Cours Calcul Intensif - MPI

Hélène Coullon , Sophie Robert, Sébastien Limet

07 novembre 2012

Communications collectives

Les communications collectives permettent de réaliser des communications impliquant plusieurs processus.

- Elles peuvent toujours être simulées par un ensemble de communications point à point mais sont très optimisées
- Une communication collective implique l'ensemble des processus du communicateur utilisé
- Il s'agit d'appels bloquants (le processus ne récupère la main que lorsque sa participation à la communication est terminée). Voir norme MPI 3.0 pour des communications collectives non bloquantes.
- Les communications collectives n'impliquent pas de synchronisation globale (sauf MPI_Barrier) et n'en nécessitent pas.
- Les communications collectives n'interfèrent jamais avec les communications point à point

Il faut **toujours** (ou presque) les préférer aux communications point à point.

Type de communications collectives MPI

Les communications collectives peuvent être séparées en 3 catégories :

- Synchronisations globales (**MPI_Barrier**) à n'utiliser que si nécessaire (rare)
- Transferts/échanges de données
  - Diffusion globale des données **MPI_Bcast**
  - Diffusion sélective des données **MPI_Scatter**
  - Collecte des données réparties **MPI_Gather**
  - Collecte par tous les processus des données réparties **MPI_Allgather**
  - Echanges globaux **MPI_Alltoall**
- Opérations de réduction (**MPI_Reduce** et **MPI_Allreduce**)

