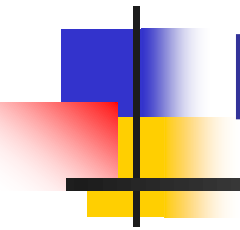
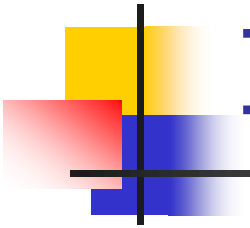


Programmation Parallèle et Distribuée

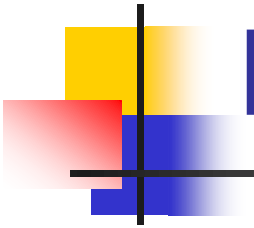
Cours 3 : Communications point-à-point avancées

Patrick Carribault
David Dureau
Marc Pérache (marc.perache@cea.fr)



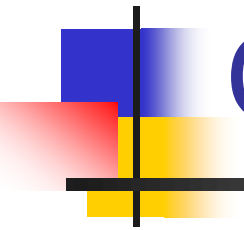
Introduction

- MPI offre des mécanismes de communications point-à-point bien plus performants que les "classiques" `MPI_Send` et `MPI_Recv`
- MPI permet à l'utilisateur de choisir :
  - Parmi plusieurs protocoles de communications
  - Les types d'appels d'envoi et de réception (appels bloquants ou non)
- MPI permet également de recevoir des messages sans que le destinataire connaisse à l'avance la taille du message

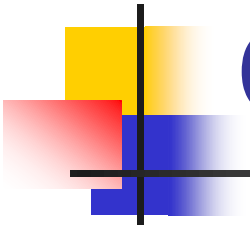


Plan du cours 3

- Communications bloquantes
  - Mode synchrone
  - Mode bufferisé
  - Mode standard
- Communications non bloquantes
  - Envoi/réception
  - Couplage bloquant/non bloquant
- Recevoir un message de taille quelconque



Communications bloquantes



Communications bloquantes

- Définition
  - Un envoi *send* est dit `bloquant` ssi au retour du *send* il est possible d'écrire dans le buffer d'envoi sans altérer le contenu du message
- Autrement dit
  - un *send* `bloquant` ne rendra pas la main (blocage) tant que la bibliothèque de communication n'aura pas géré le message et garanti la viabilité de son contenu.

Communications bloquantes

T0

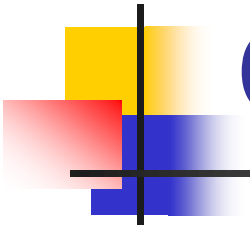
```
a = 100;
send(&a, 1, T1);
a = 0;
```

T1

```
recv(&a, 1, T0);
printf("%d\n", a);
```

- Après le *send*, T0 peut modifier le contenu de `a`
- T1 recevra bien la valeur 100 (valeur de `a` lors de l'appel de T0 à *send*) ;

- Remarque :
  - Dire qu'un envoi est bloquant ne revient pas à dire que le message ait été reçu par le destinataire !



Communications bloquantes

- Définition
  - Une réception *recv* est `bloquante` ssi au retour du *recv* le buffer de réception contient bien le contenu du message.
- Autrement dit
  - La fonction *recv* bloquera (ne rendra pas la main) tant qu'elle n'aura pas reçu et affecté le contenu du message.

Communications bloquantes

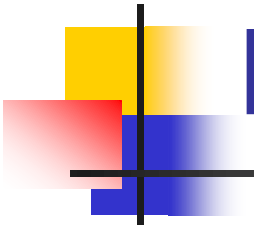
T0

```
a = 100;
send(&a, 1, T1);
a = 0;
```

T1

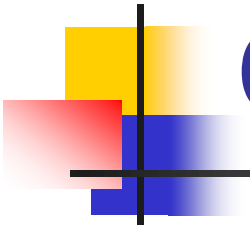
```
recv(&a, 1, T0);
printf("%d\n", a);
```

- Après le *send*,
  - T0 peut modifier le contenu de a
- Après le *recv*,
  - Le contenu du message peut être manipulé (lecture, écriture, affichage, ...) immédiatement



