

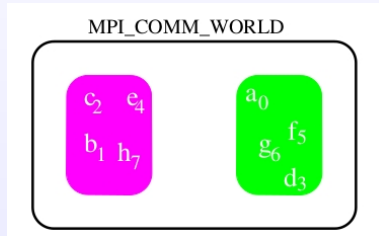
# Cours Calcul Intensif - MPI

Hélène Coullon , Sophie Robert, Sébastien Limet



07 novembre 2012

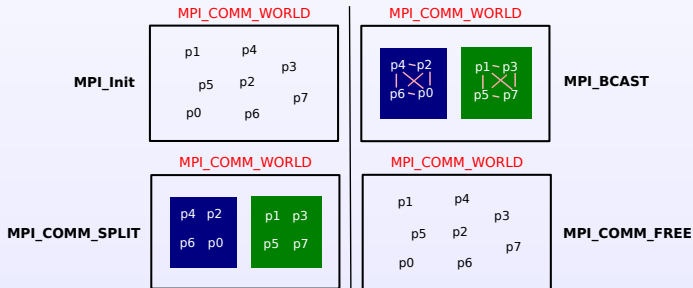
Il s'agit de partitionner un ensemble de processus MPI afin de créer des sous-ensembles sur lesquels on puisse effectuer des opérations telles que des communications point à point, collectives, etc. Chaque sous-ensemble ainsi créé aura son propre espace de communication.



- On ne peut créer un communicateur qu'à partir d'un autre communicateur
- Le communicateur par défaut `MPI_COMM_WORLD` permet de créer d'autres communicateurs
- `MPI_COMM_WORLD` est créé à `MPI_Init` et détruit à `MPI_Finalize`

# Exemple

- Regrouper d'une part les processus de rang pair et d'autre part les processus de rang impair
- Ne diffuser un message collectif qu'aux processus de rang pair et un autre message qu'aux processus de rang impair



- 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, qui permet de délimiter l'espace de communication
- Les contextes de communication sont gérés par MPI
- En pratique, pour construire un communicateur, il existe deux façons de procéder :
  - par l'intermédiaire d'un groupe de processus
  - directement à partir d'un autre communicateur

- Dans MPI, il existe diverses routines pour construire des communicateurs : **MPI\_Cart\_create**, **MPI\_Cart\_sub**, **MPI\_Comm\_create**, **MPI\_Comm\_dup** & **MPI\_Comm\_split**.
- Les constructeurs de communicateurs sont des opérateurs collectifs qui engendrent des communications entre les processus
- Les communicateurs peuvent être gérés dynamiquement et supprimés avec **MPI\_Comm\_free**

# MPI\_Comm\_split

Le sous-programme MPI\_Comm\_split permet de partitionner un communicateur donné en autant de communicateurs que l'on veut.

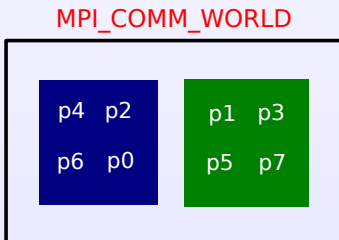
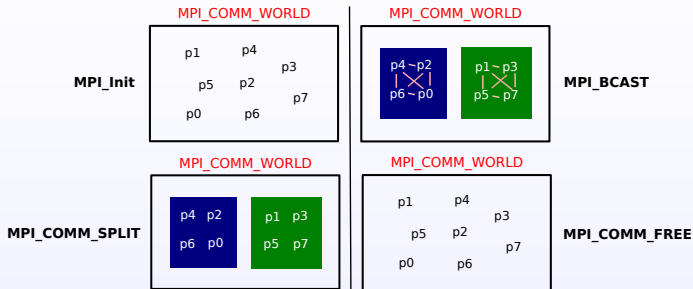
## Prototype

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

## Arguments

- MPI\_Comm **comm** : le communicateur à partir duquel on fait le partitionnement de processus
- int **color** : la couleur du processus, les processus de même couleur seront dans le même communicateur
- int **key** : la clé du processus, qui sera utilisée pour obtenir le nouvel identifiant du processus dans le nouveau communicateur
- MPI\_Comm **\*newcomm** : pointeur sur le nouveau communicateur obtenu

# Exemple



- 2 couleurs
- que ce passe-t-il si on met les même clés que dans `MPI_COMM_WORLD` ?



- 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 du type cartésien ou graphe
  - Topologies de type cartésien
    - 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
  - Topologies de type graphe : généralisation à des topologies plus complexes

## Principes

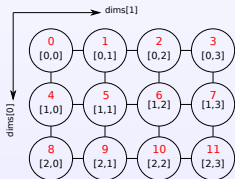
- La grille de processus est définie par
  - Sa dimension
  - Sa périodicité
  - Le nombre de processus dans chaque dimension

## Routines

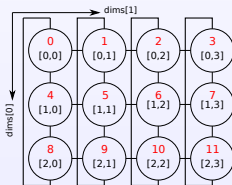
- Création d'une topologie cartésienne
- Création des bonne taille de dimensions suivant le nombre de processus
- Rang d'un processus dans une topologie cartésienne
- Coordonnées  $(x,y,z)$  d'un processus dans la topologie
- Partitionner un communicateur de topologie cartésienne en sous-groupes

# Routine MPI\_Cart\_create (1)

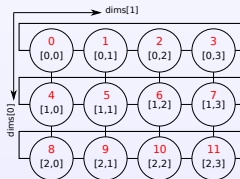
- Cette routine permet de créer une topologie cartésienne
- La routine est collective, elle concerne donc l'ensemble des processus appartenant à l'ancien communicateur