Communications collectives

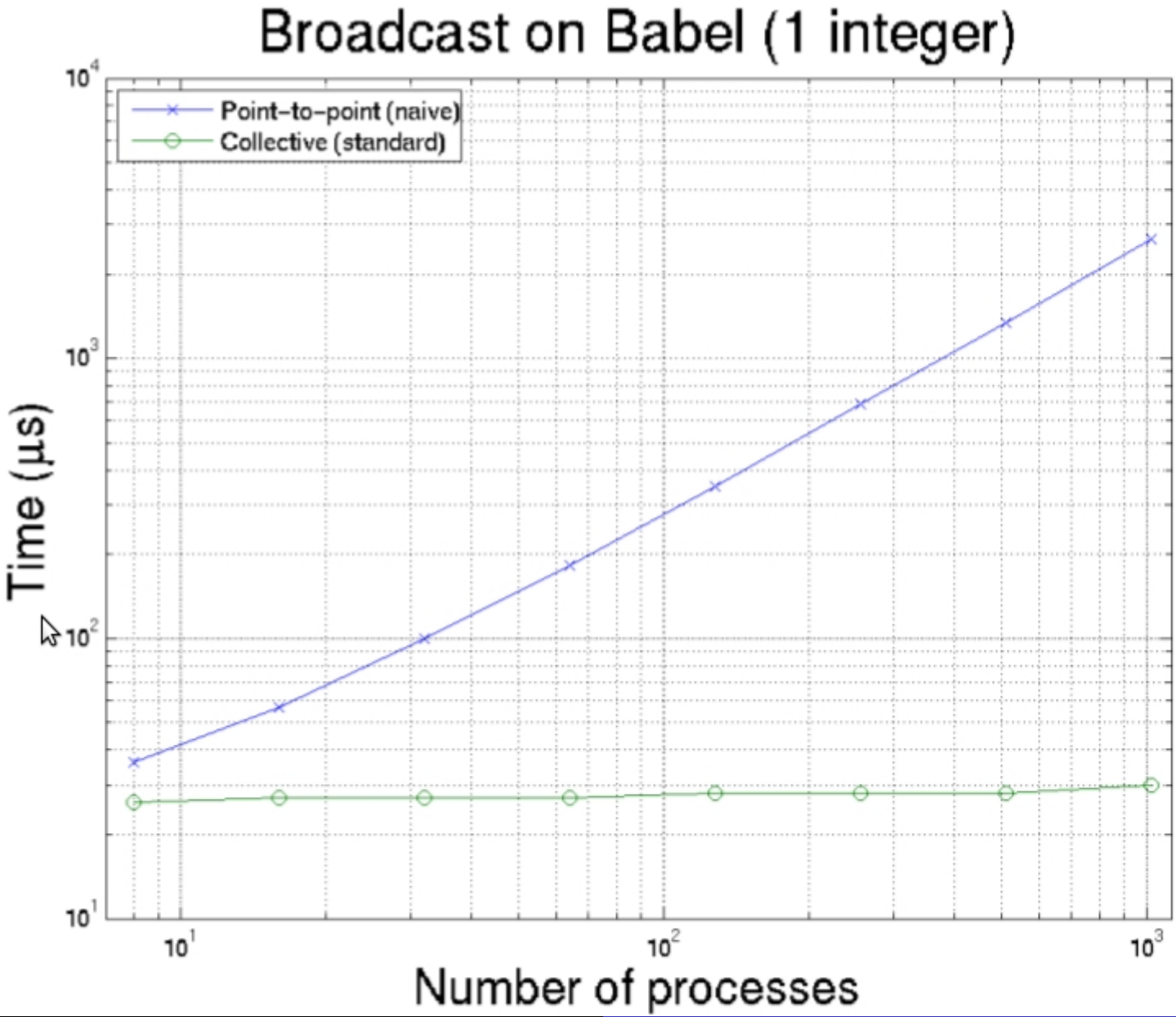
Avantages

- Les communications collectives sont fortement optimisées
- C'est l'équivalent d'une série de communications point à point en une seule opération

Inconvénients

- Peut cacher au programmeur un volume de transfert très important (par exemple MPI_Alltoall avec 1024 processus implique 1 million de messages point à point)
- Pas d'appels non bloquants (ne sera plus vrai dans la norme MPI 3.0)
- Implique tous les processus du communicateur. Il faut donc créer des sous-communicateurs si tous les processus ne sont pas concernés par une communication collective

Performances

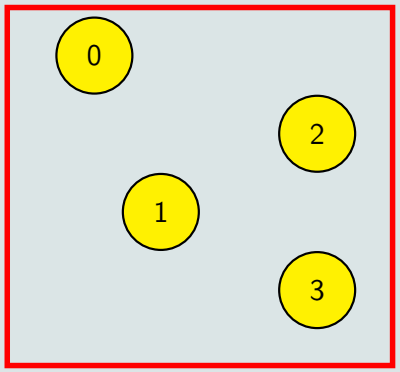


Synchronisation globale : MPI_Barrier

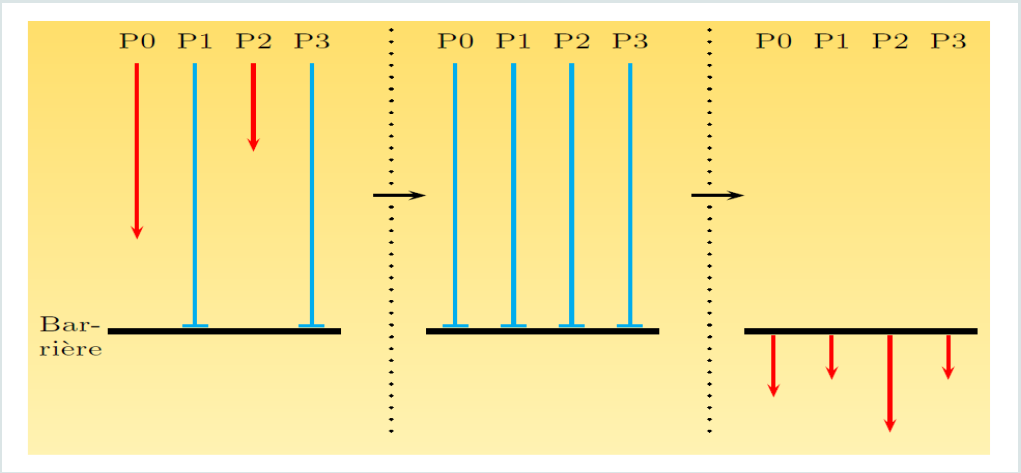
int MPI_Barrier(MPI_Comm comm)

- C'est une routine collective
- Elle permet de bloquer les processus du communicateur comm jusqu'à ce que le dernier soit arrivé à la barrière