Modes de communications

- Modes de communications bloquantes
  1. Mode synchrone
  2. Mode bufferisé
  3. Mode standard

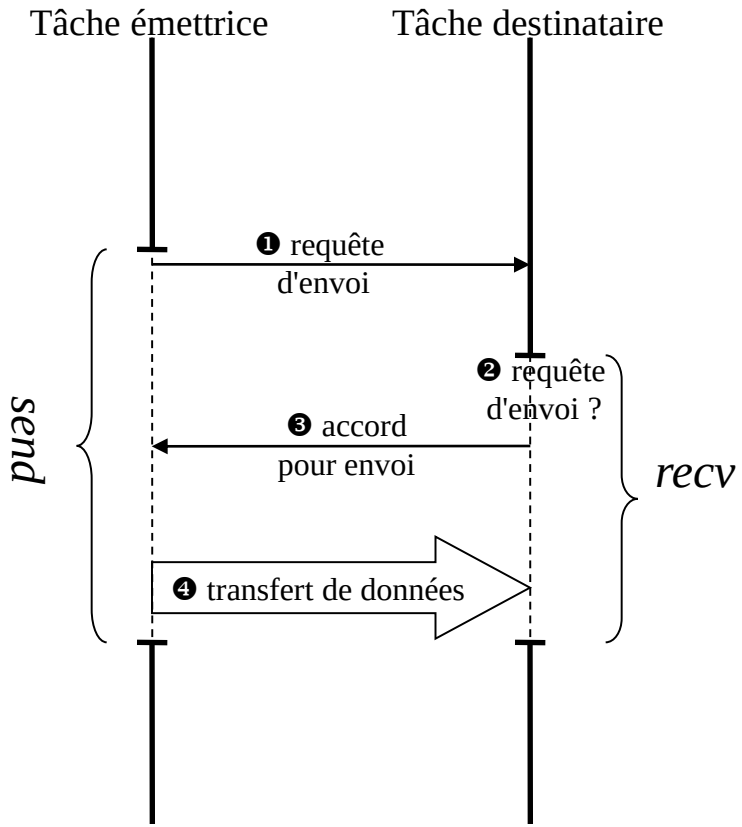


Communication synchrone

- Définition :
  - Un envoi synchrone bloquant rendra la main quand le message aura été reçu par le destinataire
- Implémentation
  - Besoin de synchroniser l'émetteur et le récepteur
  - Mise en place d'un protocole pour le transfert des données

Communication synchrone

- Protocole de communication synchrone

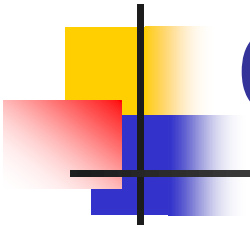


❶ Lors d'un envoi synchrone, l'expéditeur envoi au destinataire une requête d'envoi et attend que le destinataire lui réponde ;

❷ Quand le destinataire débute sa réception, il attend une requête d'envoi de l'expéditeur ;

❸ Quand le destinataire a reçu la requête d'envoi, il répond à l'expéditeur en lui accordant l'envoi ;

❹ L'expéditeur et le destinataire sont alors synchronisé, le transfert de données (*i.e.* le message proprement dit) a lieu ; l'envoi et la réception sont alors terminés.



Communication synchrone

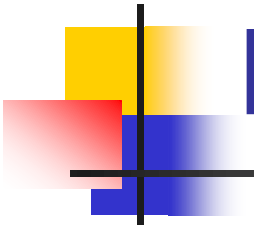
- Avantages
  - Pas de copie dans un buffer interne
  - Echange de message par mécanismes directs d'accès distants à la mémoire d'autres processus (DMA ou RDMA)
- Inconvénients
  - Impose un *rendez-vous* entre l'émetteur et le récepteur
  - Attente potentiellement inutile
- Cas optimal
  - Utilisation dans le cas où l'envoi et la réception sont appelées en même temps
  - Exemple : exploitation d'un parallélisme de données avec charge équilibrée entre les tâches

Communication synchrone MPI

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

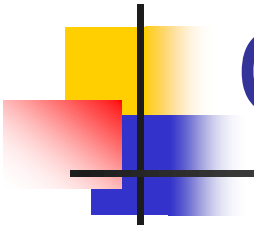
- La signature est la même que `MPI_Send`.
On force l'utilisation du mode synchrone d'échange de message.

