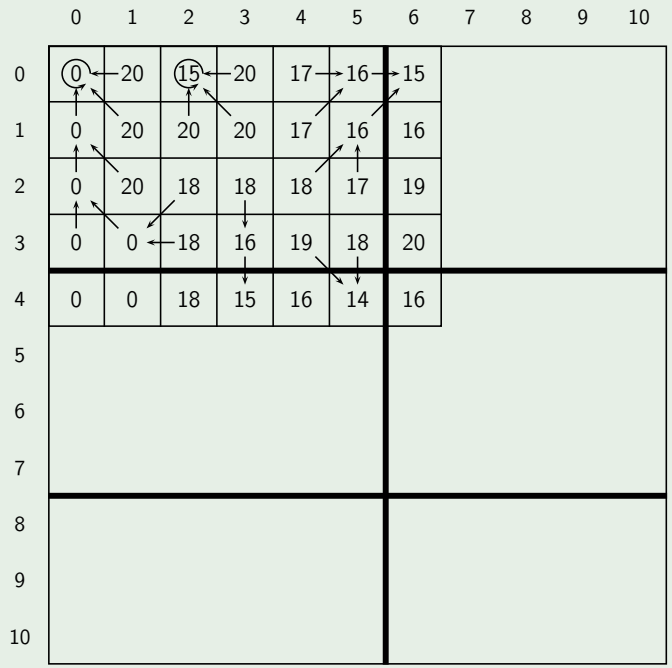
- Le plus grand nombre de processeurs effectuent les même tâches sur des données différentes
- Chaque processeur a ses propres données
- Les processeurs peuvent communiquer par envois de messages
- Cette approche est adaptée au parallélisme de données



Calcul parallèle des directions d'écoulement dans un MNT (D8)

	0	1	2	3	4	5	6	7	8	9	10..
0	0	20	15	20	17	16	15	14	15	16	17
1	0	20	20	20	17	16	16	15	14	15	16
2	0	20	18	18	18	17	19	13	15	14	15
3	0	0	18	16	19	18	20	20	20	20	20
4	0	0	18	15	16	14	16	19	15	17	14
5	0	0	13	16	18	10	18	17	15	18	13
6	0	0	18	18	18	17	18	17	16	18	16
7	0	0	17	17	17	17	17	17	17	17	17
8	0	0	9	8	9	10	11	12	13	14	15
9	0	0	9	8	5	10	11	12	10	14	15
10...	0	0	9	8	9	10	11	12	13	14	15

(c) Modèle Numérique de Terrain



(d) Calcul en local au Proc. 0

MPI - **M**essage **P**assing **I**nterface

MPI, qu'est-ce que c'est ?

- MPI est une spécification et un standard
  - Les implémentations de MPI sont fournies sous forme de bibliothèques libres ou commerciales
  - Les trois principales : MPICH2, OpenMPI, LAM-MPI
- Première version publique : Mai 1994
- Langage supporté : C, Fortran, C++, ...

MPI permet de gérer

- l'environnement d'exécution
- les communications point à point (Send, Recv)
- les communications collectives (Bcast, Reduce, Scatter,...)
- les groupes de processus et les communicateurs
- la topologie d'inter-connexion des processus(grilles, arbres,...)
- des types de données dérivés

Quelques liens

Distributions

- http ://www.mcs.anl.gov/research/projects/**mpich2**/
- http ://www.**open-mpi**.org/
- http ://www.**lam-mpi**.org/

Documents, Forums et tutoriaux

- http ://www.**mpi-forum**.org/
- http ://www.lam-mpi.org/using/**docs**/
- http ://www.lam-mpi.org/**tutorials**/
- http ://www.open-mpi.org/**doc**/
- http ://www-unix.mcs.anl.gov/mpi/**tutorial**

MPI sous language C (1)

Fichier **mpi.h**

Il doit être inclus en entête de tous les programmes MPI.