Non périodique



Lignes périodiques



Colonne périodiques

# Routine MPI\_Cart\_create (2)

## Prototype

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

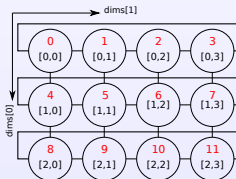
## Paramètres

- ❶ MPI\_Comm **old\_comm** : Ancien communicateur (Le plus simple : MPI\_COMM\_WORLD)
- ❷ int **ndims** : Nombre de dimensions
- ❸ int **\*dim\_size** : Nombre de processus dans chaque direction
- ❹ int **\*periods** : Tableau indiquant la périodicité pour chaque direction
- ❺ int **reorder** : Le rang des processus peut-il être modifié ou non
- ❻ MPI\_Comm **\*new\_comm** : Nouveau communicateur avec la structure cartésienne

# Routine MPI\_Cart\_create (3)

Exemple : Création une topologie cartésienne de  $3 \times 4$

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



# Routine MPI\_Dims\_create (1)

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

## Prototype

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

## Paramètres

- ❶ int **nnodes** : Nombre de processus dans la grille
- ❷ int **ndims** : Nombre de dimensions souhaitées
- ❸ int **dims** : Nombre de processus par dimension obtenu

Exemple : Création une topologie cartésienne avec répartition automatique des processus dans les dimensions

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

# Routine MPI\_Cart\_rank (1)

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

## MPI\_Cart\_rank

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

## Paramètres

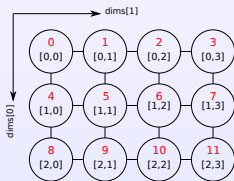
- ❶ MPI\_Comm **comm** : Communicateur de la structure cartésienne
- ❷ int \***coords** : Les coordonnées cartésiennes desquelles on souhaite récupérer le numéro de processus
- ❸ int \***rank** : Rang de processus associé aux coordonnées spécifiées



# Routine MPI\_Cart\_rank (2)

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

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



## Résultat

Proc. à (1, 2) a le rang 6

# Routine MPI\_Cart\_coords (1)

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

## Prototype

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

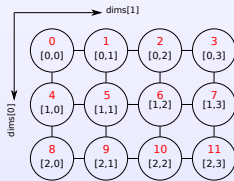
## Paramètres

- ❶ MPI\_Comm **comm** : Communicateur de la structure cartésienne
- ❷ int **rank** : Rang d'un processus au sein du communicateur
- ❸ int **maxdims** : Taille des coordonnées coords (Longueur du vecteur)
- ❹ int **\*coords** : Les coordonnées cartésiennes du processus spécifié

## Routine MPI\_Cart\_coords (2)

Proc. 0 calcule les coordonnées de Proc. 10

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



### Résultat

Proc. 10 aux coords. [2,2]

# Routine MPI\_Cart\_shift (1)

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

## Prototype

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

## Paramètres

- ❶ MPI\_Comm **comm** : Communicateur de la structure cartésienne
- ❷ int **direction** : Direction de voisinage souhaitée dans les coordonnées cartésiennes, directions  $\in [0, n-1]$  pour un maillage cartésien à n dimensions

## Prototype

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

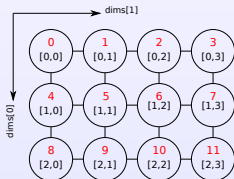
## Paramètres

- ❶ int **displ** : Taille du déplacement en voisinage (exemple : voisins à 2 cases horizontalement : droite vers gauche ou gauche vers droite) : >0 upward, <0 downward
- ❷ int **\*source** : Rang du processus voisin source dans la direction et le sens indiqués
- ❸ int **\*dest** : Rang de processus voisin destination dans la direction et le sens indiqués

# Routine MPI\_Cart\_shift (2)

Proc. 6 cherche ses voisins dans la direction 0

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



## Résultat

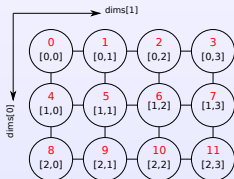
Proc. 6 a voisin au-dessus 2

Proc. 6 a voisin au-dessous 10

# Routine MPI\_Cart\_shift (3)

Proc. 6 cherche ses voisins dans la direction 1

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



## Résultat

Proc. 6 a voisin à droite 7  
Proc. 6 a voisin à gauche 5

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

## Prototype

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

## Paramètres

- ❶ MPI\_Comm comm : Communicateur de la structure cartésienne
- ❷ int maxdims : Longueur du vecteur de dimensions (nombre de dimensions)



## Prototype

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

## Paramètres

- ❶ int \*dims : Tableau d'entier du nombre de processus pour chaque dimension
- ❷ int \*periods : Tableau d'entier des périodicités pour chaque dimension
- ❸ int \*coords : Coordonnées du processus appelant dans la topologie cartésienne

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

## Prototype

```
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

## Principes

- Cette routine partitionne un communicateur en sous-groupes
- Ces sous-groupes forment sous-grilles cartésiennes avec leurs dimension plus petites que 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 (2)

## Prototype

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

## Paramètres

- ❶ MPI\_Comm **comm** : Communicateur de la structure cartésienne
- ❷ int **\*remain\_dims** : Quelles directions sont conservées dans la sous-grille
- ❸ MPI\_Comm **\*comm\_new** : Retourne un des nouveaux communicateurs créés qui contient le processus appelant

Supposons que MPI\_Cart\_create a défini une grille de  $(2 \times 3 \times 4)$ .

`remain_dims = (true, false, true)`

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

`remain_dims = (false, false, true)`

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