- Réception avec la fonction `MPI_Recv`



Modes de communications

- Modes de communications bloquantes
  1. Mode synchrone
  2. Mode bufferisé
  3. Mode standard

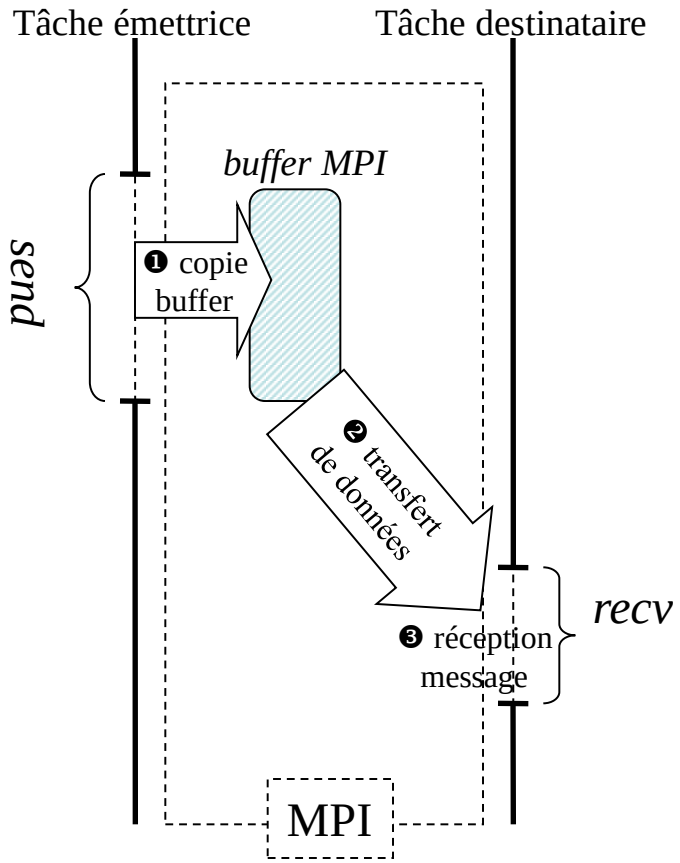


Communication bufferisée

- Définition
  - Un envoi bufferisé bloquant rendra la main quand le message aura été copié dans un buffer géré par la bibliothèque de communication

Communication bufferisée

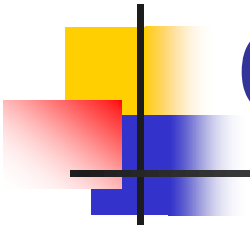
- Protocole de communication bufferisée



❶ Lors d'un envoi bufferisé, l'expéditeur copie le message dans un buffer géré par la bibliothèque de communication ; l'envoi peut alors rendre la main.

❷ La bibliothèque de communication s'approprie alors le message et peut l'envoyer au destinataire.

❸ Le destinataire reçoit le message dès que possible.



Communication bufferisée

- Avantages
  - Découplage entre le *send* et le *recv* : le *send* peut retourner avant que le *recv* ait été posté
- Inconvénients
  - Copie dans un buffer (surcoût CPU + surcoût mémoire et bande passante)
  - Taille limitée
- Cas optimal
  - les envois et les réceptions ne sont pas bien synchronisées (déséquilibrage de charge)

Allocation buffer

- L'utilisateur peut remplacer le buffer MPI par son propre buffer alloué dans son espace d'adressage
  - Attachement d'un buffer `buf` (alloué par l'utilisateur) de taille `sz` octets;

```
int MPI_Buffer_attach(void *buf, int sz);
```

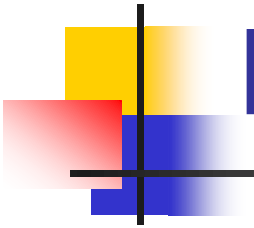
- Le buffer utilisateur peut être relâché par MPI (MPI réutilise alors son propre buffer)
  - Détachement d'un buffer : retourne l'adresse du buffer ainsi que sa taille

```
int MPI_Buffer_detach(void **buf_adr, int
                      *sz);
```

