Programmation Parallèle

Le calcul haute performance
Architecture mémoire distribuée
MPI les communications collectives - le calcul stencil

Sophie Robert

UFR ST Département info

MPI pour la programmation par passage de messages

Cours précédent

- SPMD : un programme exécuté par tous les processus
- pid/nprocs : pour différentier les instructions exécutées par chacun
- une bibliothèque de fonctions de communications

A suivre

- Les fonctions de communications collectives
- Les communicateurs et les topologies

Type de communications collectives MPI

Les communications collectives peuvent être séparées en 3 catégories

- Une synchronisation globale (**MPI_Barrier**)
- 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

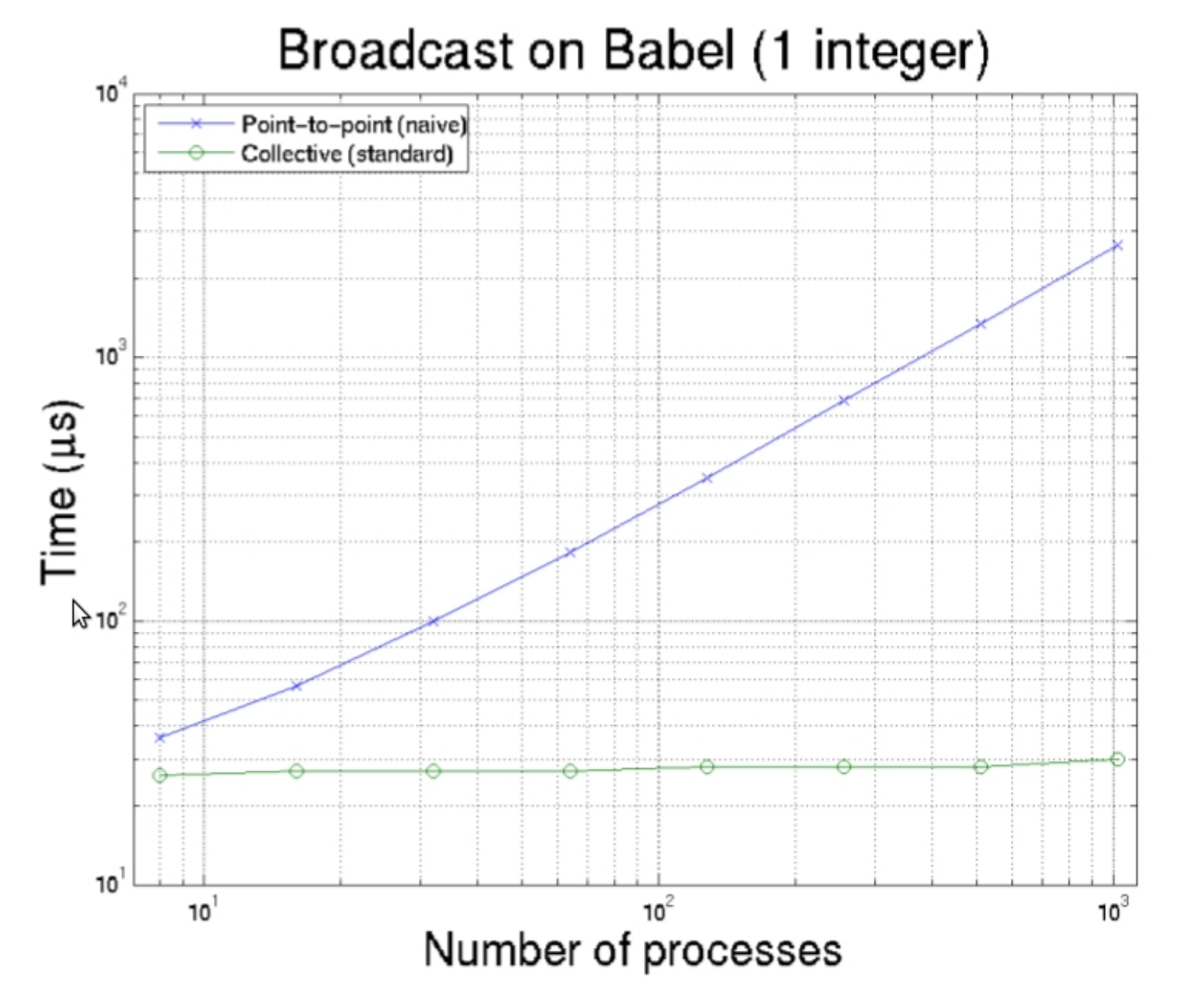
Avantages

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

Autres caractéristiques

- Elles peuvent cacher au programmeur un volume de transfert très important 
  * MPI_Alltoall avec 1024 processus implique 1 million de messages point-à-point
- Elles impliquent **tous les processus du communicateur**.