- Déclaration des prototypes de toutes les routines MPI
- Déclaration de l'ensemble des constantes MPI
- Déclaration de toutes les structures de données

Routines MPI

- Les routines MPI (en C) sont sous deux formes
  1. **MPI_Xxxx()**,
  2. **MPI_Xxxx_xxx()**.
- Il y a six routines MPI de base :

1. MPI_Init,	3. MPI_Comm_size,	5. MPI_Send,
2. MPI_Finalize,	4. MPI_Comm_rank,	6. MPI_Recv.

MPI sous langage C (2)

La structure d'un programme MPI

```
Inclure le fichier mpi.h

Initialiser l'environnement MPI
    ..........................
    ..........................
    Faire des calculs
    Appeler des routines MPI :
        - communiquer
        - se synchroniser
    ..........................
    ..........................
Terminer l'environnement MPI
```

Routine MPI_Init

Prototype

```
int MPI_Init(int* argc, char*** argv);
```

Notes

- C'est la première routine MPI exécutée par tous les processus
- Elle permet de débuter l'exécution parallèle
- Elle permet de diffuser les arguments donnés en ligne de commande
- Elle renvoie MPI_SUCCESS si l'appel n'a eu aucun problème

Routine MPI_Finalize

Prototype

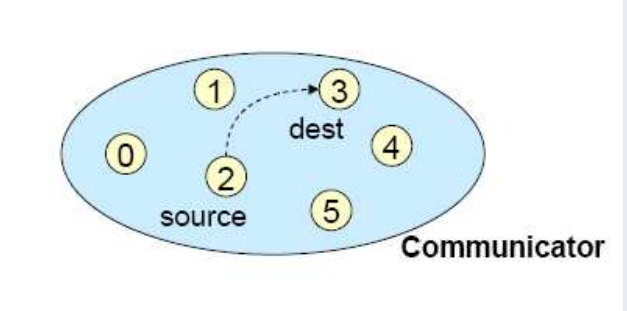
```
int MPI_Finalize(void);
```

Notes

- Elle termine l'exécution parallèle et doit être appelée par tous les processus
- Elle renvoie MPI_SUCCESS si l'appel n'a eu aucun problème
- Ensuite le programme **ne peut plus appeler** les routines MPI, l'éxécution n'est plus parallèle

Domaines de communication

- MPI définie la notion de domaine de communication, qui s'appelle le **communicateur**.



- Représente un ensemble de processus qui sont autorisés à communiquer entre eux
- Le communicateur doit être en argument de toutes les routines de communications
- Par défaut on utilise MPI_COMM_WORLD
- Dans un communicateur, chaque processus est identifié par un nombre entier unique

Routine MPI_Comm_size

Prototype

```
int MPI_Comm_size(MPI_Comm comm, int* nprocs);
```

Notes

- Cette routine retourne dans *nprocs* le nombre total de processus du communicateur *comm*
- MPI_COMM_WORLD est utilisé pour obtenir le nombre total de processus dans le programme parallèle

Routine MPI_Comm_rank

Prototype

```
int MPI_Comm_rank(MPI_Comm comm, int* pid);
```

Notes

- Cette routine retourne dans *pid* l'identifiant relatif au communicateur *comm*, du processus appelant
- Cet identifiant est un nombre entier, et est unique dans le communicateur *comm*
- Initialement, chaque processus a un identifiant unique $pid \in [0, nprocs - 1]$ dans le communicateur MPI_COMM_WORLD.

Programme MPI en C

exemple

```
1  #include <mpi.h>
2  #include <stdio.h>
3  int main(int argc, char** argv)
4  {
5     int pid, nprocs;
6
7     MPI_Init(&argc, &argv);
8     MPI_Comm_rank(MPI_COMM_WORLD, &pid);
9     MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
10    printf("I am %d of %d\n", pid, nprocs);
11    MPI_Finalize();
12
13    return 0;
14 }// end of function main
```

Installation et compilation d'un programme MPI

Installation Ubunbtu - Package

```
$ sudo apt-get install openmpi
                       openmpi-common
                       libopenmpi-dev
```

Commandes

```
mpicc -c exemple.c
mpicc -o exemple exemple.o
```

Exécution sans hostfile

Exécution sans hostfile

```
mpirun -np 4 ./exemple1
```

Résultat

```
I am 3 of 4
I am 0 of 4
I am 2 of 4
I am 1 of 4
```

Exécution avec hostfile

Contenu du fichier *hostfile*

```
machine_1
machine_2
.........
machine_n
```

Exécution

```
mpirun --hostfile hostfile -np 4 ./exemple1
```

Résultat

```
I am 3 of 4
I am 0 of 4
I am 2 of 4
I am 1 of 4
```

Communications entre des processus

Communication point à point

- C'est la communication entre deux processus
- Elle consiste en 2 opérations élémentaires : envoi et réception