Synchronisation globale



(a) Communicateur



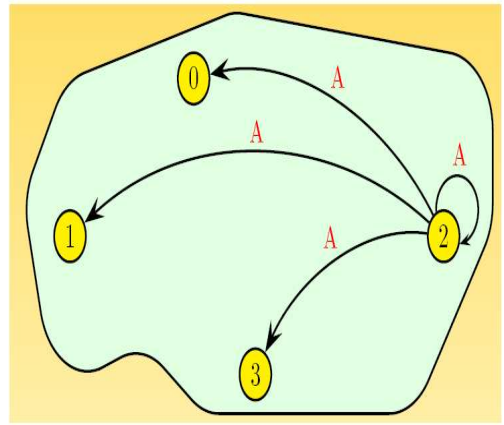
(b) Barrière

Diffusion générale : MPI_Bcast (1)

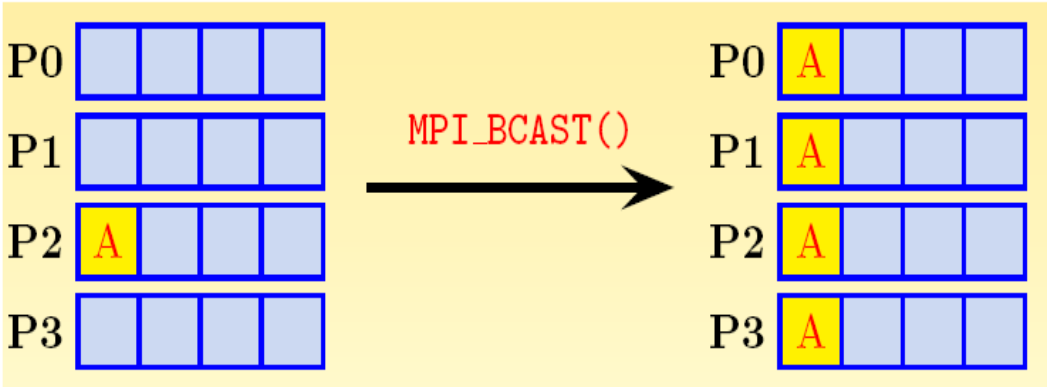
Routine MPI_Bcast

- C'est la communication de type un-vers-tous
- Cette routine permet de diffuser à tous les processus une même donnée
- Elle doit être appelée par tous les processus dans un communicateur

Processus 2 diffuse un message aux autres



(c) Communicateur



(d) Proc. 2 est la racine

Diffusion générale : MPI_Bcast (2)

Prototype

```
int MPI_Bcast( void *buffer, int count,
               MPI_Datatype datatype,
               int root, MPI_Comm comm )
```

Paramètre

1. void* **buff** : Adresse du buffer
2. int **count** : Nombre d'éléments dans le tampon de données
3. MPI_Datatype **datatype** : Type des éléments envoyés
4. int **root** : Identifiant de la racine de la communication
5. MPI_Comm **comm** : Communicateur

Diffusion générale : MPI_Bcast (3)

Prototype

```
int MPI_Bcast( void *buffer, int count,
               MPI_Datatype datatype,
               int root, MPI_Comm comm )
```

Proc. 2 diffuse un message aux autres dans MPI_COMM_WORLD

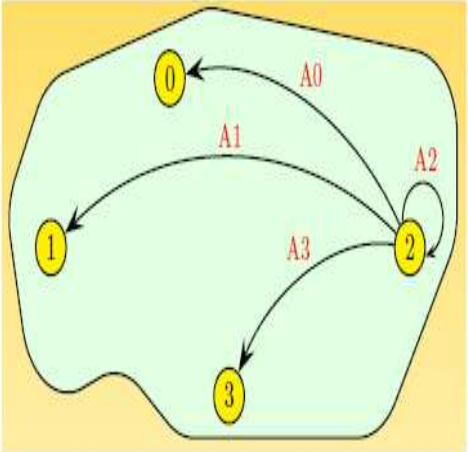
```
int buff[10], pid, nprocs, i;
MPI_Comm_rank(MPI_COMM_WORLD, &pid);
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
...
if (pid==2) for(i=0;i<10;i++) buff[i]=i;
MPI_Bcast(buff, 10, MPI_INT, 2, MPI_COMM_WORLD);
...
if (pid!=2) Afficher(buff);
```

Diffusion sélective : MPI_Scatter (1)