Performances

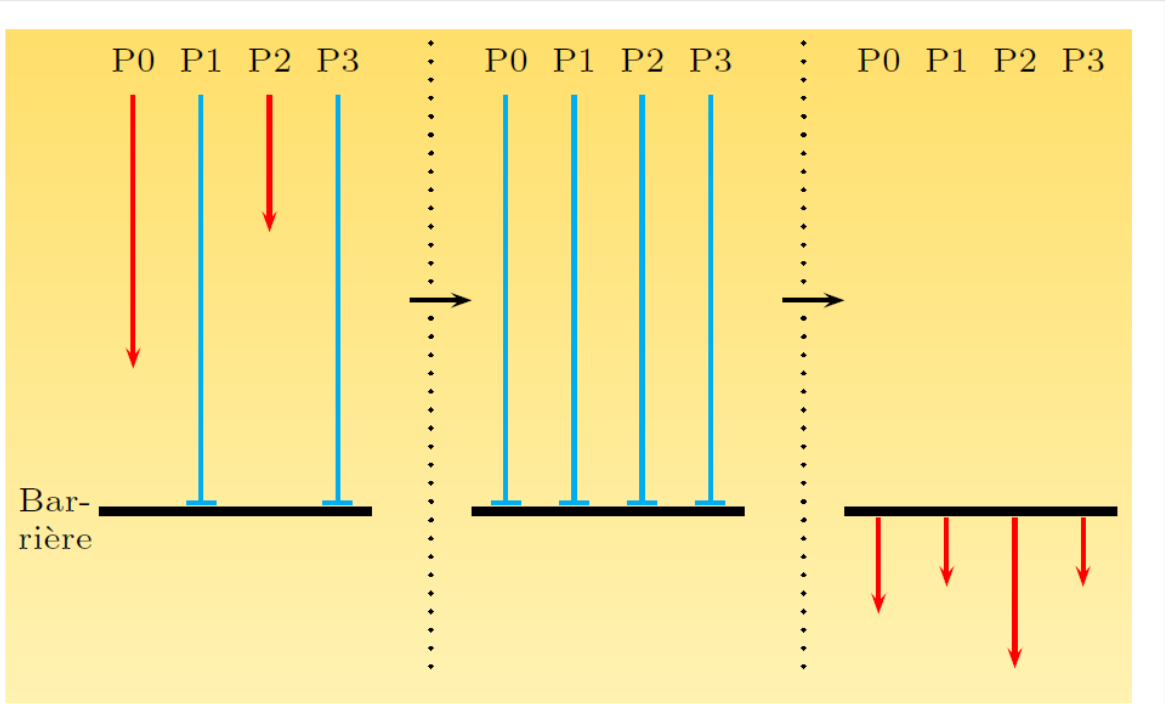


Synchronisation globale: MPI_Barrier

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



Le processus `root`

Un processus de référence

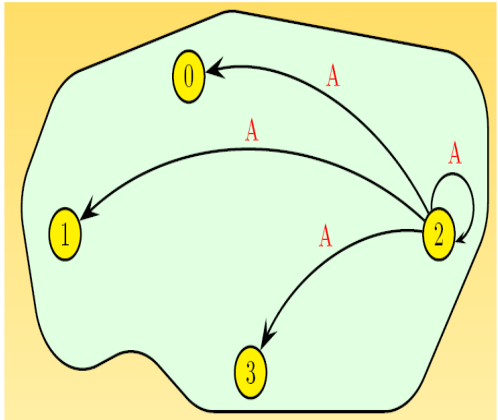
- Pour les communications un-vers-tous ou tous-vers-un un processus joue un rôle particulier
- Ce processus appelé `root` est donné en argument en ligne de commande
- Il sera systématiquement un paramètre de la routine de communication collective
-  le programme doit marcher quelque soit l'identifiant de ce processus `root`

