Message Passing Interface (MPI) – II
Communications point-à-point

Amal KHABOU
amal.khabou@lri.fr

Aperçu

- Les fonctions disponibles pour les communications point-à-point
- C’est quoi le statut d’un message?
- Les communications non-bloquantes

Ce qu'on a appris jusqu'ici

- Six fonctions MPI sont suffisantes pour programmer des machines à mémoire distribuée

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

MPI_Comm_rank (MPI_Comm comm, int *rank);
MPI_Comm_size (MPI_Comm comm, int *size);

MPI_Send (void *buf, int count, MPI_Datatype dat,
int dest, int tag, MPI_Comm comm);
MPI_Recv (void *buf, int count, MPI_Datatype dat,
int source, int tag, MPI_Comm comm, MPI_Status
*status);
```

Les communications point-à-point

- Échange de données entre deux processus
  - Les deux processus participent dans l'échange des données => communication bilatérale
- Un large ensemble de fonctions définies dans MPI-1 (50+)

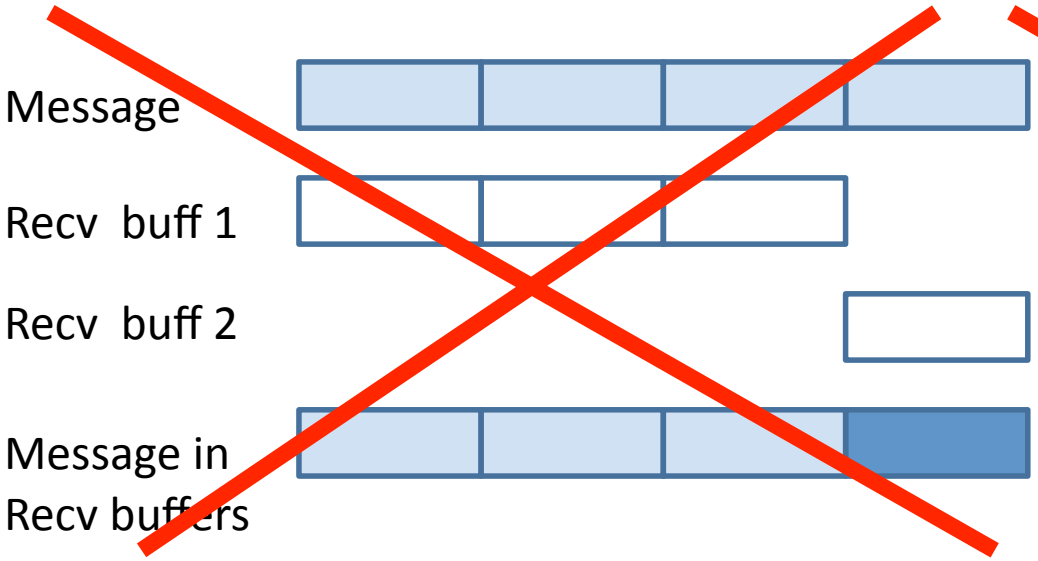
	Blocking	Non-blocking	Persistent
Standard	MPI_Send	MPI_Isend	MPI_Send_init
Buffered	MPI_Bsend	MPI_Ibsend	MPI_Bsend_init
Ready	MPI_Rsend	MPI_Irsend	MPI_Rsend_init
Synchronous	MPI_Ssend	MPI_Issend	MPI_Ssend_init

Un message contient …

- les données à envoyer par l'expéditeur au correspondant, décrites par
  - Le début du buffer d'envoi
  - le type des données
  - Le nombre de données du type précisé
- l'enveloppe du message (message header)
  - rang du processus expéditeur
  - rang du processus receveur
  - le communicateur
  - un tag

Règles pour les communications point-à-point

- **Fiabilité**: MPI garantie qu'aucun message n'est perdu
- **Dépassement interdit**: MPI garantie que 2 messages postés du processus A au processus B arrivent dans le même ordre que l'envoi
- MPI spécifie qu'un message donné ne peut pas être reçu par plus d'une fonction Recv (par opposition aux sockets!)