Communication collective

- C'est une communication qui implique un ensemble de processus d'un communicateur
- En une seule opération, on effectue une série de communications point à point
- L'ensemble des processus effectuent le même appel avec des arguments correspondants
- Certaines routines ont un processus qui est seul expéditeur ou seul destinataire : il s'appelle le processus **root**

Communications point à point

Principes

- Communication entre 2 processus dans un communicateur
  1. Émetteur :
     - l'identifiant de l'émetteur est implicite
     - Le processus émetteur envoie des messages
  2. Récepteur :
     - le récepteur est donné explicitement dans l'enveloppe
     - Le processus récepteur reçoit des messages
- Modes de communication
  - bloquant
  - non-bloquant

Types de communication

1. synchrone
2. bufferisé
3. standard
4. ready

Fonctions MPI d'envoi et de réception

Types/Modes	Bloquant	Non bloquant
Envoi standard	MPI_Send	MPI_Isend
Envoi synchrone	MPI_Ssend	MPI_Issend
Envoi bufferisé	MPI_Bsend	MPI_Ibsend
Envoi ready	MPI_Rsend	MPI_Irsend
Réception	MPI_Recv	MPI_Irecv

Types de données en MPI

Types de base

Type MPI	Type C
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	octet par octet

Table: Type de base

Principes

Un envoi et une reception bloquante garantissent l'intégrité des données (Envoi et réception des données souhaitées).

- **Appel bloquant** : un appel est bloquant si l'espace mémoire servant à la communication peut être réutilisé immédiatement après la sortie de l'appel.
- **Appel non-bloquant** : un appel non bloquant rend la main très rapidement, mais n'autorise pas la réutilisation immédiate de l'espace mémoire utilisé dans la communication. Il est nécessaire de s'assurer que la communication est terminée avant d'utiliser l'espace mémoire.

Routine de réception bloquante MPI_Recv

Prototype

```
int MPI_Recv(void *buf, int count,
             MPI_Datatype datatype, int source,
             int tag,MPI_Comm comm,MPI_Status *status)
```

Paramètres

1. void ***buff*** : Adresse du tampon de reception
2. int **count** : Nombre d'éléments dans le tampon de données
3. MPI_Datatype **datatype** : Type de chaque élément envoyé
4. int **src** : Identifiant du processus émetteur,
5. int **tag** : Étiquette de message
   - distinguer les messages reçu au niveau récepteur,
   - prendre la valeur MPI_ANY_TAG (le récepteur accepte tous les messages quelque soit l'étiquette).
6. MPI_Comm **comm** : Communicateur,
7. MPI_Status ***status**

Routine de réception bloquante MPI_Recv

MPI_Status

- C'est une structure de donnée pré-définie par MPI,
- Elle nous permet d'obtenir des informations supplémentaires sur le message reçu.
- MPI_STATUS_IGNORE

Structure MPI_Status

```
typedef struct MPI_Status
{
    int MPI_SOURCE;
    int MPI_TAG;
    int MPI_ERROR;
};
```

Envoi bloquant : Synchrone

Envoi avec la routine MPI_Ssend

- L'émetteur se signale d'abord au récepteur
- Il attend la réponse du récepteur
- Le transfert de données est réalisé
- L'appel de cette routine retourne au programme appelant lorsque le récepteur a bien reçu tout le message

Réception avec la routine MPI_Recv

- Le récepteur répond à l'émetteur
- Il provoque le transfert de données
- Il retourne au programme appelant lorsque tout le message a bien été reçu

Routine MPI_Ssend

Prototype

```
int MPI_Ssend(void *buf, int count,
              MPI_Datatype datatype, int dest,
              int tag,MPI_Comm comm);
```

Paramètres

1. void *buf : Adresse initiale du buffer d'envoi
2. int count : Nombre d'éléments envoyés
3. MPI_Datatype datatype : Type de chaque élément envoyé
4. int dest : Identifiant du processus récepteur
5. int tag : Étiquette du message
6. MPI_Comm **comm** : Communicateur

