MPI2

Hélène Coullon, Sébastien Limet, Sophie Robert

Université d'Orléans

2012-2013

Plan

Bilan de MPI1

Fonctionnalités

- Environnement de communication
- Communications point à point
- Communications collectives
- Types de données dérivés
- Topologies
- Communicateurs

Manques de MPI1

- Gestion dynamique des processus
- Communication par accès distant à la mémoire
- Les entrées-sorties parallèles (dans des fichiers)
- + quelques fonctionnalités comme l'interfaçage avec C++ etc...

Objectifs du cours

- Communication par copie mémoire (RMA)
  - Éléments classiques de l'accès concurrent à une ressource
- Gestion dynamique des processus
  - Modèle Maître/Esclave: Lancement par un maître (éventuellement parallèle) de processus esclave
  - Modèle Client/Serveur: Enregistrement de services et connexion par des processus à ces services
- Les entrées-sorties parallèles
  - Quelques fonctionnalités: gestion des fichiers distribués

Plan

Principes

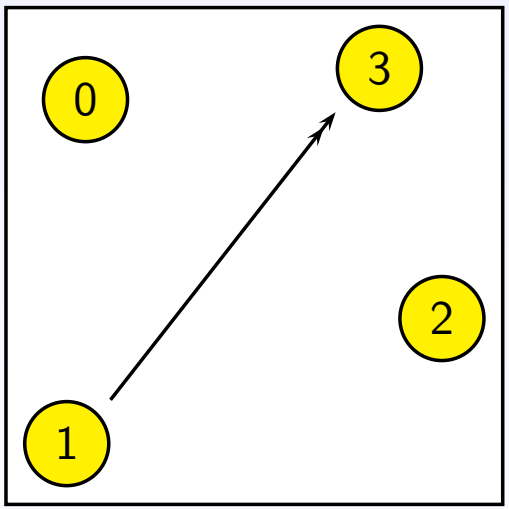
Objectifs

- Sortir du modèle de communication par message
- Permettre une communication plus souple et efficace
- Modèle de communication passive
  - RMA: Remote Memory Access
  - OSC: One Sided Communication

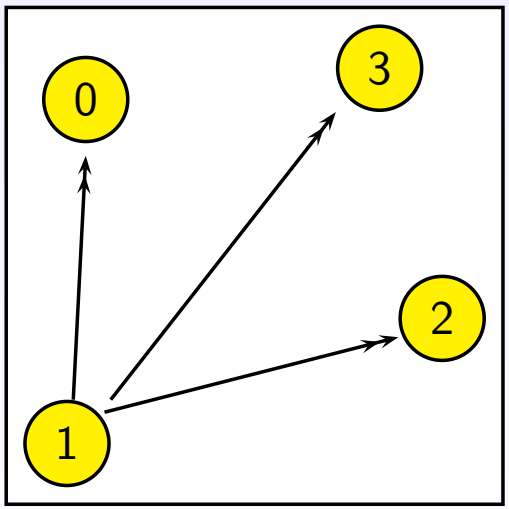
Communications par passage de messages (Rappel)

Principes

- Les processus émetteur et récepteur doivent participer à la communication.
- Les paramètres de communication sont spécifiés par les parties