Allocation buffer

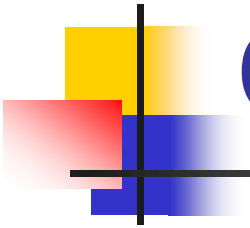
```
#define BUFFSIZE 100000
int sz;
char *buf;

MPI_Buffer_attach( malloc(BUFFSIZE), BUFFSIZE );
…
MPI_Bsend(msg1, …);
MPI_Bsend(msg2, …);
…
MPI_Buffer_detach( &buf, &sz );
free(buf);
```

- Le buffer utilisateur n'est utilisé que pour `MPI_Bsend`
- Ne pas confondre buffer MPI et buffer d'envoi
- Un seul buffer ne peut être attaché à la fois
- Il est inutile que le destinataire attache un buffer



Modes de communications

- Modes de communications bloquantes
  1. Mode synchrone
  2. Mode bufferisé
  3. Mode standard



Communication standard

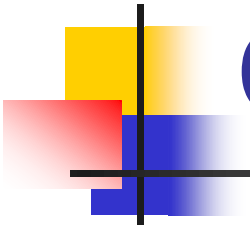
- Fonction pour une communication standard
  - `MPI_Send`
  - Explications dans le cours précédent

- Protocole pour un envoir standard :
  - MPI considère un message de taille T
    - Si le message à envoyer est de taille inférieure à T, l'envoi est alors bufferisé
    - Si le message à envoyer est de taille supérieure à T, l'envoi est alors synchronisé

Communication standard : Exemple

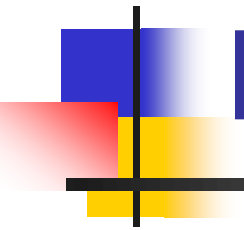
```
if ( rang == 0 )
voisin = 1;
else if ( rang == 1 )
voisin = 0;

MPI_Send(&msg1, N, MPI_BYTE, voisin, tag1, comm);
MPI_Recv(&msg2, N, MPI_BYTE, voisin, tag2, comm);
```

- Ce code est-il sur ?
- NON:
  - Si la taille du message N est petite, alors le programme se déroulera sans problème
  - Sinon le programme bloquera (*deadlock*) car l'envoi sera synchrone

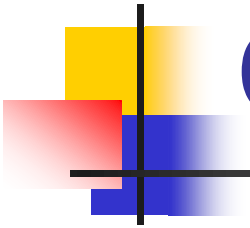


Communication standard

- Solution pour détecter ces problèmes :
  - Remplacer tous les appels à `MPI_Send` par `MPI_Ssend`
  - Peu importe la taille du message, le programme ne doit pas bloquer
- Si le programme bloque
  - Bug dans le programme



Communications non-bloquantes



Communication non-bloquante

- Définition
  - une communication est dite non bloquante ssi au retour de la fonction, la bibliothèque de communication ne garantit pas que l'échange de message ait eu lieu

- L'accès sûr aux données n'est donc pas garanti après une communication non bloquante !
  - Pour pouvoir réutiliser les données du message, il faudra appeler une fonction supplémentaire qui complètera le message (*i.e.* qui assure un accès sûr aux données)

Envoi non-bloquant MPI_Isend

```
int MPI_Isend (
void *buf(in),
int count(in),
MPI_Datatype datatype(in),
int dest(in),
int tag(in),
MPI_Comm comm(in),
MPI_Request *req(out)
);
```

La signature est la même que MPI_Send hormis l'argument supplémentaire MPI_Request *req.

MPI_Isend demande une requête d'envoi.
L'identifiant de la requête est retourné dans *req (MPI_Request = type opaque encapsulant une requête MPI).

Un appel à MPI_Isend ne garantit pas que la bibliothèque de communication s'est appropriée le contenu du message.

Terminaison : MPI_Wait

```
int MPI_Wait (
MPI_Request *req(inout),
MPI_Status *sta(out)
);
```

`MPI_Wait` bloquera jusqu'à ce que la requête de communication identifiée par `*req` soit terminée.