Envoi bloquant : Synchrone

Prototype

```
nt MPI_Ssend(void *buf, int count,
             MPI_Datatype datatype, int dest,
             int tag,MPI_Comm comm);
```

Processus 0 $\longrightarrow$ Processus 1

```
/* Processus 0 envoi un message */
if (pid == 0)
{
    for (i=0; i<10; i++) buffer[i] = i;
    MPI_Ssend(buffer, 10, MPI_INT,
              1, 123, MPI_COMM_WORLD);
}
```

Réception bloquante : Synchrone

Processus 1 ⟵ Processus 0

```
if (pid == 1){
    /* Initialiser le buffer pour recevoir */
    for (i=0; i<10; i++) buffer[i] = -1;

    /* Recevoir le message  */
    MPI_Recv(buffer, 10, MPI_INT,
             0, 123, MPI_COMM_WORLD, &status);

    /* Afficher le message   */
    for (i=0; i<10; i++) printf("%3d", buffer[i]);
    printf("\n");
}
```

Résultat

0 1 2 3 4 5 6 7 8 9

Envoi bloquant : bufferisé

Envoi avec la routine MPI_Bsend

- L'émetteur copie d'abord les données dans un buffer local,
- L'appel à cette routine retourne immédiatement au programme appelant après avoir fini la copie des données

Réception avec la routine MPI_Recv

- Le récepteur se signale à l'émetteur
- Il provoque le transfert de données
- L'appel à cette routine retourne au programme appelant à la fin du transfert.

Notes

- Ce mode est parfois plus efficace
- Attention aux temps de recopie
- Le programmeur doit allouer, attacher et détacher ses buffers.

Envoi avec la routine MPI_Bsend

Processus émetteur

```
MPI_Pack_size(XXX)  /* Calcul de la taille du tampon*/
.....................................
.....................................
malloc ou calloc(XXX) /*Allocation la mémoire du tampon*/
.....................................
.....................................
MPI_Buffer_attach(XXX)  /* Attachement du tampon*/
.....................................
.....................................
MPI_Bsend(XXX)  /*Envoi du message*/
.....................................
.....................................
MPI_Buffer_detach(XXX) /* Détachement du tampon*/
free(XXX)  /* libération du tampon*/
.....................................
```

Association d'un buffer local pour un processus

Calcul de la taille du buffer

1. Taille d'un message MPI_Pack_size
2. Surcoût de taille pour chaque message : MPI_BSEND_OVERHEAD

MPI_Pack_size

```
int MPI_Pack_size(int incount,
                  MPI_Datatype datatype,
                  MPI_Comm comm,int *size);
```

1. int incount : Nombre d'éléments du paquet
2. MPI_Datatype datatype : Type en MPI d'un élément
3. MPI_Comm comm : Communicateur
4. int *size : La taille du message en octets

Attacher le buffer : Routine MPI_Buffer_attach

Prototype

```
int MPI_Buffer_attach(void *buffer, int size);
```

1. void *buffer : Adresse du tampon à attacher
2. int size : Taille du tampon en octets

- Cette routine fournit au système un tampon afin de copier le message avant son envoi
- Le tampon est utilisé uniquement par les messages envoyés par MPI_Bsend
- Un seul tampon peut être attaché à un processus à la fois

Routine MPI_Bsend

Prototype

```
int MPI_Bsend(void *buf, int count,
              MPI_Datatype datatype,
              int dest,int tag,MPI_Comm comm);
```

Paramètres

1. void *buf : Adresse du tampon d'envoi
2. int count : Nombre d'éléments à envoyer dans le tampon
3. MPI_Datatype datatype : Type de chaque élément envoyé
4. int dest : Identifiant du processus récepteur
5. int tag : Étiquette du message
6. MPI_Comm comm : Communicateur

Détacher un buffer : Routine MPI_Buffer_Detach

Prototype

```
int MPI_Buffer_detach(void *buffer, int *size);
```

Paramètres

1. void *buffer : Adresse du tampon à détacher
2. int *size : Taille du tampon en octets

- Cette routine détache le tampon actuel associé à MPI_Bsend
- L'appel de cette routine permet de bloquer le programme appelant jusqu'à ce que tous les messages dans ce tampon aient été transmis.
- Au retour de cette routine, l'utilisateur peut réutiliser ou désallouer l'espace occupé par le tampon

Envoi bloquant : standard

Envoi avec la routine MPI_Send

- Deux implémentations possibles
  1. Bufferisé
  2. Synchrone
- L'implémentation dépend de
  - La taille du message à échanger,
  - L'implémentation de MPI.