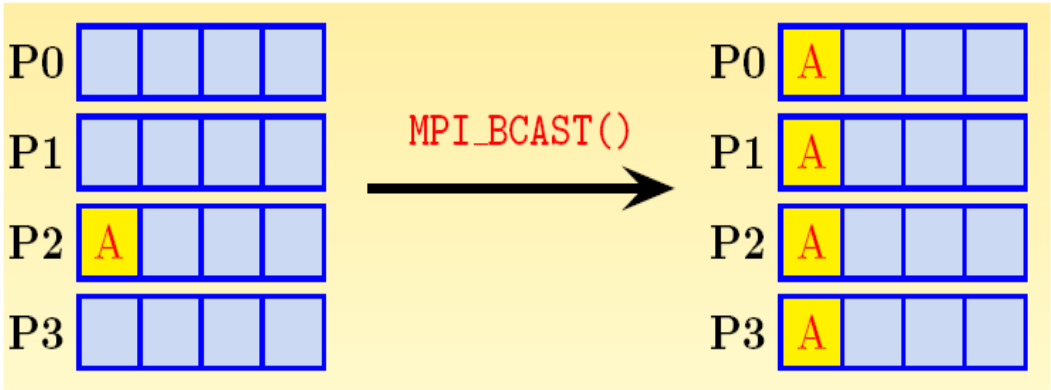
Diffusion générale: MPI_Bcast

Routine MPI_Bcast

- C'est une 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

Le processus 2 diffuse un message aux autres





Diffusion générale: MPI_Bcast

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

Prototype

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

root 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==root) for(i=0;i<10;i++)
    buff[i]=i;
MPI_Bcast(buff, 10, MPI_INT, root, MPI_COMM_WORLD);
          ...
Afficher(buff);
```

La variable `buf` pour la communication collective

Déclaration - Initialisation - Après la communication

- Tous les processus déclarent `buf`
- Avant la diffusion seul le processus `root` a initialisé `buf`
- Après la diffusion tous les processus ont les mêmes données dans `buf`

Schéma de communication pour un MPI_Bcast

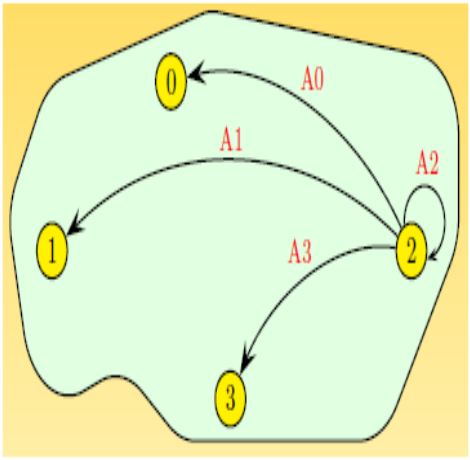
- Quelle est la complexité d'une diffusion de un-vers-tous ?

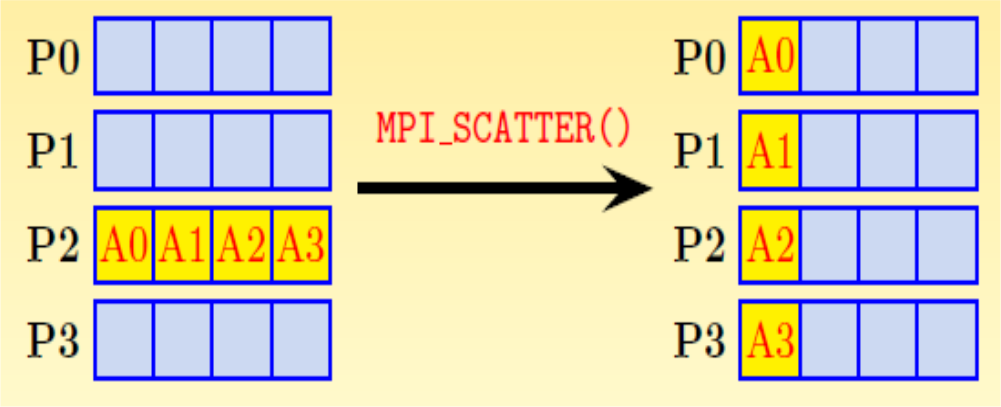
Diffusion sélective: MPI_Scatter

Routine MPI_Scatter

- Cette routine permet au processus `root` 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, suivant leur rang

Le processus 2 répartit des données aux autres processus





Gestion par des communications point-à-point