Point-à-point



Collective

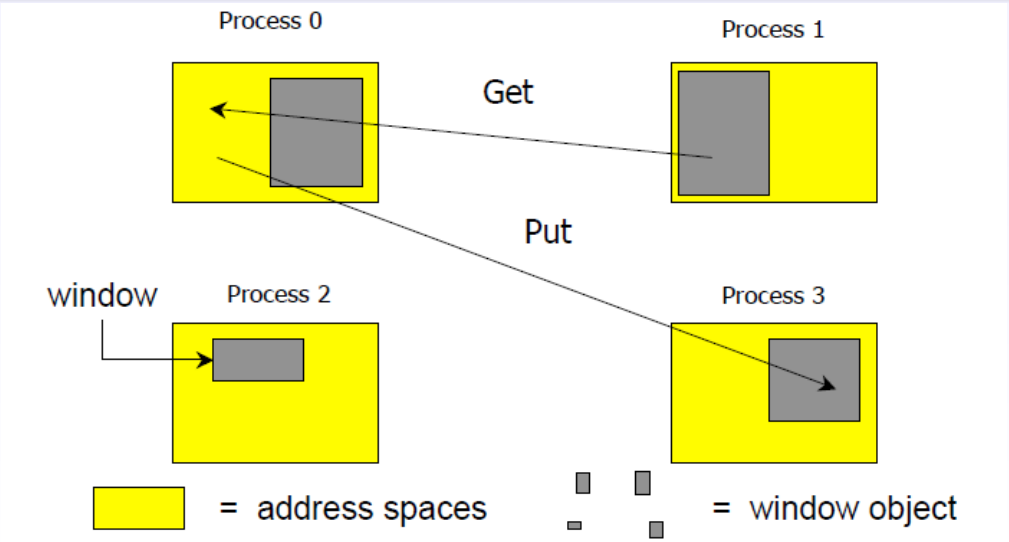
Remote Memory Access (RMA)

Principe

- Un seul des deux processus effectuent la communication
  One Sided Communication
- Le processus souhaitant faire une communication va directement se servir dans la mémoire de l'autre
  Remote Memory Access

$\Rightarrow$ Problèmes

- de synchronisations
- d'accès concurrents à une zone mémoire



RMA en MPI: Mise en œuvre

Principaux objets

- Les fenêtres: parties de la mémoire local mise à disposition des autres processeurs
- Opérations de communications
  - `MPI_Put, MPI_Accumulate`: écritures dans la mémoire distante
  - `MPI_Get`: Lecture dans la mémoire distante
- Les synchronisations: commandes permettant de (forcer à) mettre à jour les zones de mémoire partagée
  - `MPI_Win_fence`: barrière de synchronisation
  - `MPI_Win_lock,MPI_Win_unlock` : synchronisation point à point

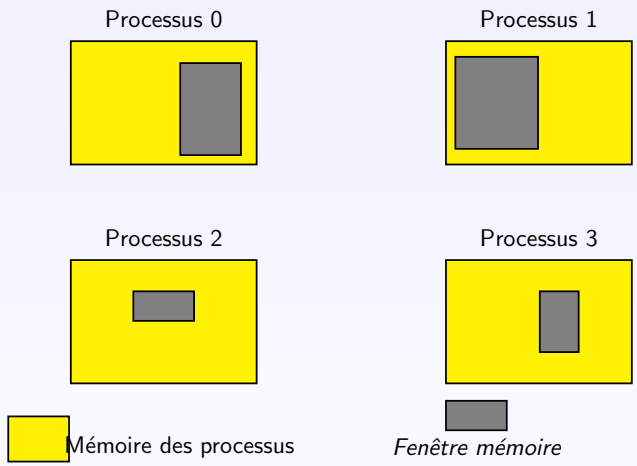
Les Fenêtres: MPI_Win

Definition

C'est une zone mémoire qu'un processus met à disposition des autres processus

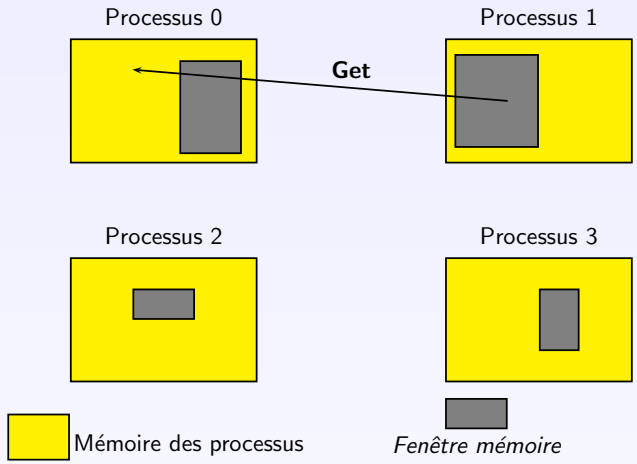
Opérations

- MPI_Win_create
  - pointeur sur la zone mémoire de la fenêtre
  - nombre de cases mémoires de la zone
  - taille d'une cellule
  - Structure MPI_INFO
  - Communicateur gérant cette fenêtre
  - La fenêtre ainsi créée
- MPI_Win_free
  - La fenêtre à libérer