```
if ( rank == 0 ) {
MPI_Send ( buf, 4, …);
}
if ( rank == 1 ) {
MPI_Recv (buf, 3, …);
MPI_Recv(&(buf[3]), 1,
…);
}
```

Appariement des messages (I)

- Comment le processus effectuant la réception peut savoir si le message qu'il vient de recevoir est celui qu'il attendait?
  - l'expéditeur du message reçu doit correspondre à l'expéditeur du message attendu
  - le tag du message reçu doit correspondre au tag du message attendu
  - le communicateur du message reçu doit correspondre au communicateur du message attendu

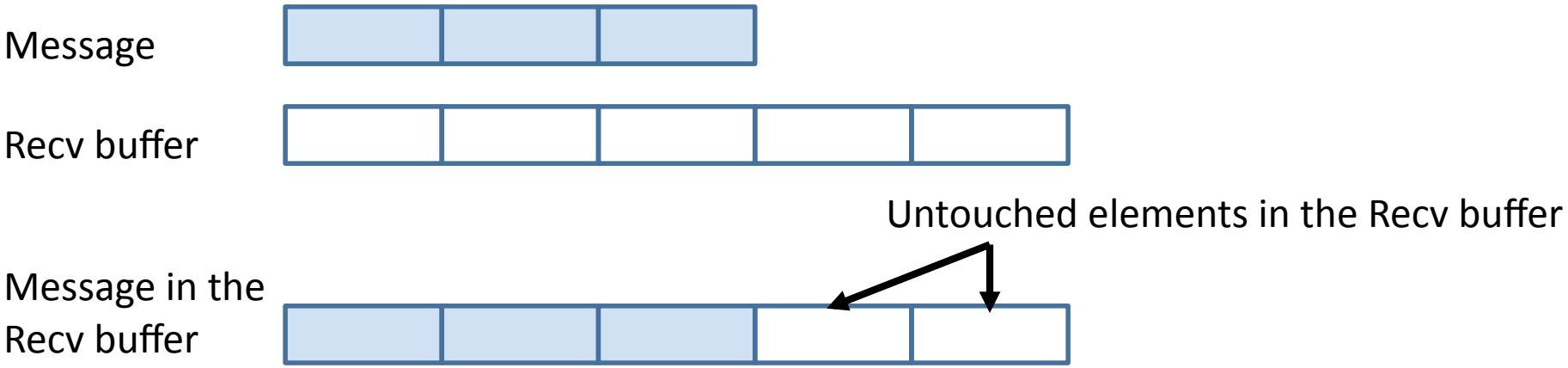
Appariement des messages (II)

- Que se passe t-il si la taille du message reçu ne correspond pas à la taille du message attendu?
  - la taille du message n'est pas un critère de correspondance
  - si elle est plus petite, pas de problème
  - si elle est plus grande
    - une erreur (MPI_ERR_TRUNC) sera renvoyée
    - l'application s'arrête
    - l'application se bloque
    - un core-dump

Appariement des messages (III)

- Exemple 1: exemple correcte

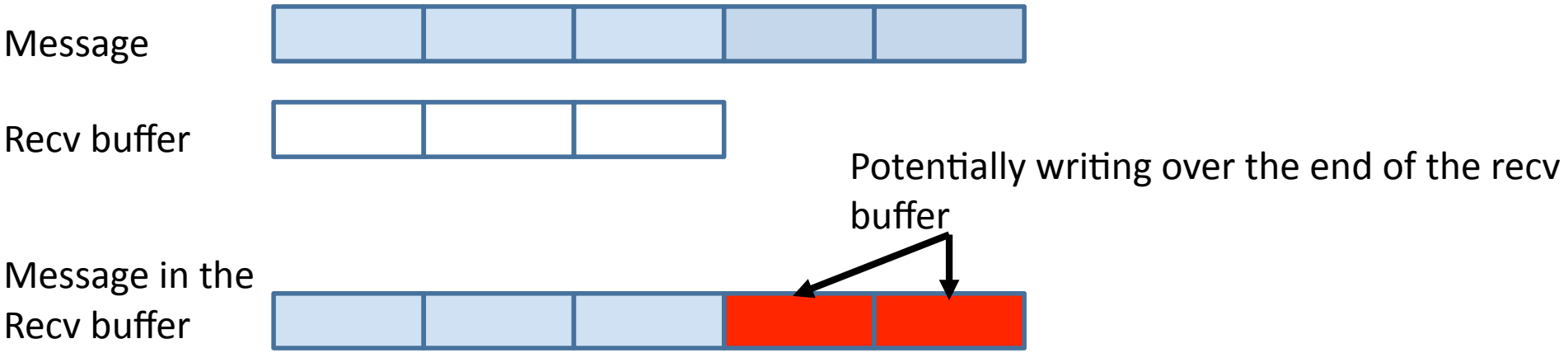
```
if (rank == 0 ) {
MPI_Send(buf, 3, MPI_INT, 1, 1, MPI_COMM_WORLD);
}
else if ( rank == 1 ) {
MPI_Recv(buf, 5, MPI_INT, 0, 1, MPI_COMM_WORLD,
&status);
}
```



Appariement des messages (IV)

- Exemple 2: exemple erroné