```
// n et n_local
int* tab = new int[n];
if (pid==root)
    for (int i=0; i<n; i++) tab[i] = ...; // tab a des
    valeurs que sur root
int* tab_recu = new int[n_local];
if (pid==root)
  for (int i=0; i<nprocs; i++)
     if (i!=root)
       MPI_Ssend(tab+i*n_local,n_local,MPI_INT,i,tag,
    MPI_COMM_WORLD);
     else
       for (int j=0; j<n_local; j++)
         tab_recu[j] = tab[n_local*root+j];
else
  MPI_Recv(tab_recu,n_local,MPI_INT,root,tag,
    MPI_COMM_WORLD,MPI_STATUS_IGNORE);
```

Diffusion sélective: MPI_Scatter

Prototype

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

Paramètres

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

Diffusion sélective: MPI_Scatter

Prototype

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

Paramètres

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

Prototype

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

Interprétation

- Le processus `root` envoie au processus `i` sendcount éléments de type sendtype à partir de l'adresse sendbuf + i * sendcount
- Les données sont stockées par chaque récepteur à l'adresse recvbuf

Diffusion sélective: MPI_Scatter

```
int* tab_recu2 = new int[n_local];
MPI_Scatter(tab,n_local,MPI_INT,tab_recu2,n_local,
  MPI_INT,root,MPI_COMM_WORLD);

cout << "tab_recu de " << pid << " : ";
for (int i=0; i<n_local; i++)
  cout << tab_recu2[i] << " ";
cout << endl;
```

MPI_Scatter -> MPI_Scatterv

Une répartition régulière

MPI_Scatter uniquement sur des données dont la taille est divisible par le nombre de processus

MPI_Scatter -> MPI_Scatterv

Gestion avec des communications point-à-point

```
int n_local = n/nprocs;
int reste = n%nprocs;
if (pid<reste)
  n_local++;
int* tab_r = new int[n_local];
```

MPI_Scatter -> MPI_Scatterv

Gestion avec des communications point-à-point

```
if (pid==root) { //le tableau tab (taille n) est alloue et initialise
   sur root
  int ptr = 0;
  int n1, n2 = n/nprocs;
  if (n%nprocs!=0)
    n1 = n2+1;
  else
    n1=n2;
  for (int i=0; i<nprocs; i++) {
    if (i==root)
      for (int j=0; j<n_local; j++)
          tab_r[j] = tab[ptr+j];
    else if (i<reste)
          MPI_Ssend(tab+ptr,n1,MPI_INT,i,tag,MPI_COMM_WORLD);
    else
          MPI_Ssend(tab+ptr,n2,MPI_INT,i,tag,MPI_COMM_WORLD);
    if (i<reste)
      ptr += n1;
    else
      ptr+=n2;
  }
}
else
 MPI_Recv(tab_r,n_local,MPI_INT,root,tag,MPI_COMM_WORLD,
   MPI_STATUS_IGNORE);
```

MPI_Scatter -> MPI_Scatterv

Prototype

