

# Systèmes Parallèles et Distribués

## 5. Applications parallèles

Franck Butelle Camille Coti

LIPN, Université Paris 13  
Formation Ingénieurs SupGalilée Info 3

01/2012

# Plan

- 1 Exemples d'applications MPI
- 2 Fonctionnalités avancées de MPI

# Plan

- 1 Exemples d'applications MPI
  - Décomposition de domaine
- 2 Fonctionnalités avancées de MPI

## Maître-esclave

### Distribution des données

Le maître distribue le travail aux esclaves

- Le maître démultiplexe les données, multiplexe les résultats
- Les esclaves ne **communiquent pas** entre eux

### Efficacité

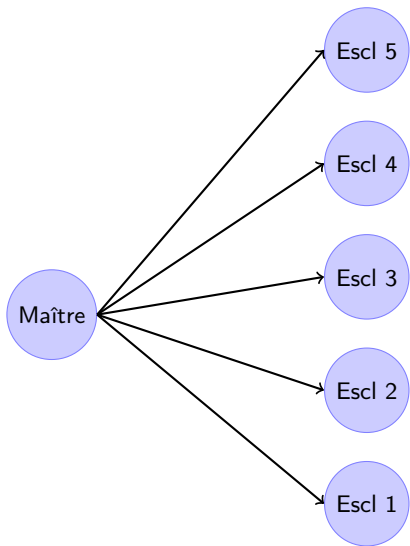
Files d'attentes de données et résultats au niveau du maître

- On retrouve la partie séquentielle de la loi d'Amdahl
- Communications : maître ↔ esclaves
- Les esclaves ne travaillent que quand ils attendent des données ou qu'ils envoient leurs résultats

Seuls les esclaves participent effectivement au calcul

- Possibilité d'un bon speedup à grande échelle (esclaves  $\gg$  maître) **si les communications sont rares**
- Peu rentable pour quelques processus
- Attention au bottleneck au niveau du maître

## Maître-esclave



### Équilibrage de charge

Statique :

- Utilisation de `MPI_Scatter` pour distribuer les données
- `MPI_Gather` pour récupérer les résultats

Dynamique :

- Mode **pull** : les esclaves demandent du travail
- Le maître envoie les chunks 1 par 1

### Travaux pratiques 2

Implémenter un squelette de maître-esclave utilisant les deux méthodes. Avantages et inconvénients ?

# Découpage en grille

## Grille de processus

On découpe les données et on attribue un processus à chaque sous-domaine

## Décomposition 1D

Découpage en bandes

0	1	2	3
---	---	---	---

## Décomposition 2D

Découpage en rectangles, plus scalable

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

## Ghost region

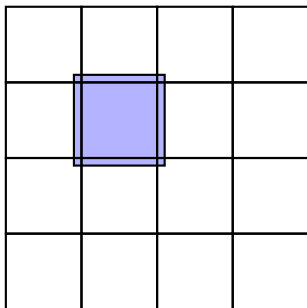
### Frontières entre les sous-domaines

Un algorithme peut avoir besoin des valeurs des points voisins pour calculer la valeur d'un point

- Traitement d'images (détection de contours...), automates cellulaires...

Réplication des données situées à la frontière

- Chaque processus dispose d'un peu des données des processus voisins
- Mise à jour à la fin d'une étape de calcul



### Travaux pratiques 3

Utilisation d'une ghost region dans le maître-esclave dynamique en utilisant une décomposition 1D.

## Utilisation d'une topologie

### Décomposition en structure géométrique

On transpose un communicateur sur une topologie cartésienne

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

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

### En 2D

- `comm_old` : le communicateur de départ
- `ndims` : ici 2
- `dims` : nombre de processus dans chaque dimension (ici {4, 4})
- `periods` : les dimensions sont-elles périodiques
- `reorder` : autoriser ou non de modifier les rangs des processus
- `comm_cart` : nouveau communicateur

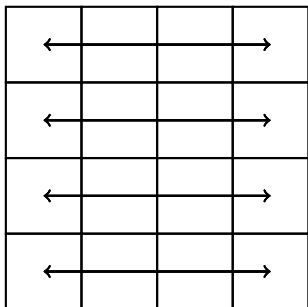


## Utilisation d'une topologie

### Extraction de sous-communicateurs

Communicateur de colonnes, de rangées...

