Programmation Parallèle pour les architectures à Mémoire Distribuée

Sophie Robert

Pôle info

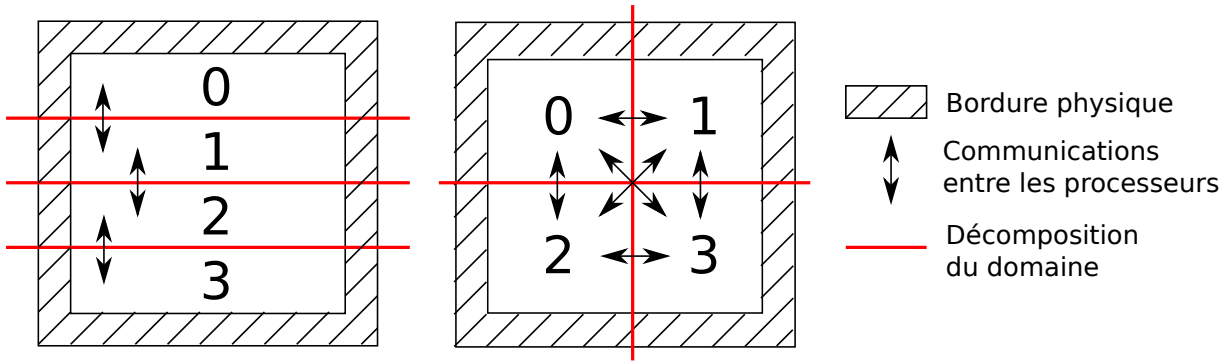
Introduction

Topologies Cartésiennes

- De nombreux calculs parallèles sont effectués sur des matrices/grilles régulières
  - Calculs sur des maillages réguliers
  - Calculs stencil
- Agencement des processus dans une grille régulière
- Simplification du calcul du voisinage

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.

Parallélisation d'un calcul stencil et topologie

Lien avec les processus

- Pour faciliter la mise en œuvre on aimerait lier la décomposition du domaine avec l'organisation des processus.
- Définir des organisations logiques des processus pour faciliter l'écriture du code et optimiser les communications.

Plan de la suite

- Les communicateurs : comment créer des groupes de processus
- Les topologies

Les communicateurs

Principe

Il s'agit de partitionner un ensemble de processus MPI afin de créer des sous-ensembles sur lesquels on peut effectuer des opérations (communications, calculs). Chaque sous-ensemble ainsi créé aura son propre espace de communication.

- On crée un communicateur à partir d'un autre
- MPI_COMM_WORLD est créé par MPI_Init et détruit par MPI_Finalize
- MPI_COMM_WORLD est le père de tous.

Groupes et communicateurs

Un communicateur est constitué

- d'un groupe, qui est un ensemble ordonné de processus
- d'un contexte de communication mis en place à l'appel du sous-programme de construction du communicateur et qui permet de délimiter l'espace de communication
- Les contextes de communication sont gérés par MPI

Les routines MPI

- Il existe diverses routines pour construire des communicateurs : **MPI_Cart_create**, **MPI_Comm_split**,**MPI_Cart_sub**.
- Les constructeurs de communicateurs sont des opérateurs collectifs qui engendrent des communications entre les processus
- Les communicateurs peuvent être supprimés avec **MPI_Comm_free**

MPI_Comm_split

MPI_Comm_split permet de partitionner un communicateur en autant de communicateurs que l'on veut.

```
int MPI_Comm_split(MPI_Comm comm, int color, int key,
                   MPI_Comm *newcomm)
```

Arguments

- MPI_Comm **comm** : le communicateur à partir duquel on partitionne les processus
- int **color** : la couleur du processus
- int **key** : la clé du processus
- MPI_Comm ***newcomm** : pointeur sur le nouveau communicateur obtenu

MPI_Comm_split

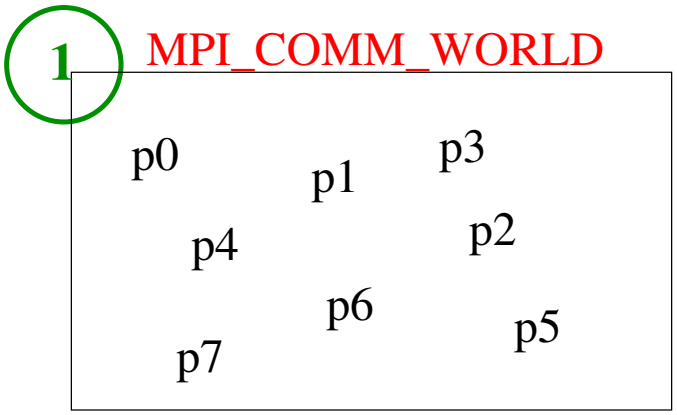
La couleur

Le paramètre **color** correspond à une règle permettant de créer les groupes de processus. Tous les processus de même couleur seront dans le même communicateur.