```
int MPI_Scatterv(void *sendbuf, int* sendcounts,
    int* displs, MPI_Datatype sendtype, void *recvbuf,
    int recvcount, MPI_Datatype recvtype, int root,
    MPI_Comm comm)
```

Les nouveaux paramètres

1. int* **sendcounts**: Nombre d'éléments à envoyer par processus
2. int* **displs**: Déplacement dans le buffer d'envoi par processus

Gestion de la taille non divisible

Construction du pointeur et de la taille

- Soit un tableau de `20` entiers alloué et initialisé sur le processus `root`
- Soit une exécution parallèle sur `nprocs=6`

pid	0	1	2	3	4	5
sendcounts						
displs						

MPI_Scatter -> MPI_Scatterv

Gestion avec la routine Scatterv

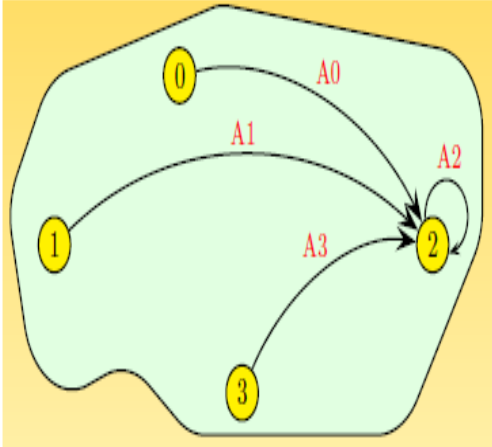
```
int* sendcounts;
int* displ;
int n_local = n/nprocs;
int reste = n%nprocs;
if (pid==root) {
    sendcounts = new int[nprocs];
    displ = new int[nprocs];
    int ptr = 0;
    for (int i=0; i<reste; i++) {
        sendcounts[i] = n_local+1;
        displ[i] = ptr;
        ptr+=(n_local+1);
    }
    for (int i=reste; i<nprocs; i++) {
        sendcounts[i] = n_local;
        displ[i] = ptr;
        ptr+=n_local;
    }
}
if (pid<reste)
    n_local++;
int* tab_r = new int[n_local];
MPI_Scatterv(tab,sendcounts,displ,MPI_INT,tab_r,n_local,MPI_INT,root,
    MPI_COMM_WORLD);
```

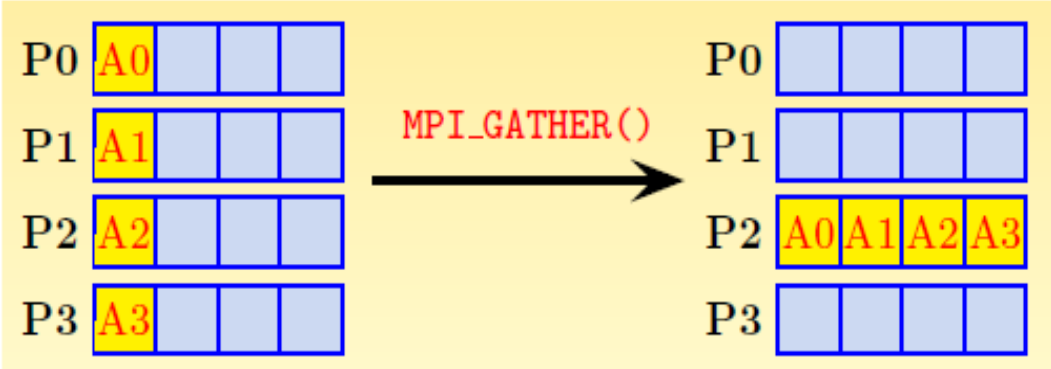
Rassemblement MPI_Gather

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 root
- C'est une communication de type tous-vers-un

Le processus 2 collecte des données depuis les autres processus





Collecte MPI_Gather

Prototype

```
int MPI_Gather(void *sendbuf, int sendcount,
               MPI_Datatype sendtype,void *recvbuf,
               int recvcount, 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

Collecte MPI_Gather

Prototype

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

Paramètres

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

Le processus 2 collecte des données des autres processus.

```
// Chaque processus a un tableau sendbuf de n_local
    entiers
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
MPI_Comm_rank(MPI_COMM_WORLD, &pid);
int *recvbuf;
if (pid==root)
    recvbuf = new int[n_local*nprocs];

MPI_Gather(sendbuf, n_local, MPI_INT, recvbuf, n_local,
    MPI_INT, root, MPI_COMM_WORLD);

if (pid==root)
    Afficher(recvbuf);
```

MPI_Gather -> MPI_Gatherv

Une répartition régulière

On rassemble toujours *nprocs* $\times$ *taille_locale* données sur un processus

Sinon :

```
int MPI_Gatherv(const void *sendbuf, int sendcount,
    MPI_Datatype sendtype, void *recvbuf,
    int* recvcounts, int* displs,
    MPI_Datatype recvtype, int root, MPI_Comm comm)
```

Les nouveaux paramètres

1. int ***recvcounts**: Nombre d'éléments reçus par processus
2. int ***displs**: Déplacement dans le buffer de réception