- `int MPI_Cart_sub( MPI_Comm comm_old, int *remain_dims, MPI_Comm *comm_new );`



### Communicateurs de lignes

- `comm_old` : le communicateur de départ
- `remain_dims` : quelles dimensions sont dans le communicateur ou non
- `comm_new` : le nouveau communicateur

### Travaux pratiques 4

Implémenter la grille 2D de gauche et faire un broadcast sur chaque rangée.

# Plan

- 1 Exemples d'applications MPI
- 2 Fonctionnalités avancées de MPI
  - Datatypes
  - Réductions

## Utilisation des datatypes MPI

### Principe

- On définit le datatype
  - ▶ `MPI_Type_contiguous`, `MPI_Type_vector`, `MPI_Type_hvector`,  
`MPI_Type_indexed`, `MPI_Type_hindexed`, `MPI_Type_struct`
- On le commit
  - ▶ `MPI_Type_commit`
- On le libère à la fin
  - ▶ `MPI_Type_free`

### Combinaison des types de base

`MPI_CHAR`, `MPI_SHORT`, `MPI_INT`, `MPI_LONG`, `MPI_UNSIGNED_CHAR`,  
`MPI_UNSIGNED_SHORT`, `MPI_UNSIGNED_LONG`, `MPI_UNSIGNED`, `MPI_FLOAT`,  
`MPI_DOUBLE`, `MPI_LONG_DOUBLE`, `MPI_BYTE`

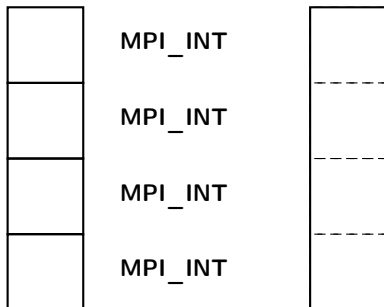
## Construction de datatypes MPI

### Données contiguës

On crée un block d'éléments :

- `int MPI_Type_contiguous( int count, MPI_Datatype oldtype, MPI_Datatype *newtype );`

### VECTOR



### Travaux pratiques 5

Implémenter l'anneau à jeton en faisant circuler des vecteurs en utilisant un `MPI_Datatype` (un seul élément sera envoyé à chaque communication, quel que soit la taille des données envoyées).

## Construction de datatypes MPI

### Vecteur de données

On crée un vecteur d'éléments :

- `int MPI_Type_vector( int count, int blocklength, int stride  
MPI_Datatype oldtype, MPI_Datatype *newtype );`

On agrège des blocs de **blocklength** éléments séparés par un vide de **stride** éléments.

## Construction de datatypes MPI

### Structure générale

On donne les éléments, leur nombre et l'offset auquel ils sont positionnés.

- `int MPI_Type_struct( int count, int *array_of_blocklengths, MPI_Aint *array_of_displacements, MPI_Datatype *array_of_types, MPI_Datatype *newtype);`

### Exemple

On veut créer une structure MPI avec un entier et deux flottants à double précision

- `count = 2` : on a deux éléments
- `array_of_blocklengths = 1, 2` : on a 1 entier et 2 doubles
- `array_of_displacements = 0, sizeof( int), sizeof( int ) + sizeof( double )` (ou utilisation de `MPI_Address` sur une instance de cette structure)
- `array_of_types = MPI_INT, MPI_DOUBLE`

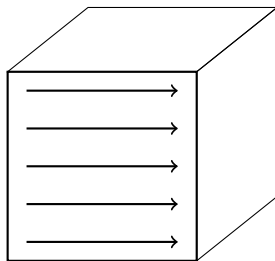
## Exemple d'utilisation : FFT 3D

## La FFT 3D

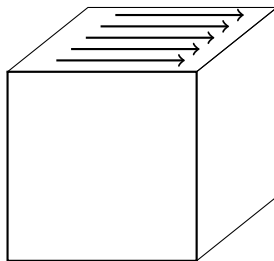
Calcul de la transformée de Fourier 3D d'une matrice 3D

- FFT dans une dimension
- FFT dans la deuxième dimension
- FFT dans la troisième dimension

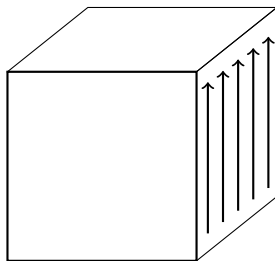
Direction X



Direction Y



Direction Z



## Parallélisation de la FFT 3D

### Noyau de calcul : la FFT 1D

La FFT 1D est parallélisable

- Chaque vecteur peut être calculé indépendamment des autres

On veut calculer chaque vecteur **séquentiellement** sur **un seul processus**

- Direct pour la 1ere dimension
- Nécessite une transposition pour les dimensions suivantes