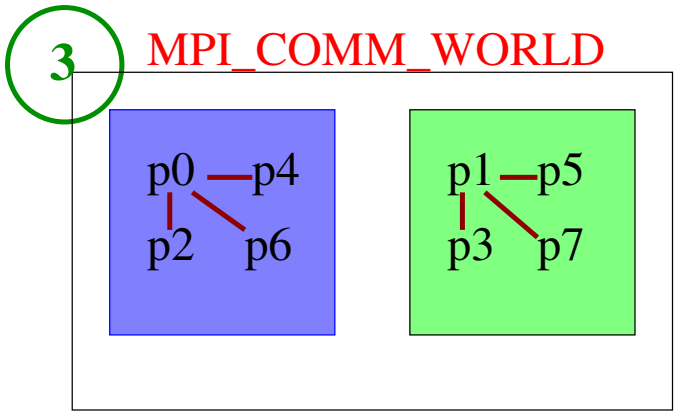
La clé

Le paramètre **key** sera utilisé pour obtenir le nouvel identifiant du processus dans le nouveau communicateur

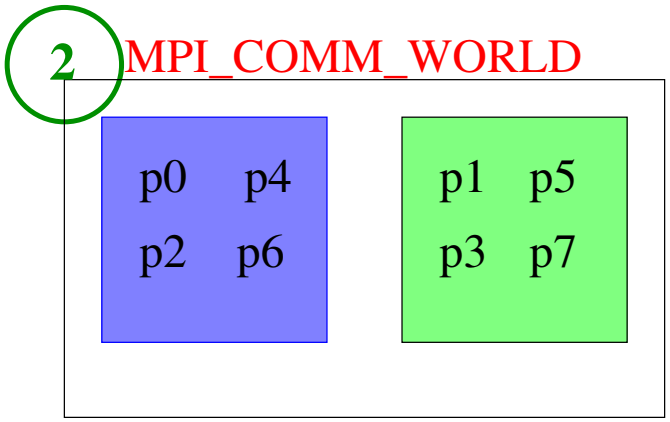
Couleur = parité du rang du processus



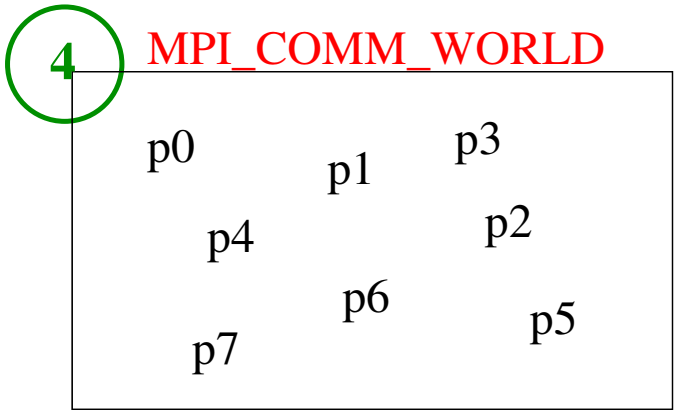
MPI_Init



MPI_Bcast



MPI_Comm_split



MPI_Comm_free

Exemple suite

```
int pid, newpid;
MPI_Comm newcomm;
MPI_Init (&argc , &argv) ;
MPI_Comm_rank(MPI_COMM_WORLD, &pid ) ;
int val = pid;
int color = (pid%2==0);
int key = 0;

MPI_Comm_split(MPI_COMM_WORLD, color , key ,&newcomm);
MPI_Comm_rank(newcomm, &newpid ) ;

MPI_Bcast(&val ,1 ,MPI_INT,0 ,newcomm);

MPI_Finalize() ;
```

Topologies de processus

Organisation des processus

- Dans la plupart des applications, plus particulièrement dans les méthodes de décomposition de domaine, on fait correspondre le domaine de calcul à la grille de processus. Dans ce cadre, il est intéressant de pouvoir disposer les processus suivant une topologie régulière.
- MPI permet de définir des topologies virtuelles de type cartésien ou graphe
- Topologie cartésienne
  * chaque processus est défini dans une grille de processus
  * la grille peut être périodique ou non
  * les processus sont identifiés par leurs coordonnées dans la grille
- Topologie graphe : généralisation à des topologies plus complexes

Topologies de type cartésien

Principe

- La grille de processus est définie par
  - Sa dimension et sa périodicité
  - Le nombre de processus dans chaque dimension