Réception avec la routine MPI_Recv

La transmission se termine lorsque le message est arrivé

Routine MPI_Send

Prototype

```
int MPI_Send(void *buf, int count,
             MPI_Datatype datatype,int dest,
             int tag, MPI_Comm comm);
```

Paramètres

1. void ***buff** : Adresse du tampon de données à envoyer
2. int **count** : Nombre d'éléments dans le tampon de données
3. MPI_Datatype **datatype** : Type de chaque élément envoyé
4. int **dest** : Identifiant du processus de destination
5. int **tag** : Étiquette de message
6. MPI_Comm **comm** : Communicateur

Routine MPI_Send

Exemple : Proc. 0 → Proc. 1

```
int pid, nprocs, i;
int buff[10];
MPI_Comm_rank(MPI_COMM_WORLD, &pid);
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
.....................................
.....................................
if (pid == 0){
  for(i=0;i<10;i++)
    buff[i]=i;
  MPI_Send(buff, 10, MPI_INT, 1, 9, MPI_COMM_WORLD);
}
.....................................
.....................................
```

Routine de MPI_Recv

Exemple : Proc. 1 ← Proc. 0

```
int pid, nprocs;
MPI_Status status;
int buff[10];
MPI_Comm_rank(MPI_COMM_WORLD, &pid);
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
.....................................
.....................................
if (pid == 1)
  MPI_Recv(buff, 10, MPI_INT,0, 9, MPI_COMM_WORLD,
           &status);
.....................................
.....................................
```

Envoi bloquant : ready

Attention

L'utilisation de cette communication est déconseillée

Envoi avec la routine MPI_Rsend

Cet appel n'a lieu que lorsque la réception est prête !

1. L'appel provoque le transfert de données
2. L'appel de cette routine retourne au programme appelant à la fin de la réception de données

Réception avec la routine MPI_Recv

- L'appel de cette routine se signale à l'émetteur
- Le récepteur attend et reçoit les données
- L'appel de cette routine retourne au programme appelant à la fin de la réception de données

Routine MPI_Rsend

Prototype

```
int MPI_Rsend(void *buf, int count,
              MPI_Datatype datatype,
              int dest,int tag,MPI_Comm comm)
```

Paramètres

1. void *buf : Adresse initiale du tampon d'envoi
2. int count : Nombre d'éléments envoyés
3. MPI_Datatype datatype : Type de chaque élément envoyé
4. int dest : Identifiant du processus récepteur
5. int tag : Étiquette du message
6. MPI_Comm comm : Communicateur

Combinaison l'envoi et la réception

Routine MPI_Sendrecv

- L'échange combine :
  - l'envoi d'un message vers un processus
  - la réception d'un autre message d'un autre ou du même processus
- Les buffers et les types peuvent être différents
- C'est une routine très utile pour des opérations dans une chaîne de processus
- Elle est pratique et efficace

Routine MPI_Sendrecv_replace

- Le même buffer est employé pour l'envoi et la réception,
- L'implémentation gère le stockage intermédiaire additionnel

Routine MPI_Sendrecv

Prototype

```
int MPI_Sendrecv(void *sendbuf, int sendcount,
                 MPI_Datatype sendtype,int dest,
                 int sendtag,void *recvbuf,
                 int recvcount,MPI_Datatype recvtype,
                 int source,int recvtag,
                 MPI_Comm comm,MPI_Status *status)
```

Paramètres

1. void *sendbuf : Adresse du tampon d'envoi
2. int sendcount : Nombre d'éléments envoyés
3. MPI_Datatype sendtype : Type de chaque élément envoyé
4. int dest : Identifiant du processus récepteur
5. int sendtag : Étiquette du message envoyé

Routine MPI_Sendrecv

Paramètres

6. void *recvbuf : Adresse initiale du tampon de réception
7. int recvcount : Nombre d'éléments de réception
8. MPI_Datatype recvtype, Type de chaque élément de réception
9. int source : Identifiant du processus émetteur
10. int recvtag : Étiquette du message de réception
11. MPI_Comm comm : Communicateur
12. MPI_Status *status : État du message de réception

Routine : MPI_Sendrecv

Exemple :

- Chaque processeur a deux buffers :
  1. buffer pour l'envoi
  2. buffer2 pour la réception
- Le processeur i
  - envoie les données dans le buffer au processeur i-1
  - reçoit les données du processeur i+1, et les met dans buffer2.