### Transposition de la matrice

On effectue une rotation de la matrice

- Redistribution des vecteurs
  - ▶ All to All
- Rotation des points (opération locale)



## Rotation des points

### Algorithme

```
MPI_Alltoall(tab 2 dans tab 1 )  
for( i = 0; i < c; ++i ) {  
  for( j = 0; j < b; ++j ) {  
    for( k = 0; k < a; ++k ) {  
      tab2[i][j][k] = tab2[i][k][j];  
    }  
  }  
}
```

### Complexité

- Le Alltoall coûte cher
- La rotation locale est en  $O(n^3)$

**On essaye d'éviter le coût de cette rotation  
en le faisant faire par la bibliothèque MPI**

## Datatypes non-contigüs

### Sérialisation des données en mémoire

La matrice est sérialisée en mémoire

- Vecteur[0][0], Vecteur[0][1], Vecteur[0][2]... Vecteur[1][0], etc

### Rotation des données sérialisées

x	x	x	x	→	x	o	o	o
o	o	o	o		x	o	o	o
o	o	o	o		x	o	o	o
o	o	o	o		x	o	o	o

### Utilisation d'un datatype MPI

C'est l'écart entre les points des vecteur qui change avant et après la rotation → on définit un datatype différent en envoi et en réception.

Avantages :

- La rotation est faite par la bibliothèque MPI
- Gain d'une copie (buffer) au moment de l'envoi / réception des éléments (un seul élément au lieu de plusieurs)

## Définition d'opérations

### Syntaxe

- `int MPI_Op_create( MPI_User_function *function, int commute, MPI_Op *op );`

On fournit un pointeur sur fonction. La fonction doit être associative et peut être commutative ou pas. Elle a un prototype défini.

### Exemple

- Définition d'une fonction :

```
void addem( int *, int *, int *, MPI_Datatype * );
void addem( int *invec, int *inoutvec, int *len, MPI_Datatype *dtype ){
    int i;
    for ( i = 0; i < *len; i++ )
        inoutvec[i] += invec[i];
}
```

- Déclaration de l'opération MPI :

```
MPI_Op_create( (MPI_User_function *)addem, 1, &op );
```

## Exemple d'utilisation : TSQR

### Définition

La *décomposition QR* d'une matrice  $A$  est une décomposition de la forme

$$A = QR$$

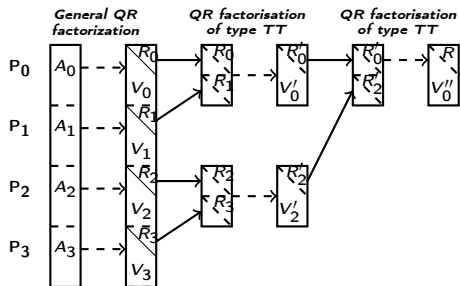
Où  $Q$  est une matrice orthogonale ( $QQ^T = I$ )  
et  $R$  est une matrice triangulaire supérieure.

### TSQR basé sur CAQR

Algorithme à évitement de communications pour matrices "tall and skinny" (hauteur  $\gg$  largeur)

- On calcule plus pour communiquer moins (optimal en nombre de communications)
  - ▶ Les communications coûtent cher, pas les flops
- Calcul de facto QR partielle sur des sous-domaines (en parallèle), recomposition 2 à 2, facto QR, etc

## Algorithme TSQR



## Algorithme

Sur un arbre binaire :

- QR sur sa sous-matrice
- Communication avec le voisin
  - Si rang pair : réception de  $r+1$
  - Si rang impair : envoi à  $r-1$
- Si rang impair : fin

## Arbre de réduction

### Opération effectuée

On effectue à chaque étape de la réduction une factorisation QR des deux facteurs  $R$  mis l'un au-dessus de l'autre :

$$R = QR(R_1, R_2)$$

- C'est une opération binaire
  - ▶ Deux matrices triangulaires en entrée, une matrice triangulaire en sortie

- Elle est associative

$$QR(R_1, QR(R_2, R_3)) = QR(QR(R_1, R_2), R_3)$$

- Elle est commutative

$$QR(R_1, R_2) = QR(R_2, R_1)$$

### Utilisation de MPI\_Reduce

L'opération remplit les conditions pour être une MPI\_Op dans un MPI\_Reduce

- Définition d'un datatype pour les facteurs triangulaires supérieurs  $R$
- Utilisation de ce datatype et de l'opération QR dans un MPI\_Reduce