Les routines MPI permettent de

1. Créer une topologie cartésienne
2. Créer les bonnes tailles des dimensions suivant le nombre de processus
3. Définir le rang d'un processus dans une topologie cartésienne
4. Définir les coordonnées (x,y,..) d'un processus dans la topologie
5. **Partitionner un communicateur de topologie cartésienne en sous-groupes**

Routine MPI_Cart_create

Principe

- Cette routine permet de créer une topologie cartésienne
- La routine est **collective**, elle concerne donc l'ensemble des processus appartenant à l'ancien communicateur

```
int MPI_Cart_create(MPI_Comm comm_old, int ndims,
                    int *dims, int *periods,
                    int reorder, MPI_Comm *comm_cart)
```

Routine MPI_Cart_create

```
int MPI_Cart_create(MPI_Comm comm_old, int ndims,
                    int *dims, int *periods,
                    int reorder, MPI_Comm *comm_cart)
```

Paramètres

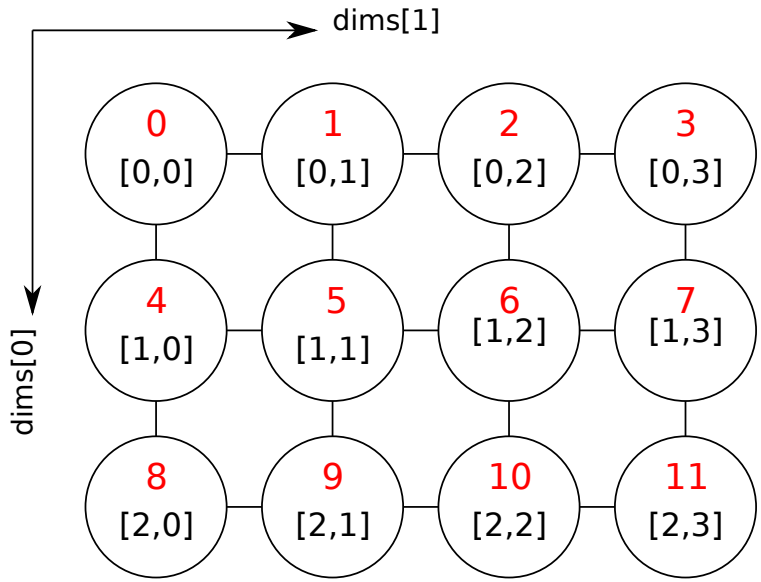
1. MPI_Comm **comm_old** : le communicateur père
2. int **ndims** : le nombre de dimensions
3. int ***dims** : le nombre de processus dans chaque dimension
4. int ***periods** : le tableau indiquant la périodicité pour chaque dimension
5. int **reorder** : le rang des processus peut-il être modifié ou non
6. MPI_Comm ***comm_cart** : le nouveau communicateur avec la structure cartésienne

Routine MPI_Cart_create

```
int ndims=2, dims[2], periods[2], reorder;
dims[0]=3; dims[1]=4;
periods[0]=????; periods[1]=????;
reorder=TRUE;
MPI_Comm comm_cart;
MPI_Cart_create(MPI_COMM_WORLD, ndims, dims,
                periods, reorder, &comm_cart);
```