Des informations relatives à la communication sont retournées dans `*sta`.

Au retour de `MPI_Wait`,

- `*req` est affectée à `MPI_REQUEST_NULL` (invalide la requête) ;
- il est possible d'écrire dans le buffer d'envoi utilisé par `MPI_Isend`.

Remarque :

`MPI_Send` ⇔ `MPI_Isend` + `MPI_Wait`

Exemple

```
MPI_Request req;
MPI_Status sta;

MPI_Isend(buf, N, MPI_BYTE,
dest, tag1, comm,
&req);

instruction1;
instruction2;
…
instructionN;

MPI_Wait(&req, &sta);
```

Les instructions entre `MPI_Isend` et `MPI_Wait` ne peuvent pas écrire dans le buffer `buf`.

Pendant ce temps, le message est envoyé.

- Avantage ?
  - Couvrir les communications par du calcul (la communication peut s'opérer pendant l'exécution des instructions de calcul)

Réception non bloquante

```
int MPI_Irecv (
void *buf(out),
int count(in),
MPI_Datatype datatype(in),
int source(in),
int tag(in),
MPI_Comm comm(in),
MPI_Request *req(out)
);
```

La signature est la même que `MPI_Recv` hormis le dernier argument `MPI_Request *req`.

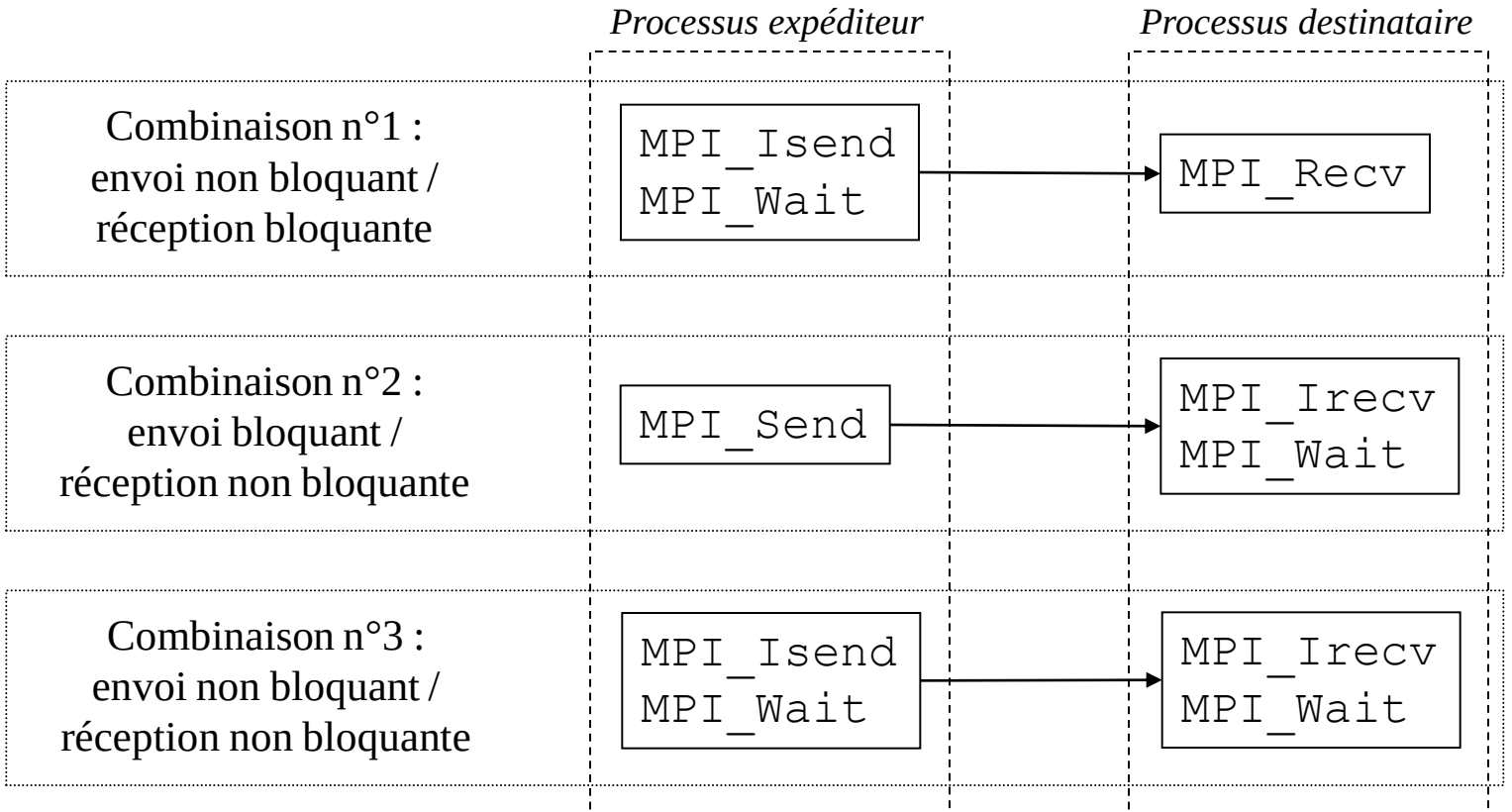
`MPI_Irecv` demande une requête de réception. L'identifiant de la requête est retourné dans `*req`.