```
if (rank == 0 ) {
MPI_Send(buf, 5, MPI_INT, 1, 1, MPI_COMM_WORLD);
}
else if ( rank == 1 ) {
MPI_Recv(buf, 3, MPI_INT, 0, 1, MPI_COMM_WORLD,
&status);
}
```



Blocage (I)

- **Question**: Comment deux processus peuvent s'échanger des données en même temps sans problèmes?
- Possibilité 1

Process 0

```
MPI_Send(buf,…)
MPI_Recv(buf,…)
```

Process 1

```
MPI_Send(buf,…)
MPI_Recv(buf,…)
```

  - Peut causer un blocage, ça dépend de la taille des messages et la capacité de la bibliothèque MPI à bufferiser les messages

Blocage (II)

- Possibilité 2: reordonner les fonctions MPI d'un seul processus

Process 0

```
MPI_Recv(rbuf,…)
MPI_Send(buf,…)
```

Process 1

```
MPI_Send(buf,…)
MPI_Recv(rbuf,…)
```

- Autres possibilitiés:
  - communications asynchrones – détail plus tard
  - un envoi bufferisé (MPI_Bsend) – pas de détails ici
  - utiliser MPI_Sendrecv – pas de détails ici

Exemple

- Implémentation d'un anneau en utilisant Send/Recv
  - Le processus de rang 0 commence l'anneau

```
MPI_Comm_rank (comm, &rank);
MPI_Comm_size (comm, &size);

if (rank == 0 ) {
MPI_Send(buf, 1, MPI_INT, rank+1, 1, comm);
MPI_Recv(buf, 1, MPI_INT, size-1, 1, comm, &status);
}
else if ( rank == size-1 ) {
MPI_Recv(buf, 1, MPI_INT, rank-1, 1, comm, &status);
MPI_Send(buf, 1, MPI_INT, 0, 1, comm);
}
else {
MPI_Recv(buf, 1, MPI_INT, rank-1, 1,comm,&status);
MPI_Send(buf, 1, MPI_INT, rank+1, 1,comm);
}
```

Les jokers

- **Question:** Est ce qu'il est possible d'utiliser des arguments jockers dans les fonctions Send/Recv?
- **Réponse:**
  - Pour Send: non
  - Pour Recv:
    - tag: oui, MPI_ANY_TAG
    - source: oui, MPI_ANY_SOURCE
    - communicator: non

Statut d'un message (I)

- le MPI status contient des informations directement accessibles
  - qui envoie le message
  - c'était quoi le tag
  - c'est quoi le type d'erreur du message
- … d'autres informations accessibles à travers des appels de fonctions
  - la taille du message
  - Est ce que le message a été annulé?

Statut d'un message (II) – C

```
MPI_Status status;

MPI_Recv ( buf, cnt, MPI_INT, …, &status);

/*directly access source, tag, and error */
src = status.MPI_SOURCE;
tag = status.MPI_TAG;
err = status.MPI_ERROR;