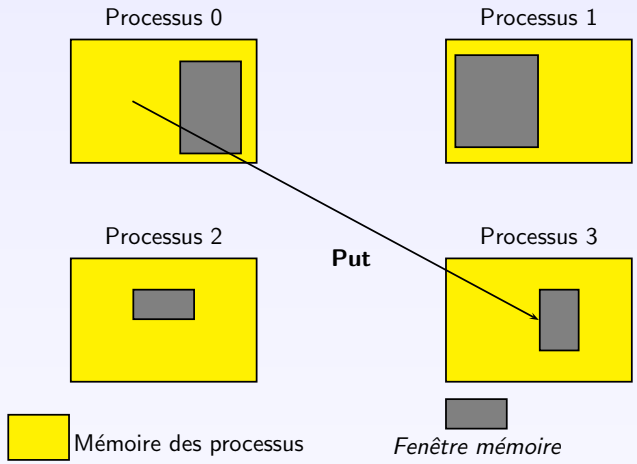
Lectures

- `MPI_Get`
  - pointeur sur la zone mémoire locale
  - Nombre d'éléments de cette zone
  - Type de données
  - Rang MPI du processus distant
  - Numéro de case de début de lecture dans la fenêtre distante
  - Nombre de cases à lire
  - Type des cases dans la fenêtre distante
  - Fenêtre concernée par le get



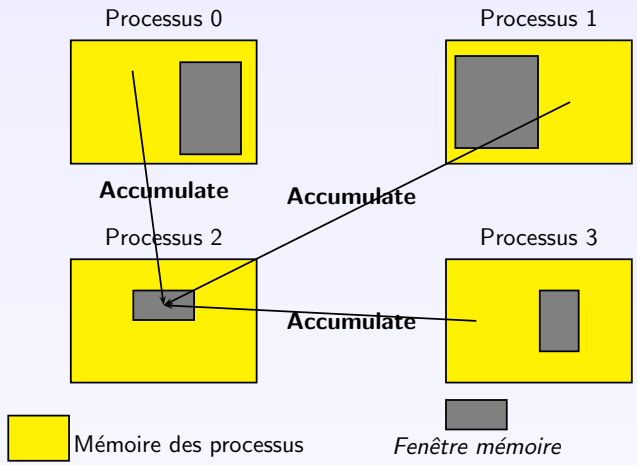
Écritures

- `MPI_Put`
  - pointeur sur la zone mémoire locale
  - Nombre d'éléments de cette zone
  - Type de données
  - Rang MPI du processus distant
  - Numéro de case de début de lecture dans la fenêtre distante
  - Nombre de cases à lire
  - Type des cases dans la fenêtre distante
  - Fenêtre concernée par le get



Écritures

- `MPI_Accumulate`
  - pointeur sur la zone mémoire locale
  - Nombre d'éléments de cette zone
  - Type de données
  - Rang MPI du processus distant
  - Numéro de case de début de lecture dans la fenêtre distante
  - Nombre de cases à lire
  - Type des cases dans la fenêtre distante
  - Opération d'accumulation
  - Fenêtre concernée par le get



Synchronisation des RMA

Objectifs

- RMA $\Rightarrow$ un seul processus actif dans la communication
- Problèmes pour connaître la *fraîcheur* d'une information

$\Rightarrow$ Mécanismes de synchronisation

1. Les barrières: `MPI_Win_fence`
2. Les verrous: `MPI_Win_lock, MPI_Win_lock`

Les barrières: MPI_Win_fence

Fonctionnement

- Tous les processus du communicateur exécute la barrière
- Toutes les RMA sur tous les processus sont finalisées avant de reprendre