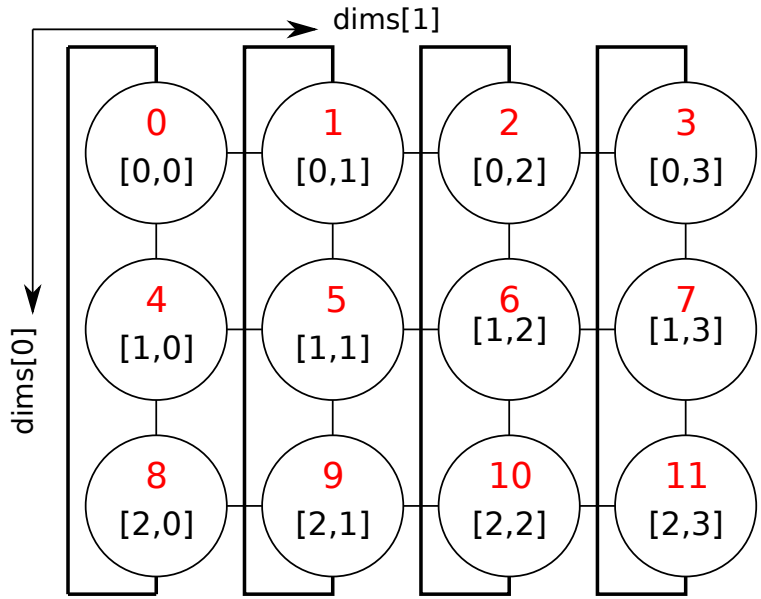
Routine MPI_Cart_create

Topologie cartésienne en 2 dimensions et non périodique

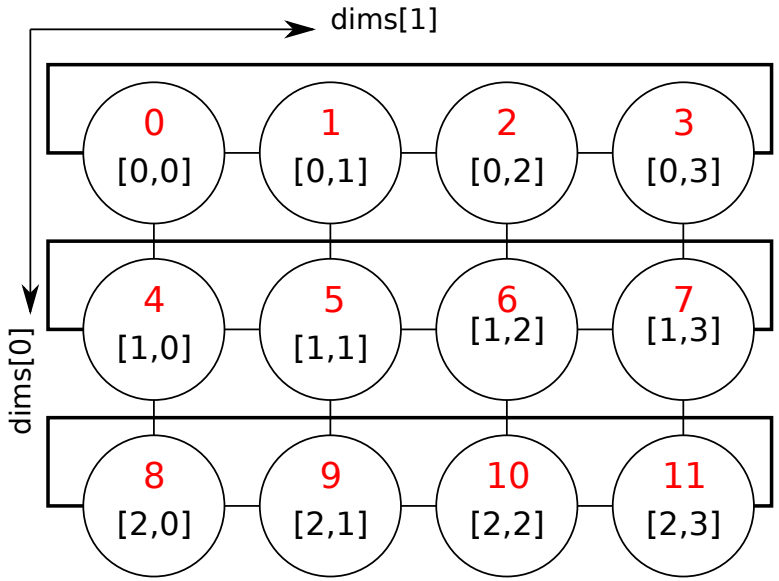


Routine MPI_Cart_create

Topologie cartésienne en 2 dimensions



périodique sur l'axe 0



périodique sur l'axe 1

Routine MPI_Cart_create

mpirun -np ??

-  le paramètre -np doit être compatible avec le nombre de processus de l'exécution parallèle.

Routine MPI_Dims_create

Prototype

Cette routine permet d'obtenir une répartition automatique et idéale des processus suivant le nombre de dimensions souhaitées.

```
int MPI_Dims_create(int nnodes, int ndims, int *dims)
```

Paramètres

1. int **nnodes** : le nombre de processus dans la grille
2. int **ndims** : le nombre de dimensions souhaitées
3. int ***dims** : le nombre de processus par dimension obtenu

Routine MPI_Dims_create

Création avec répartition automatique des processus

```
int nb_procs;
MPI_Comm_size(MPI_COMM_WORLD,&nb_procs);
int ndims=2, periods[2], dims[2];
int reorder=TRUE;
periods[0]=FALSE; periods[1]=FALSE;
dims[0]=dims[1]=0; // ne pas oublier d'initialiser

MPI_Dims_create(nb_procs,ndims,dims);
MPI_Cart_create(MPI_COMM_WORLD, ndims, dims,
                periods, reorder, &new_comm);
```

Lien entre rang et coordonnées

Pour les communications point-à-point

- Le seul identifiant reconnu est le numéro du processus dans le communicateur.
- `MPI_Cart_rank` des coordonnées au rang (pid dans MPI_COMM_WORLD)
- `MPI_Cart_coords` du rang (pid dans MPI_COMM_WORLD) aux coordonnées dans la topologie

Routine MPI_Cart_rank

Cette routine permet de connaître le rang du processus associé aux coordonnées données.

```
int MPI_Cart_rank(MPI_Comm comm, int *coords,
                  int *rank)
```

Paramètres

1. MPI_Comm **comm** : le communicateur de la structure cartésienne
2. int ***coords** : les coordonnées du processus dans la topologie
3. int ***rank** : le rang de processus associé aux coordonnées spécifiées

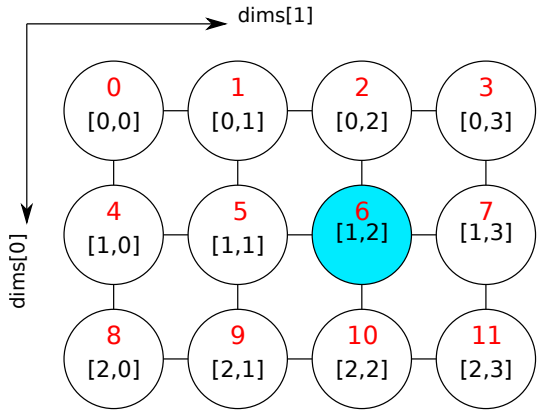
Routine MPI_Cart_rank

```
int coords[2], rank;
coords[0]= 1; coords[1]= 2;
if (pid==0){
   MPI_Cart_rank(comm_cart,coords,&rank);
   printf("Proc. (%d,%d) a le rang %d",
   coords[0], coords[1], rank);
}
```