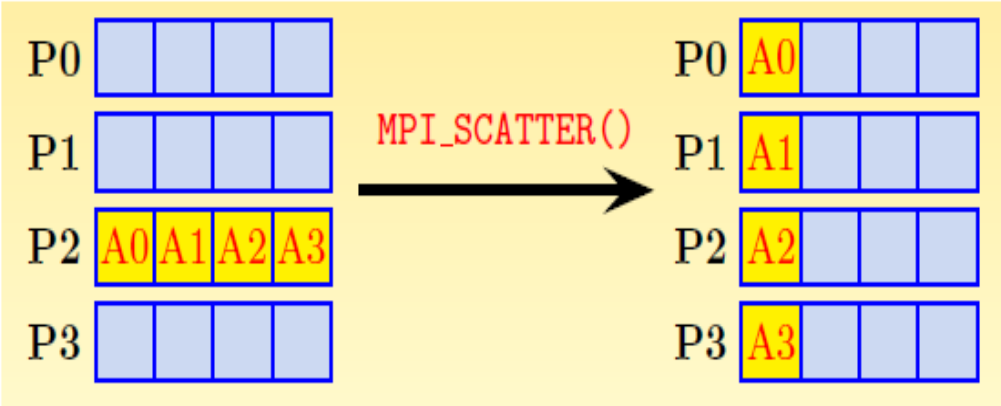
Routine MPI_Scatter

- Cette routine permet au processeur racine de répartir un message sur les processus du communicateur
- C'est une opération de type un-vers-tous, où des données différentes sont envoyées sur chaque processus receveur, suivant leur rang

Proc. 2 répartit des données aux autres processus



(e) Communicateur



(f) Proc. 2 est la racine

Diffusion sélective : MPI_Scatter (2)

Prototype

```
int MPI_Scatter(void *sendbuf,int sendcnt,
                MPI_Datatype sendtype,void *recvbuf,
                int recvcnt,MPI_Datatype recvtype,
                int root,MPI_Comm comm)
```

1. void ***sendbuf** : Adresse du tampon d'envoi
2. int **sendcount** : Nb d'éléments envoyés à chaque processus
3. MPI_Datatype **sendtype** : Type d'élément envoyé
4. void ***recvbuf** : Adresse du tampon de réception
5. int **recvcount** : Nombre d'éléments reçus
6. MPI_Datatype **recvtype** : Type de chaque élément reçu
7. int **root** : Identifiant de la racine de la communication
8. MPI_Comm **comm** : Communicateur

Diffusion sélective : MPI_Scatter (3)

Prototype

```
int MPI_Scatter(void *sendbuf,int sendcnt,
                MPI_Datatype sendtype,void *recvbuf,
                int recvcnt,MPI_Datatype recvtype,
                int root,MPI_Comm comm)
```

Notes

- Proc. root envoie au processus i sendcount éléments de type sendtype à partir de l'adresse :

  sendbuf + i * sendcount * sizeof(sendtype)

- Les données sont stockées par chaque récepteur à l'adresse recvbuf

Diffusion sélective : MPI_Scatter (4)

Prototype

```
int MPI_Scatter(void *sendbuf,int sendcnt,
                MPI_Datatype sendtype,void *recvbuf,
                int recvcnt,MPI_Datatype recvtype,
                int root,MPI_Comm comm)
```

Proc. 2 répartit des données aux autres procs.

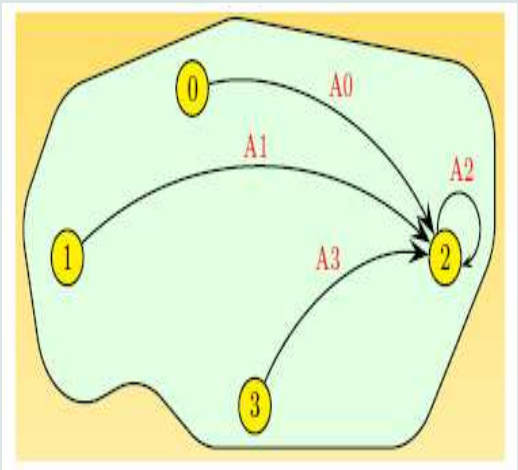
```
int root=2, *sendbuf, recvbuf[10];
if (pid==2){
    sendbuf = (int*)malloc(nprocs*10*sizeof(int));
    for(i=0;i<nprocs*10;i++) sendbuf[i]=i;
}
MPI_Scatter( sendbuf, 10, MPI_INT,
             recvbuf, 10, MPI_INT,
             root, MPI_COMM_WORLD);
if (pid!=2) Afficher(recvbuf);
```