Un appel à `MPI_Irecv` ne garantit pas la réception du message.

Pour terminer la réception, un appel à `MPI_Wait` est nécessaire.

Couplage bloquant/non bloquant

- Un envoi non bloquant peut être réceptionné par une réception bloquante et vice versa.



Communication non-bloquante

```
int MPI_Test (
MPI_Request *req(inout),
int *flag(out),
MPI_Status *sta(out)
);
```

Retourne vrai (valeur non nulle) dans `*flag` si et seulement si la requête `*req` est terminée.

Si `*flag` est vrai, alors `*req` est affectée à `MPI_REQUEST_NULL` et `*sta` est rempli.

Si `*flag` est faux, les contenus de `*req` et `*sta` ne sont pas garantis.

Communication non-bloquante

- Exemple :

```
MPI_Irecv(msg, N, MPI_BYTE, dest, tag, comm, &req);
do {
instruction1;
…
instructionN;
MPI_Test(&req, &flag, &sta);
} while( !flag );
```

Communications non-bloquantes

```
int MPI_Waitall (
int nb_req(in),
MPI_Request *tab_req(inout),
MPI_Status *tab_sta(out)
);
```

Retourne quand les `nb_req` requêtes contenues dans le tableau `tab_req` sont terminées.

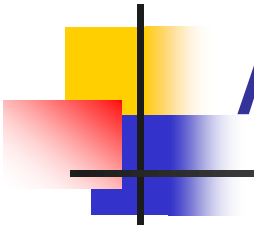
Les statuts sont retournés dans le tableau `tab_sta`.

Remarque :
L'ordre dans lequel les requêtes se terminent n'a pas d'importance.

Communications non-bloquantes

- Exemple : envoi/réception avec gauche/droite

```
MPI_Request req[4];
MPI_Status sta[4];

gauche = (rang + P - 1) % P;
droite = (rang + 1) % P;

MPI_Isend(&x[1], 1, MPI_DOUBLE, gauche, tag, comm, req);
MPI_Isend(&x[N], 1, MPI_DOUBLE, droite, tag, comm, req+1);
MPI_Irecv(&x[0], 1, MPI_DOUBLE, gauche, tag, comm, req+2);
MPI_Irecv(&x[N+1], 1, MPI_DOUBLE, droite, tag, comm, req+3);

MPI_Waitall(4, req, sta);
```



Autres fonctions

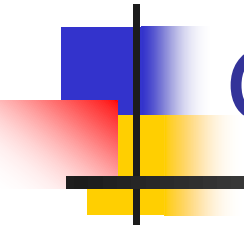
- MPI propose un large choix de fonctions pour compléter les communications non bloquantes
- `MPI_Testall`
  - Teste si toutes les requêtes d'un ensemble sont terminées
- `MPI_Waitany` / `MPI_Testany`
  - Attend/teste jusqu'à ce qu'une requête soit terminée
  - Retourne l'indice de la requête terminée
- `MPI_Waitsome` / `MPI_Testsome`
  - Attend/teste jusqu'à ce qu'une requête ou plusieurs requêtes soient terminées
  - retourne un tableau de requêtes terminées