/*determine message length and whether it has been
cancelled */
MPI_Get_count (status, MPI_INT, &rcnt);
MPI_Test_cancelled (status, &flag);
```

Statut d’un message (III)

- Si vous n’êtes pas interessé par le statut, vous pourriez passer en argument
  - MPI_STATUS_NULL
  - MPI_STATUSES_NULL

  pour MPI_Recv et toutes les autres fonctions MPI qui retournent un statut

Communications non-bloquantes (I)

- La fonction MPI_Send est finalisée (returns) quand les données sont stockées ailleurs en toute sécurité
- La fonction MPI_Recv est finalisée (returns), quand toutes les données sont disponibles dans le buffer de réception
- Les communications non-bloquantes initialisent les opérations Send et Receive mais n'attendent pas leur achèvement
- Les fonctions qui vérifient ou attendent l'achèvement d'une communication initialisée doivent être appelées explicitement
- Les opérations non-bloquantes "return" *immédiatement,* toutes les fonctions MPI correspondantes commencent par le préfixe `I` (e.g. `MPI_Isend` or `MPI_Irecv`).

Communications non-bloquantes (II)

```
MPI_Isend (void *buf, int cnt, MPI_Datatype
dat, int dest, int tag, MPI_Comm comm,
MPI_Request *req);


MPI_Irecv (void *buf, int cnt, MPI_Datatype
dat, int src, int tag, MPI_Comm comm,
MPI_Request *reqs);
```

Communications non-bloquantes (III)

- Aprés l'initialisation d'une communication non-bloquante, on ne peut pas toucher au buffer avant son achècement i.e. on ne peut pas faire de supposition sur l'instant à partir duquel le message a été réellement transfèré

- Toutes les fonctions non-bloquantes ont un argument supplémentaire: un *request*

- Un request identifie d'une façon unique une communication en cours, il est utiliser pour vérifier ou attendre l'achèvement d'une communication en cours

Fonctions du contrôle de la fin (I)

```
MPI_Wait (MPI_Request *req, MPI_Status *stat);
MPI_Waitall (int cnt, MPI_Request *reqs,
MPI_Status *stats);
MPI_Waitany (int cnt, MPI_Request *reqs, int
*index,
MPI_Status *stat);
MPI_Waitsome (int cnt, MPI_Request *reqs, int
*outcnt, int *indices, MPI_Status *stats);
```

- Fonctions d'attente d'achèvement
  - MPI_Wait – attend la fin d'une communication
  - MPI_Waitall – attend la fin de toutes les commud'une liste
  - MPI_Waitany – attend la fin d'une comm dans une liste
  - MPI_Waitsome – attend la fin d'au moins une comm dans une liste
- Le contenu du statut n'est pas défini pour la fonction Send

Fonctions du contrôle de la fin (II)

```
MPI_Test     (MPI_Request *req, int *flag,
              MPI_Status *stat);
MPI_Testall  (int cnt, MPI_Request *reqs, int *flag,
              MPI_Status *stats);
MPI_Testany  (int cnt, MPI_Request *reqs, int *index,
              int *flag, MPI_Status *stat);
MPI_Testsome(int cnt, MPI_Request *reqs, int*outcnt,
              int *indices, int *flag,
              MPI_Status *stats);
```

Les fonctions de test vérifient si une communication non-bloquante est arrivée à terme ou pas

- MPI_Test
- MPI_Testall
- MPI_Testany
- MPI_Testsome

Problème du blocage revisité

- **Question**:Comment deux processus peuvent s'échanger des données en même temps sans problèmes?
- Possibilité 3: utilisation des communications non-bloquantes

Process 0

```
MPI_Irecv(rbuf,…,&req);
MPI_Send (buf,…);
MPI_Wait (req,
&status);
```

Process 1

```
MPI_Irecv(rbuf,…,&req);
MPI_Send (buf,…);
MPI_Wait (req,
&status);
```

- remarque:
  - il faut utiliser 2 buffers différents!
  - plusieurs façons pour formuler ce scenario
  - le même code pour les deux processus