Avantages/inconvénients

+ Très simple à mettre en place
+ Très sûr
- Coûteux (équivalent à un MPI_Barrier)
- Casse un peu le modèle One Sided Communication

Les barrières: MPI_Win_fence

MPI_Win_fence

- assertions
- fenêtre sur laquelle porte la barrière

Les assertions

- Informations prises en compte ou non par l'implémentation de MPI2
- Permet d'optimiser les barrières
- Les différentes assertions
  - MPI_MODE_NOPRECEDE: Pas de fence précédente
  - MPI_MODE_NOPUT: Promesse de ne pas faire de put dans la fence
  - MPI_MODE_NOSTORE: on promet qu'aucune mise à jour de la fenêtre n'a été faite depuis le dernier fence
  - MPI_MODE_NOSUCCEED: c'est la dernière fence du l'application.
  - 0: toujours valide pas d'assertion

Les verrous: MPI_Win_lock, MPI_Win_unlock

Fonctionnement

- Synchronisation entre l'émetteur et le récepteur
- Sur le verrouillage, bloque (ou non) l'accès aux autres processus
- Sur le déverrouillage,la fenêtre distante finalise les mises à jour la concernant, puis se rend de nouveau accessible

Avantages/inconvénients

+ Système beaucoup plus fin
+ Permet des communications efficaces
- On entre dans la complexité de la programmation concurrente

Les verrous: MPI_Win_lock, MPI_Win_unlock

MPI_Win_lock

- type de verrou
  - MPI_LOCK_EXCLUSIVE
  - MPI_LOCK_SARDE
- rang du processus distant
- assertions
- fenêtre concernée par le verrous

MPI_Win_unlock

- rang du processus distant
- fenêtre concernée par le verrous

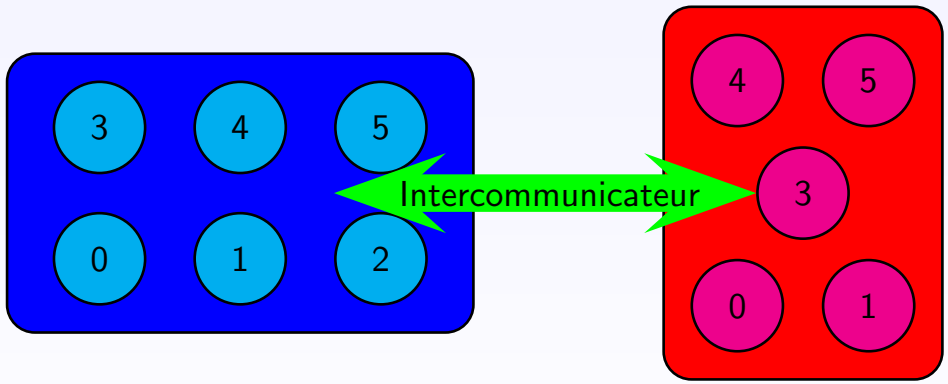
RMA: Exemples

Plan

Modèles de gestions des processus

Généralités

- Sortir du modèle SPMD induit par MPI
- Permettre de mieux exploiter les architectures parallèles actuelles
- Deux groupes de processus vont
  - être lancés séparément
  - communiquer via un inter-communicateur



Les deux modèles

Le mode maître/esclave

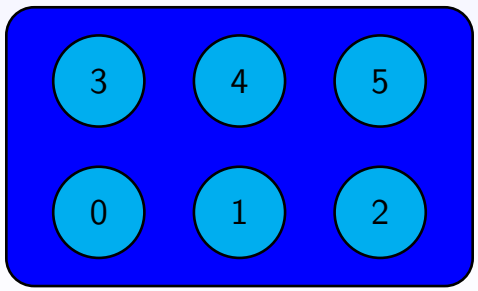
Un (ensemble de) processus maître lance en cours d'exécution un (ensemble de) processus esclave

1. Les processus maître sont lancés classiquement, dans leur `MPI_Comm_world`
2. Lancement des processus esclave par un `MPI_Spawn` dans leur propre `MPI_Comm_world`
3. Création d'un intercommunicateur entre les deux ensembles de processus