Communications et modes

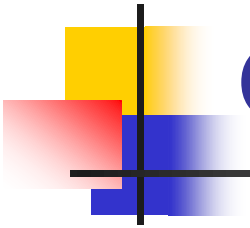
- Ne pas confondre communications non bloquantes et communications asynchrones
- On peut avoir des communications non bloquantes tout en choisissant le mode de communication : synchrone, bufferisé ou standard

Type/Mode	Standard	Bufferisé	Synchrone
Bloquant	MPI_Send	MPI_Bsend	MPI_Ssend
Non bloquant	MPI_Isend	MPI_Ibsend	MPI_Issend

Recevoir un message de taille quelconque



Recevoir un message de taille quelconque



- Comment faire pour recevoir un message de taille quelconque (par exemple, pour recevoir un message auto décrit) ?
  - La fonction `MPI_Recv` oblige de connaître une borne maximale de la taille du message à recevoir
    - `MPI_Recv` n'est pas approprié
  - MPI définit des fonctions qui permettent de récupérer des informations sur un message envoyé avant de le réceptionner : `MPI_Iprobe` et `MPI_Probe`

Vérifier l'arrivée d'un message

```
int MPI_Iprobe (
int source(in),
int tag(in),
MPI_Comm comm(in),
int *flag(out),
MPI_Status *sta(out)
);
```

Vérifie si un message provenant de l'expéditeur `source` avec l'étiquette `tag` est arrivé (`MPI_ANY_SOURCE` et `MPI_ANY_TAG` autorisés).

Retourne vrai (valeur non nulle) dans `*flag` si un message est arrivé, faux (valeur nulle) sinon.

Dans le cas où un message est arrivé, le statut `*sta` est rempli.
Son contenu n'est pas défini si un message n'est pas arrivé.

Attendre l'arrivée d'un message

```
int MPI_Probe (
int source(in),
int tag(in),
MPI_Comm comm(in),
MPI_Status *sta(out)
);
```

Attend jusqu'à ce qu'un message provenant de l'expéditeur `source` avec l'étiquette `tag` est arrivé (`MPI_ANY_SOURCE` et `MPI_ANY_TAG` autorisés).

Au retour de `MPI_Probe`, le statut `*sta` est rempli.

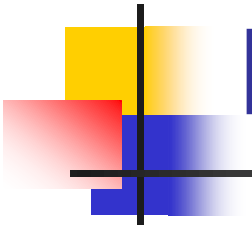
Recevoir un message après `MPI_Iprobe/MPI_Probe`

- Les appels à `MPI_Iprobe` et `MPI_Probe` vérifient ou attendent l'arrivée d'un message mais n'effectuent pas la réception proprement dite.

- Pour recevoir le message :
  1. Appel à `MPI_Get_count` pour retrouver la taille du message
  2. Allocation un buffer à la taille retournée ;
  3. Appel à `MPI_Recv` pour réceptionner le message lui-même.

Recevoir un message après MPI_Iprobe/MPI_Probe

```
MPI_Status sta;
int taille, arrive;
do {
instruction1;
…
instructionN;
MPI_Iprobe(MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &arrive, &sta);
} while (!arrive);

MPI_Get_count(&sta, MPI_BYTE, &taille);
char *buf = malloc( taille );
MPI_Recv(buf, taille, MPI_BYTE, sta.MPI_SOURCE, sta.MPI_TAG,
MPI_COMM_WORLD, &sta);
```



Résumé

- MPI permet de sélectionner les modes d'envoi :
  - synchrone (peut permettre une copie mémoire-à-mémoire)
  - bufferisé (permet découplage envoi/réception)
  - standard (le plus portable)
- Les appels non bloquants permettent :
  - de couvrir les communications par du travail utile
  - d'éviter des *deadlocks*
- Toute communication non bloquante doit être terminée à l'aide d'une fonction du type `MPI_Wait*`/`MPI_Test*` ;
- Grâce à `MPI_Iprobe`/`MPI_Probe`, on peut recevoir des messages auto-décrit avec MPI.