Routine : MPI_Sendrecv

Pseudo-code

```
12  MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
13  MPI_Comm_rank(MPI_COMM_WORLD, &myid);
14
15  right = (myid + 1) % numprocs;
16  left = myid - 1;
17
18  if (left < 0) left = numprocs - 1;
19
20   MPI_Sendrecv(buffer, 10, MPI_INT,
21                left, 123,
22                buffer2, 10, MPI_INT,
23                right, 123,
24               MPI_COMM_WORLD, &status);
```

Routine MPI_Sendrecv_replace

Prototype

```
int MPI_Sendrecv_replace(void *buf, int count,
                         MPI_Datatype datatype,
                         int dest,int sendtag,
                         int source,int recvtag,
                         MPI_Comm comm,
                         MPI_Status *status)
```

Paramètres

1. void *buf : Adresse initiale du tampon d'envoi et de réception
2. int count : Nombre d'éléments envoyés et de réception
3. MPI_Datatype datatype : Type de chaque élément

Routine MPI_Sendrecv_replace

Prototype

```
int MPI_Sendrecv_replace(void *buf, int count,
                         MPI_Datatype datatype,
                         int dest,int sendtag,
                         int source,int recvtag,
                         MPI_Comm comm,
                         MPI_Status *status)
```

Paramètres

4. int dest : Identifiant de processeur récepteur
5. int sendtag : Étiquette du message envoyé
6. int source : Identifiant du processeur émetteur

Routine MPI_Sendrecv_replace

Prototype

```
int MPI_Sendrecv_replace(*buf, count, datatype,
                         dest, sendtag, source,
                         recvtag, comm, *status);
```

Paramètres

7. int recvtag : Étiquette du message de réception
8. MPI_Comm comm : Communicateur
9. MPI_Status *status : État du message de réception

Routine : MPI_Sendrecv_replace

Pseudo-code

```
12    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
13    MPI_Comm_rank(MPI_COMM_WORLD, &myid);
14
15    right = (myid + 1) % numprocs;
16    left = myid - 1;
17
18    if (left < 0) left = numprocs - 1;
19
20     MPI_Sendrecv_replace(buffer, 10, MPI_INT,
21                          left, 123, right,
22                          123, MPI_COMM_WORLD,
23                          &status);
```

Communication non-bloquante

Attention

Une communication non bloquante ne garanti pas l'intégrité des données mais permet certaines optimisations. Il faut être très prudent dans son utilisation.

Principes

- Elle permet principalement d'optimiser le temps de communication
- L'appel retourne immédiatement, sans savoir si l'opération a été achevée

L'optimisation peut être utilisée en 3 phrases

1. Initialisation de la communication non-bloquante
2. Réalisation d'autres travaux : autres calculs, communication...
3. Attente de l'achèvement de la communication non-bloquante

Communication non-bloquante

Type de communication

- standard : MPI_Isend - MPI_Irecv
- bufférisation : MPI_Ibsend - MPI_Irecv
- synchrone : MPI_Issend - MPI_Irecv
- ready : MPI_Irsend - MPI_Irecv

L'utilisateur doit lui-même s'assurer que

Le message a bien été envoyé ou reçu avec des fonctions MPI adaptées

- MPI_Test : permet de tester si l'opération de communication, identifiée par request, est terminée
- MPI_Wait : attend que l'opération de communication se termine

Routine de réception non-bloquante MPI_Irecv

Prototype

```
int MPI_Irecv(void *buf, int count,
              MPI_Datatype datatype,
              int source,int tag,
              MPI_Comm comm, MPI_Request *request)
```

Paramètres

1. void ***buff** : Adresse initiale du tampon de données
2. int **count** : Nombre d'éléments dans le tampon de données
3. MPI_Datatype **datatype** : Type de chaque élément envoyé
4. int **src** : Identifiant du processus émetteur
5. int **tag** : Étiquette de message
6. MPI_Comm **comm** : Communicateur
7. MPI_Request ***request** : Requête de communication