Routine MPI_Cart_rank

```
int coords[2], rank;
coords[0]= 1; coords[1]= 2;
if (pid==0){
   MPI_Cart_rank(comm_cart,coords,&rank);
   printf("Proc. (%d,%d) a le rang %d",
   coords[0], coords[1], rank);
}
```

Le processus 0 calcule le rang d'un processus avec ses coordonnées



Routine MPI_Cart_coords

Cette routine fournit les coordonnées d'un processus de rang donné dans la grille.

```
int MPI_Cart_coords(MPI_Comm comm, int rank,
                    int maxdims, int *coords)
```

Paramètres

1. MPI_Comm **comm** : le communicateur de la structure cartésienne
2. int **rank** : le rang d'un processus au sein du communicateur
3. int **maxdims** : le nombre de dimensions des coordonnées
4. int ***coords** : Les coordonnées cartésiennes du processus spécifié

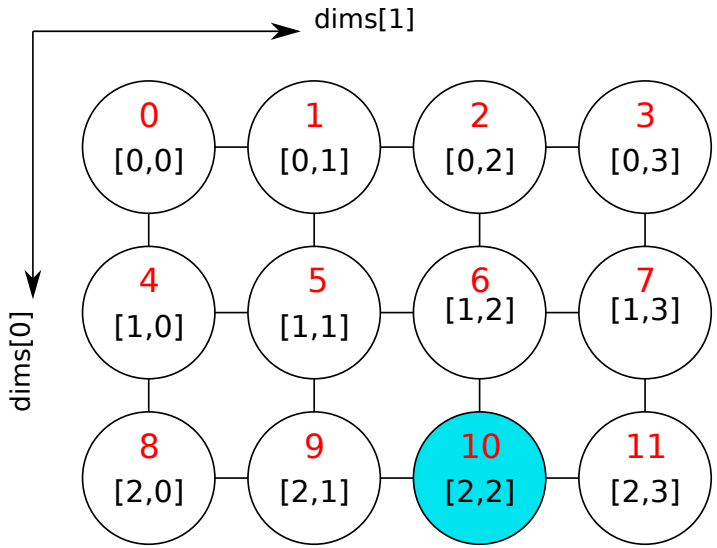
Routine MPI_Cart_coords

```
if (pid==0){
   int coords[2], rank=10;
   MPI_Cart_coords(comm_cart, rank,ndims, coords);
   printf("Proc. %d aux coords. [%d,%d]",rank,
           coords[0], coords[1]);
}
```

Routine MPI_Cart_coords

```
if (pid==0){
   int coords[2], rank=10;
   MPI_Cart_coords(comm_cart, rank,ndims, coords);
   printf("Proc. %d aux coords. [%d,%d]",rank,
          coords[0], coords[1]);
}
```

Proc. 0 calcule les coordonnées de Proc. 10



Routine MPI_Cart_shift

Cette routine permet de connaître le rang **des voisins** d'un processus **dans une direction donnée**.

```
int MPI_Cart_shift(MPI_Comm comm, int direction,
                   int displ, int *source, int *dest)
```

Paramètres

1. MPI_Comm **comm** : Le communicateur de la topologie cartésienne
2. int **direction** : la direction de voisinage souhaitée dans les coordonnées cartésiennes, directions $\in$ [0, n-1] pour un maillage cartésien à n dimensions