Collecte : MPI_Gather (1)

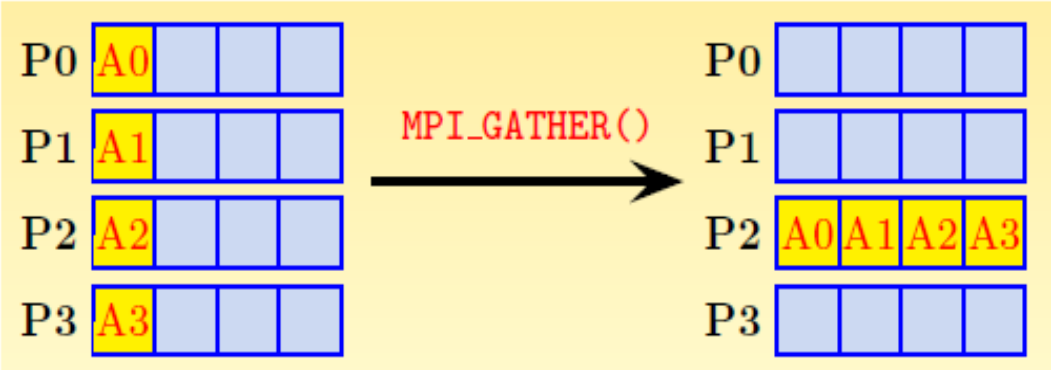
Routine MPI_Gather

- Cette fonction permet au processus racine de collecter les données provenant de tous les processus (lui y compris)
- Le résultat n'est connu que par le processus racine

Proc. 2 collecte des données depuis les autres processus



(a) Communicateur



(b) Proc. 2 est la racine

Collecte MPI_Gather (2)

Prototype

```
int MPI_Gather(void *sendbuf, int sendcnt,
               MPI_Datatype sendtype,void *recvbuf,
               int recvcnt, MPI_Datatype recvtype,
               int root, MPI_Comm comm)
```

Paramètres

1. void ***sendbuf** : Adresse du tampon d'envoi
2. int **sendcount** : Nombre d'éléments envoyés
3. MPI_Datatype **sendtype** : Type d'élément envoyé
4. void ***recvbuf** : Adresse du tampon de réception
5. int **recvcount** : Nombre d'éléments reçus
6. MPI_Datatype **recvtype** : Type d'élément reçu
7. int **root** : Identifiant du processus racine
8. MPI_Comm **comm** : communicateur

Collecte MPI_Gather (3)

Prototype

```
int MPI_Gather(void *sendbuf, int sendcnt,
               MPI_Datatype sendtype,void *recvbuf,
               int recvcnt, MPI_Datatype recvtype,
               int root, MPI_Comm comm)
```

Proc. 2 collecte des données d'autres procs.

```
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
MPI_Comm_rank(MPI_COMM_WORLD, &pid);
int nprocs,sendbuf[10], root=2, *recvbuf;
if (pid==root)
   recvbuf = (int*)malloc(nprocs*10*sizeof(int));
MPI_Gather( sendbuf, 10, MPI_INT,
            recvbuf, 10, MPI_INT,
            root, MPI_COMM_WORLD);
if (pid==root) Afficher(recvbuf);
```

Collecte générale : MPI_Allgather (1)

Prototype

```
int MPI_Allgather(void *sendbuf, int sendcount,
                  MPI_Datatype sendtype,void *recvbuf,
                  int recvcount, MPI_Datatype recvtype,
                  MPI_Comm comm)
```

Paramètres

1. void ***sendbuf** : Adresse du tampon d'envoi
2. int **sendcount** : Nombre d'éléments envoyés
3. MPI_Datatype **sendtype** : Type d'élément envoyé
4. void ***recvbuf** : Adresse du tampon de réception
5. int **recvcount** : Nombre d'éléments reçus
6. MPI_Datatype **recvtype** : Type de chaque élément reçu
7. MPI_Comm **comm** : Communicateur

Collecte générale : MPI_Allgather (2)

Prototype

```
int MPI_Allgather(void *sendbuf, int sendcount,
                  MPI_Datatype sendtype,void *recvbuf,
                  int recvcount, MPI_Datatype recvtype,
                  MPI_Comm comm)
```