Envoi non-bloquant Standard

Envoi avec la routine MPI_Isend

- Le processus émetteur
  - initialise l'opération d'envoi
  - mais rend la main avant sa réalisation
- L'appel sera terminé avant que le message ne soit parti
- Après l'initialisation et avant l'envoi effectif, d'autres calculs peuvent simultanément se faire

Réception avec la routine MPI_Irecv

- Le processus récepteur
  - initialise la réception
  - mais rend la main avant sa réalisation
- L'appel sera terminé avant que le message ne soit reçu
- Après l'initialisation et avant la réception effective, d'autres calculs peuvent simultanément se faire

Routine MPI_Isend

Prototype

```
int MPI_Isend(void *buf, int count,
              MPI_Datatype datatype,
              int dest, int tag,
              MPI_Comm comm, MPI_Request *request)
```

Paramètres

1. void ***buff** : Adresse initiale du tampon de données,
2. int **count** : Nombre d'éléments dans le tampon de données,
3. MPI_Datatype **datatype** : Type de chaque élément envoyé,
4. int **dest** : Identifiant du processus de destination,
5. int **tag** : Étiquette de message,
6. MPI_Comm **comm** : Communicateur
7. MPI_Request *request

Envoi non-bloquant synchrone

Envoi avec la routine MPI_Issend

```
int MPI_Issend(void *buf, int count,
               MPI_Datatype datatype,
               int dest, int tag,
               MPI_Comm comm, MPI_Request *request)
```

Envoi non-bloquant bufferisé

Envoi avec la routine MPI_Ibsend

```
int MPI_Ibsend(void *buf, int count,
               MPI_Datatype datatype,
               int dest, int tag,
               MPI_Comm comm, MPI_Request *request)
```

Envoi non-bloquant ready

Envoi avec la routine MPI_Irsend

```
int MPI_Irsend(void *buf, int count,
               MPI_Datatype datatype,
               int dest, int tag,
               MPI_Comm comm, MPI_Request *request)
```

Routine MPI_Test

Prototype

```
int MPI_Test(MPI_Request *request,
             int *flag, MPI_Status *status)
```

- Elle vérifie l'achèvement d'une opération associée à request
- Elle donne **flag=true** si l'opération (send/recv) identifiée par **request** est terminée
- request est alors libérée et mis en valeur à MPI_REQUEST_NULL par cet appel
- status contient des informations sur l'opération terminée

Routine MPI_Test

Utilisation de MPI_Test

```
.....................................
.....................................
MPI_Irecv(buffer, 10, MPI_INT,
          left, 123, MPI_COMM_WORLD,
          &request);

MPI_Test(&request, &flag, &status);
while (!flag)
{
    /* Do some work ... */
    .....................................
    .....................................
    MPI_Test(&request, &flag, &status);
}
.....................................
.....................................
```

Routine MPI_Wait

Prototype

```
int MPI_Wait(MPI_Request *request, MPI_Status *status)
```

- Cette routine retourne au programme appelant lorsque l'opération identifiée par **request** est complète.
- **request** est alors libérée et mis en valeur à MPI_REQUEST_NULL par cet appel.
- **status** contient des informations lorsque l'opération est terminée

Routine MPI_Wait

Utilisation de MPI_Wait

```
.....................................
.....................................
MPI_Irecv(buffer, 10, MPI_INT,
          left, 123, MPI_COMM_WORLD,
          &request);

MPI_Isend(buffer2, 10, MPI_INT,
          right, 123, MPI_COMM_WORLD,
          &request2);

MPI_Wait(&request, &status);
MPI_Wait(&request2, &status);
.....................................
.....................................
```

Communication point-à-point : Résumé

Fonctions MPI d'envoi et de réception

Types/Modes	Bloquant	Non bloquant
Envoi standard	MPI_Send	MPI_Isend
Envoi synchrone	MPI_Ssend	MPI_Issend
Envoi bufferisé	MPI_Bsend	MPI_Ibsend
Envoi ready	MPI_Rsend	MPI_Irsend
Réception	MPI_Recv	MPI_Irecv