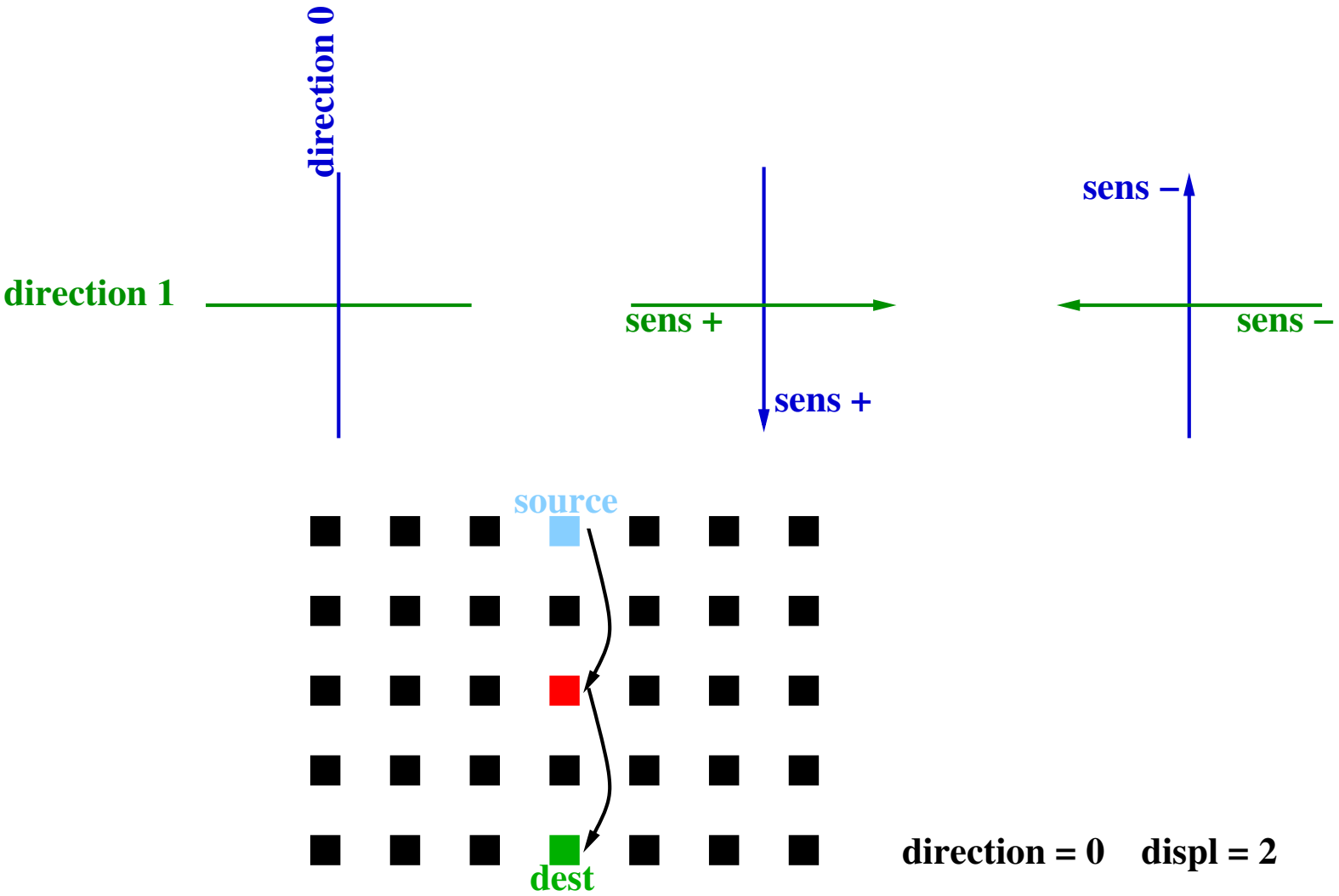
Routine MPI_Cart_shift

```
int MPI_Cart_shift(MPI_Comm comm, int direction,
                   int displ, int *source,
                   int *dest)
```

Paramètres

1. int **displ** : la taille du déplacement et son sens grâce au signe
2. int ***source** : le rang du processus voisin source dans la direction et le sens indiqués
3. int ***dest** : le rang de processus voisin destination dans la direction et le sens indiqués

Routine MPI_Cart_shift



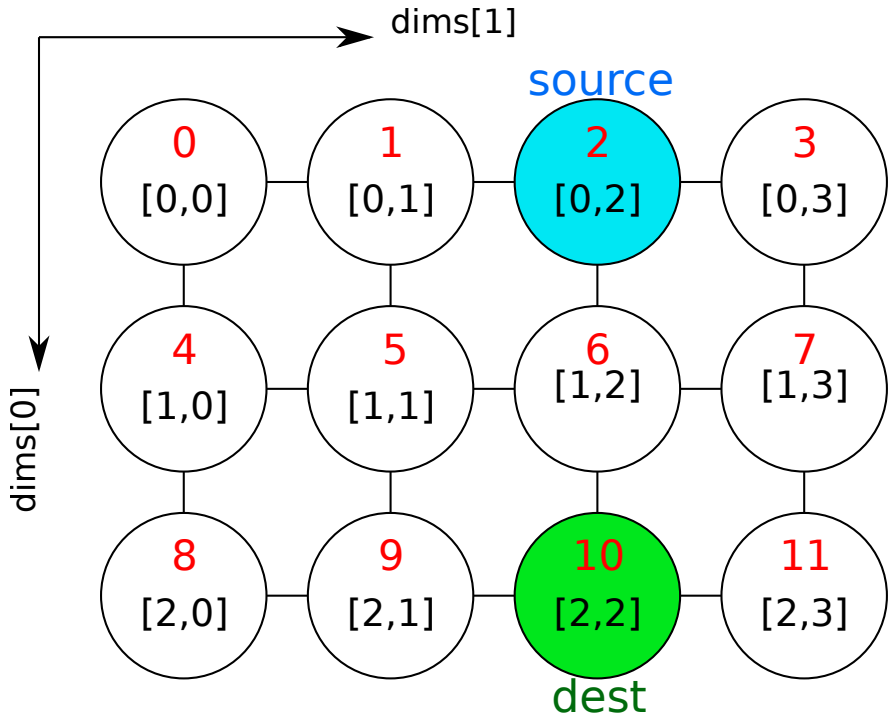
Routine MPI_Cart_shift

```
int nup, nlow;
MPI_Cart_shift(comm_cart, 0, 1, &nup, &nlow);
printf("Proc. %d a voisin au-dessus %d\n",pid, nup);
printf("Proc. %d a voisin au-dessous %d\n",pid, nlow);
```