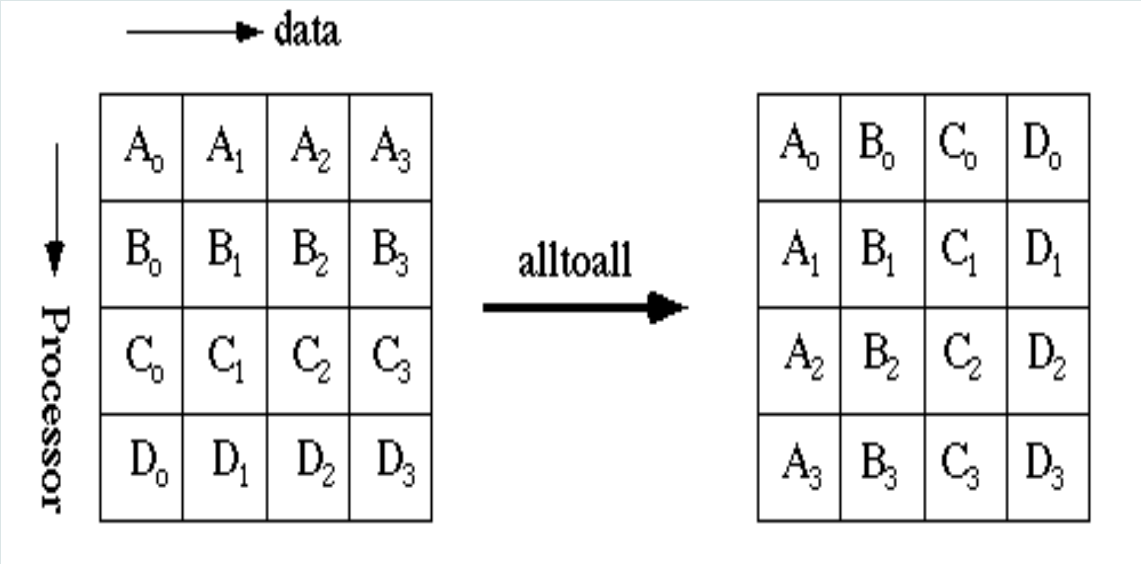
```
int pid, nprocs, sendbuf[10], *recvbuf;
MPI_Comm_rank(MPI_COMM_WORLD, &pid);
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
recvbuf = (int*)malloc(nprocs*10*sizeof(int));

MPI_Allgather( sendbuf, 10, MPI_INT,
               recvbuf, 10, MPI_INT,
               MPI_COMM_WORLD);

Afficher(recvbuf);
```

Échanges croisés : MPI_Alltoall (1)

Routine MPI_Alltoall

Envoie un message distinct de chacun des processus pour chaque processus.

Principe



Échanges croisés : MPI_Alltoall (2)

Prototype

```
int MPI_Alltoall(void *sendbuf, int sendcount,
                 MPI_Datatype sendtype,void *recvbuf,
                 int recvcount, MPI_Datatype recvtype,
                 MPI_Comm comm)
```

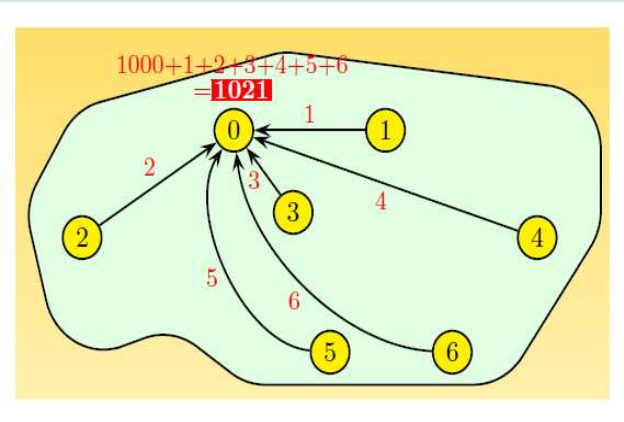
Paramètres

1. void ***sendbuf** : Adresse du tampon d'envoi
2. int **sendcount** : Nombre d'éléments envoyés
3. MPI_Datatype **sendtype** : Type d'élément
4. void ***recvbuf** : Adresse du tampon de réception
5. int **recvcount** : Nombre d'élément reçus
6. MPI_Datatype **recvtype** : Type de chaque élément reçu
7. MPI_Comm **comm** : Communicateur