MPI_Gatherv exemple

Rassemblement d'un tableau de taille quelconque

```
n_local=n/nprocs
reste=n%nprocs
if (pid==root) {
     recvcounts = new int[nprocs];
     displs = new int[nprocs];
     int ptr = 0;
     for (int i=0; i<reste; i++) {
       recvcounts[i] = n_local+1; displs[i] = ptr;
       ptr+= n_local+1; }
     for (int i=reste; i<nprocs; i++) {
       recvcounts[i] = n_local;
       displs[i] = ptr;
       ptr+= nlocal;}
}
if (pid<reste)
   n_local+=1;
MPI_Gatherv((void*)sendbuf, n_local, MPI_INT,
            (void*)recvbuf, recvcounts, displs,
            MPI_INT, 0,MPI_COMM_WORLD);
```

Collecte générale: MPI_Allgather

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

Collecte générale: MPI_Allgather

Prototype

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

Paramètres

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

Répartition irrégulière

Prototype MPI_Allgatherv

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

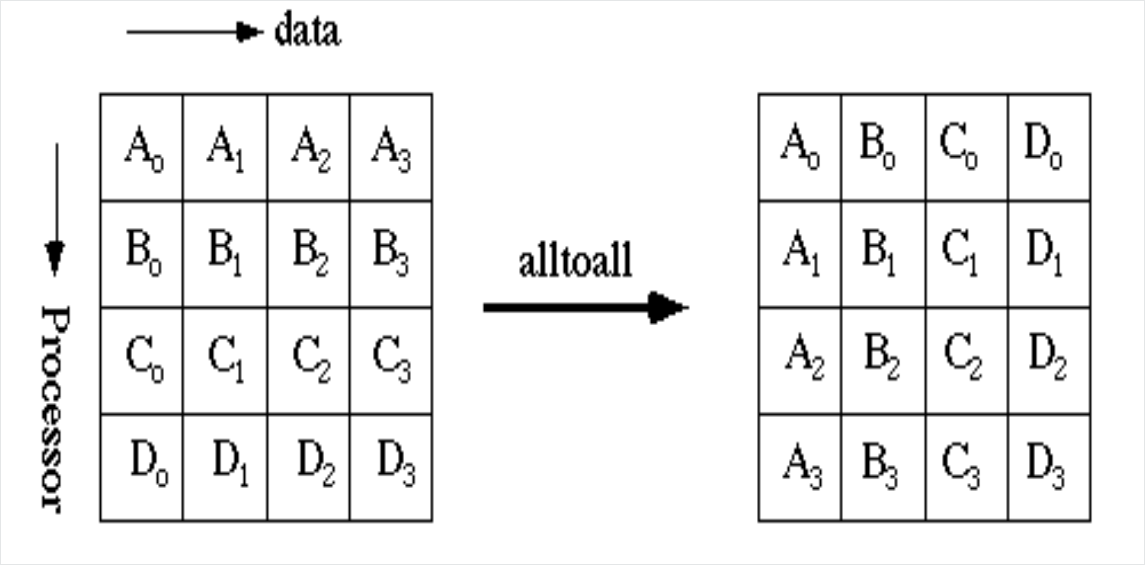
-  Tous les processus doivent définir `recvcount` et `displs`

Échanges croisés: MPI_Alltoall

Routine MPI_Alltoall

Envoie un message distinct de tous les processus pour chaque processus.

Principe



Échanges croisés: MPI_Alltoall

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

Échanges croisés: MPI_Alltoall

Prototype

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

Paramètres

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

Répartition irrégulière

Prototype MPI_Alltoallv

```
int MPI_Alltoallv(void *sendbuf, int* sendcounts,
    int* sdispls, MPI_Datatype sendtype,
    void *recvbuf, int* recvcounts,
    int *rdispls, MPI_Datatype recvtype, MPI_Comm comm)
```