Routine MPI_Cart_shift

```
int nup, nlow;
MPI_Cart_shift(comm_cart, 0, 1, &nup, &nlow);
printf("Proc. %d a voisin au-dessus %d\n",pid, nup);
printf("Proc. %d a voisin au-dessous %d\n",pid, nlow);
```

Proc. 6 cherche ses voisins dans la direction 0



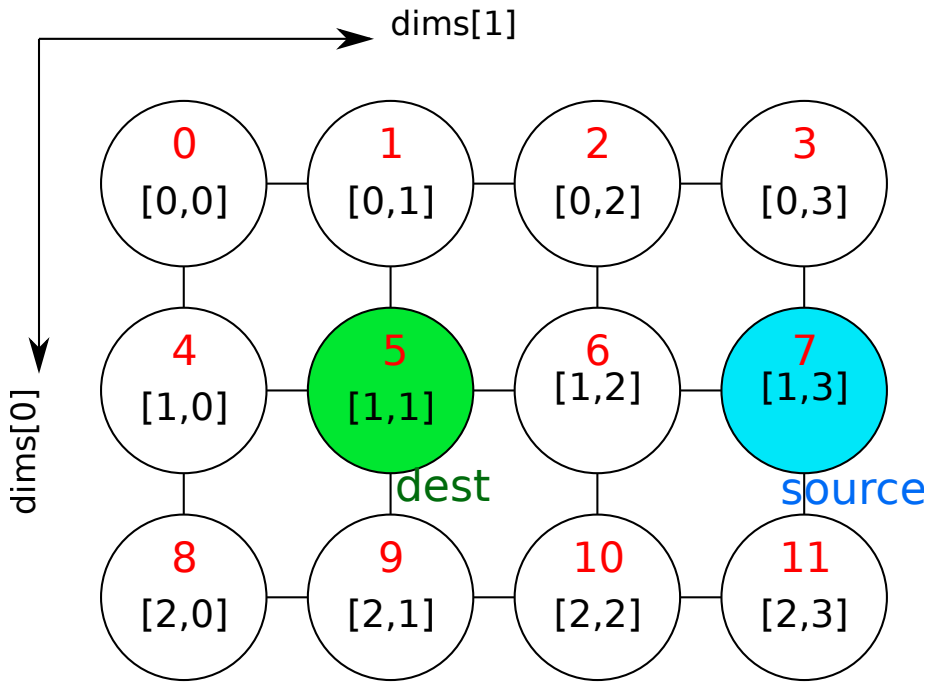
Routine MPI_Cart_shift

```
int nleft, nright;
MPI_Cart_shift(comm_cart, 1, -1, &nright, &nleft);
printf("Proc. %d a voisin droite %d\n",pid, nright);
printf("Proc. %d a voisin gauche %d\n",pid, nleft);
```

Routine MPI_Cart_shift

```
int nleft, nright;
MPI_Cart_shift(comm_cart, 1, -1, &nright, &nleft);
printf("Proc. %d a voisin droite %d\n",pid, nright);
printf("Proc. %d a voisin gauche %d\n",pid, nleft);
```

Proc. 6 cherche ses voisins dans la direction 1



Routine MPI_Cart_get

Cette routine donne les informations sur le communicateur donné de dimensions maxdims.

```
int MPI_Cart_get(MPI_Comm comm, int maxdims,
                 int *dims, int *periods,
                 int *coords)
```

Paramètres

1. MPI_Comm comm : Communicateur de la structure cartésienne
2. int maxdims : Longueur du vecteur de dimensions (nombre de dimensions)

Routine MPI_Cart_get

```
int MPI_Cart_get(MPI_Comm comm, int maxdims,
                 int *dims, int *periods,
                 int *coords)
```