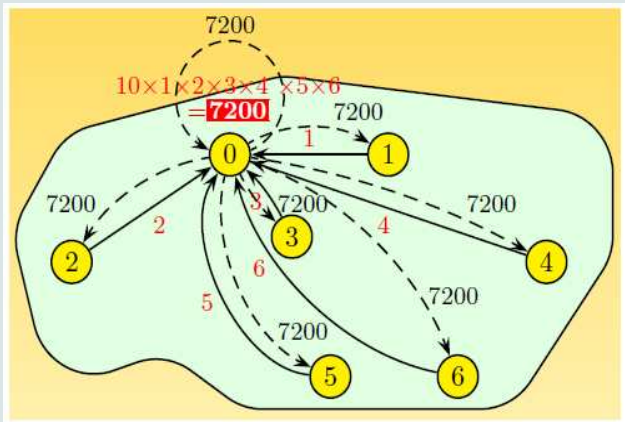
Réductions : MPI_Reduce et MPI_Allreduce (1)

- La réduction est une opération appliquée aux données réparties sur un ensemble de processus pour n'obtenir qu'une seule valeur sur
  - un seul processus : **MPI_Reduce**
  - tous les processus : **MPI_Allreduce** (MPI_Reduce suivi d'un MPI_Bcast)
- Par exemple : Cette valeur peut être la somme des éléments ou la valeur maximale des éléments

Exemple : Réductions



(c) MPI_Reduce(somme)



(d) MPI_Allreduce(produit)

Réductions réparties : MPI_Reduce et MPI_Allreduce (2)

Tableau d'opérations principales

Nom	Opération
MPI_SUM	Somme des éléments
MPI_PROD	Produit des éléments
MPI_MAX	Recherche du maximum
MPI_MIN	Recherche du minimum
MPI_MAXLOC	Recherche de l'indice du maximum
MPI_MINLOC	Recherche de l'indice du minimum
MPI_LAND	ET logique
MPI_LOR	OU logique
MPI_LXOR	XOR logique

Routine MPI_Reduce (1)

Prototype

```
int MPI_Reduce(void *sendbuf, void *recvbuf,
               int count,MPI_Datatype datatype,
               MPI_Op op,int root,MPI_Comm comm)
```

Paramètres

1. void ***sendbuf** : Adresse du tampon d'envoi
2. void ***recvbuf** : Adresse du tampon de réception
3. int **count** : Nombre d'éléments envoyés
4. MPI_Datatype **datatype** : Type des éléments
5. MPI_Op **op** Opération réalisée sur des données envoyées
6. int **root** : Identifiant de la racine de la communication
7. MPI_Comm **comm** : Communicateur

Routine MPI_Reduce (2)

Exemple

```
int pid, nprocs, sent=0, recv=0;
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
MPI_Comm_rank(MPI_COMM_WORLD, &pid);
if (pid==0) sent=1000;
else sent=pid;
MPI_Reduce(&sent, &recv, 1,
           MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
if (pid==0)printf("recv = %d", recv);
```

Résultat à Proc. 0 avec 7 processus

```
    value of recv = 1021
```

Routine MPI_Allreduce (1)

Prototype

```
int MPI_Allreduce (void *sendbuf,void *recvbuf,
                   int count,MPI_Datatype datatype,
                   MPI_Op op, MPI_Comm comm)
```

Paramètres

1. void ***sendbuf** : Adresse du tampon d'envoi
2. void ***recvbuf** : Adresse du tampon de réception
3. int **count** : Nombre d'éléments envoyés
4. MPI_Datatype **datatype** : Type des éléments
5. MPI_Op **op** : Opération réalisée sur des données envoyées
6. MPI_Comm **comm** : Communicateur

Routine MPI_Allreduce (2)

Exemple

```
int pid, nprocs, sent=0, recv=0;
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
MPI_Comm_rank(MPI_COMM_WORLD, &pid);
if (pid==0) sent=10;
else sent=pid;
MPI_Allreduce(&sent, &recv, 1,
              MPI_INT, MPI_PROD, MPI_COMM_WORLD);
printf("value of recv = %d", recv);
```

Résultat à Proc. 0 avec 7 processus

```
    value of recv = 7200
```