MPI_Alltoallv

Exemple

P_0	83	86	77	15	93
P_1	46	85	68	40	25
P_2	75	65	10	72	76
P_3	77	99	99	71	25

$\rightarrow$

P_0	
P_1	
P_2	
P_3	

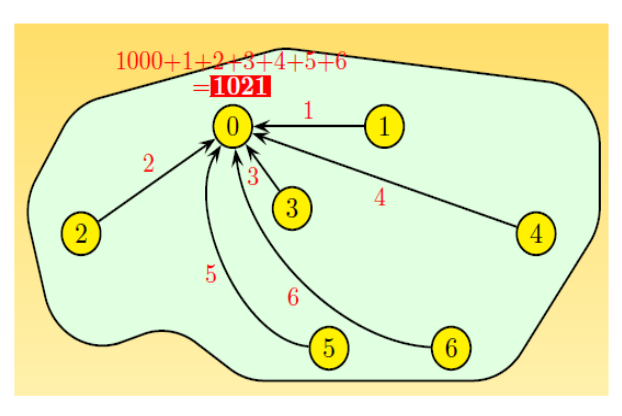
Les paramètres de la routine

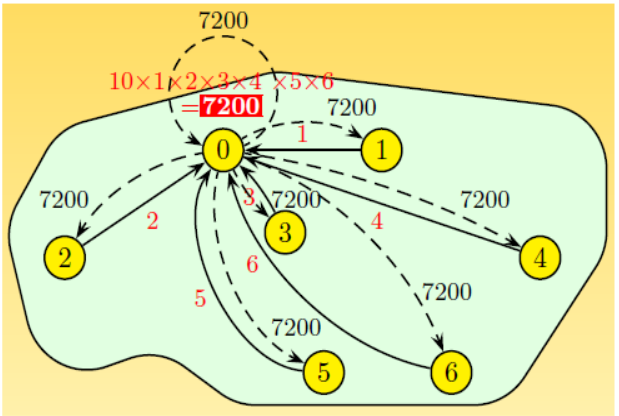
pid	0	1	2	3
sendcounts				
sdispls				
recvcounts				

Réductions : MPI_Reduce et MPI_Allreduce

Opération classique

- 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)





MPI_Reduce et MPI_Allreduce

Tableau des 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

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

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 sur le processus 0 avec 7 processus

```
value of recv = 1021
```

Routine MPI_Allreduce

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

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 sur tous les processus avec 7 processus

```
value of recv = 7200
```

Réduction sur des tableaux

Exemple

	P_0	P_1	P_2
tab	0 1 2	1 2 3	2 3 4

```
MPI_Allreduce(tab,res,3,MPI_INT,MPI_SUM,MPI_COMM_WORLD)
    ;
```