Paramètres

1. int *dims : Tableau d'entiers du nombre de processus pour chaque dimension
2. int *periods : Tableau d'entiers des périodicités pour chaque dimension
3. int *coords : Coordonnées du processus appelant dans la topologie cartésienne

Routine MPI_Cartdim_get

Cette routine retourne le nombre de dimensions pour le communicateur indiqué dans la structure cartésienne.

```
int MPI_Cartdim_get(MPI_Comm comm, int *ndims)
```

Paramètres

1. MPI_Comm comm : Communicateur de la structure cartésienne
2. int *ndims : Nombre de dimensions de la structure cartésienne du communicateur

L'intérêt des topologies

Faciliter les communications

- Si la décomposition du domaine est liée à la topologie on peut raisonner avec les coordonnées des processus
- Le rang du processus sera utilisé pour les communications point-à-point
- Communications collectives 

Décomposer la topologie

- Créer des sous-groupes par ligne et par colonne.

Exemples au tableau

1. la transposition de la matrice
2. une étape du pivot de gauss

Routine MPI_Cart_sub

Principe

- Cette routine partitionne un communicateur **en sous-groupes**
- Ces sous-groupes forment des sous-grilles cartésiennes avec des dimensions inférieures à l'ancienne
- L'intérêt majeur est de pouvoir effectuer **des opérations collectives** restreintes à un sous-ensemble de processus appartenant à :
  1. une même ligne (ou colonne), si la topologie initiale est 2D
  2. un même plan, si la topologie initiale est 3D

Routine MPI_Cart_sub

```
int MPI_Cart_sub(MPI_Comm comm, int *remain_dims,
                 MPI_Comm *comm_new)
```

Paramètres

1. MPI_Comm **comm** : Communicateur de la structure cartésienne
2. int ***remain_dims** : Quelles directions sont conservées dans la sous-grille
3. MPI_Comm ***comm_new** : Retourne un des nouveaux communicateurs créés qui contient le processus appelant

Routine MPI_Cart_sub

remain_dims = (true, false, true)

Supposons que MPI_Cart_create a défini une grille de (2×3×4).

- Nous avons 3 nouvelles sous-grilles
- Chacune a 8 processus avec une topologie cartésienne de 2×4

remain_dims = (false, false, true)

- Nous avons 6 nouvelles sous-grilles
- Chacune a 4 processus avec une topologie cartésienne 1D

La topologie **graphe**

Les processus sont représentés comme les nœuds d'un graphe

P	edges	index
0	1,2	2
1	0,2,3,4	6
2	0,1	8
3	1,4	10
4	1,3	12

Création de la topologie

```
int MPI_Graph_create(MPI_Comm comm_old, int nnodes,
                     int *index, int *edges, int reorder,
                     MPI_Comm *comm_graph);
```

Les paramètres

1. MPI_Comm comm_old : le communicateur de départ
2. int nnodes : le nombre de nœuds dans le graphe
3. int *index : lié au degré des nœuds et à edges
4. int *edges : les arêtes
5. int reorder : pour autoriser MPI à réordonner les processus
6. MPI_Comm *comm_graph : le nouveau communicateur

Accès aux informations pour communiquer

```
int MPI_Graph_neighbors_count(MPI_Comm comm,
                              int rank, int *nneighbors);
```

Les paramètres

1. MPI_Comm comm : le communicateur de la topologie graphe
2. int rank : le rang du processus
3. int* nneighbors : retourne le nombre de voisins

Accès aux informations pour communiquer

```
int MPI_Graph_neighbors(MPI_Comm comm, int rank,
                        int maxneighbors, int *neighbors);
```

Les paramètres

1. MPI_Comm comm : le communicateur
2. int rank : le rang du processus
3. int maxneighbors : le nombre de voisins
4. int* neighbors : les voisins du processus dans le graphe

Extrait du code de communication

```
int edges[12] = {1,2,0,2,3,4,0,1,1,4,1,3};
int index[5] = {2,6,8,10,12};
MPI_Graph_create(MPI_COMM_WORLD,5,index,edges,
                 true, &CommGraph);
int nb_v;
MPI_Graph_neighbors_count(CommGraph,pid,&nb_v);
int* v = new int[nb_v];
MPI_Graph_neighbors(CommGraph,pid,nb_v,v);
int a = pid;
int b;
int res = a;
for (int i=0; i<nb_v; i++) {
    MPI_Sendrecv (&a,1,MPI_INT,v[i],123,
                  &b,1,MPI_INT,v[i],MPI_ANY_TAG,
                  CommGraph, &status);
    res += b;
}
```