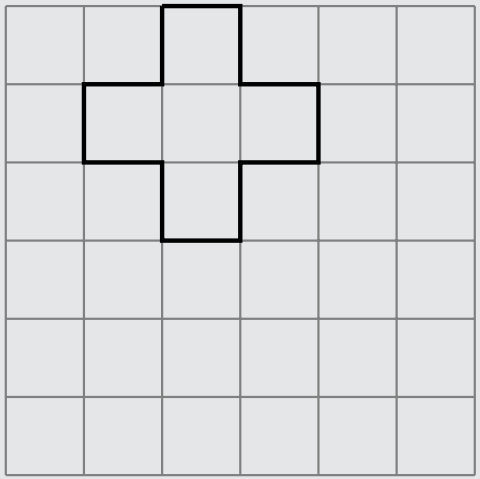
Résultat

	P_0	P_1	P_2
res			

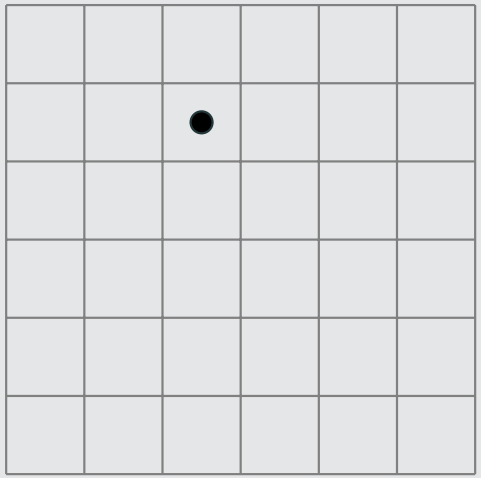
Le calcul stencil : Introduction

Sur l'exemple du jeu de la vie

cour=



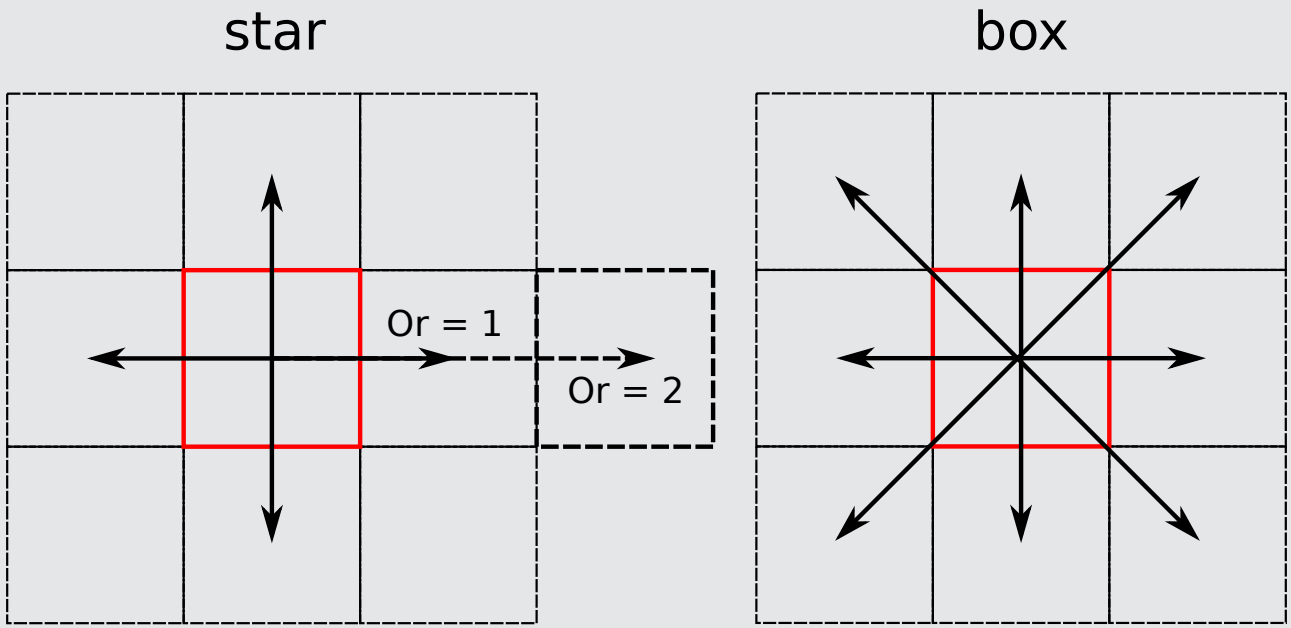
suiv=



Le calcul stencil : Introduction

De manière générale

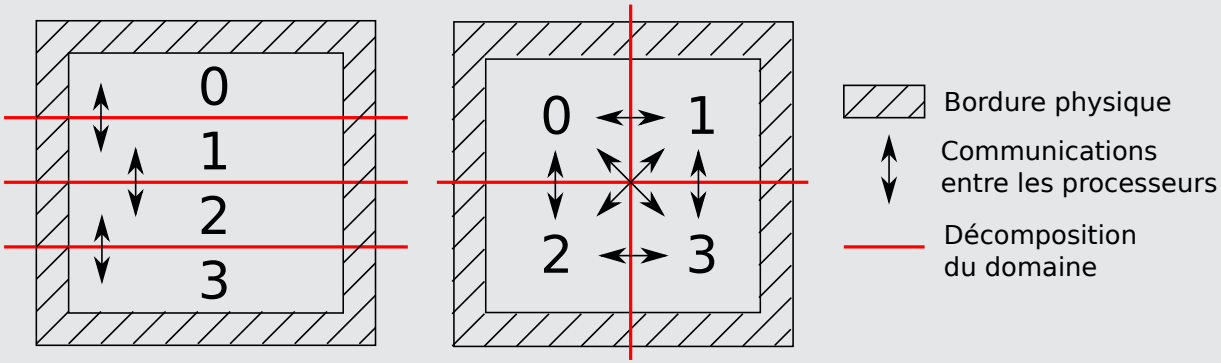
- Il existe plusieurs types de voisinages



- La grille peut avoir plus de 2 dimensions
- Les conditions en bordure du domaine peuvent engendrer des calculs différents

Parallélisation d'un calcul stencil

Décomposition du domaine



- Vers une distribution des données sur les processus
- Cette décomposition va dépendre du calcul à effectuer et de la taille des données
- En fonction du voisinage, il faut définir les bordures internes (les ghosts) qu'il faudra communiquer.