Le mode maître/esclave

Un (ensemble de) processus maître lance en cours d'exécution un (ensemble de) processus esclave

1. Les processus maître sont lancés classiquement, dans leur `MPI_Comm_world`
2. Lancement des processus esclave par un `MPI_Spawn` dans leur propre `MPI_Comm_world`
3. Création d'un intercommunicateur entre les deux ensembles de processus

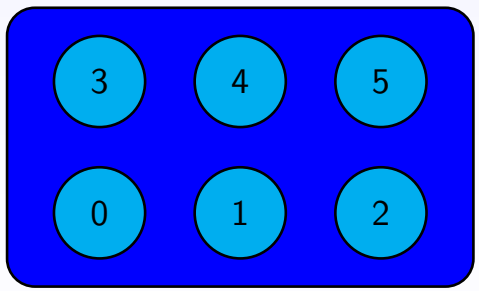


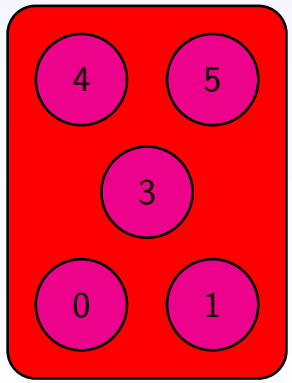
Les deux modèles

Le mode maître/esclave

Un (ensemble de) processus maître lance en cours d'exécution un (ensemble de) processus esclave

1. Les processus maître sont lancés classiquement, dans leur `MPI_Comm_world`
2. Lancement des processus esclave par un `MPI_Spawn` dans leur propre `MPI_Comm_world`
3. Création d'un intercommunicateur entre les deux ensembles de processus



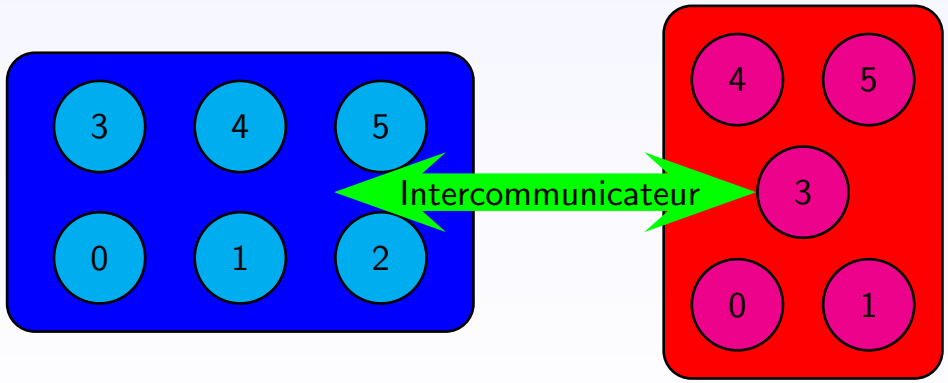


Les deux modèles

Le mode maître/esclave

Un (ensemble de) processus maître lance en cours d'exécution un (ensemble de) processus esclave

1. Les processus maître sont lancés classiquement, dans leur `MPI_Comm_world`
2. Lancement des processus esclave par un `MPI_Spawn` dans leur propre `MPI_Comm_world`
3. Création d'un intercommunicateur entre les deux ensembles de processus

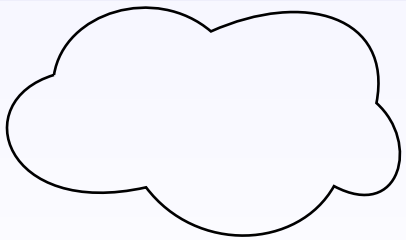


Les deux modèles

Le mode client/Serveur

Un (ensemble) de processus serveur va proposer un service qui pourra être utilisé par des clients. Modèle des sockets TCP/IP

1. Un processus particulier gère les services
2. Le(s) processus serveur enregistre son service
3. Le(s) processus client recherche le service et font une demande
4. Le serveur accepte la requête $\Rightarrow$ création d'un intercommunicateur entre les deux ensembles de processus

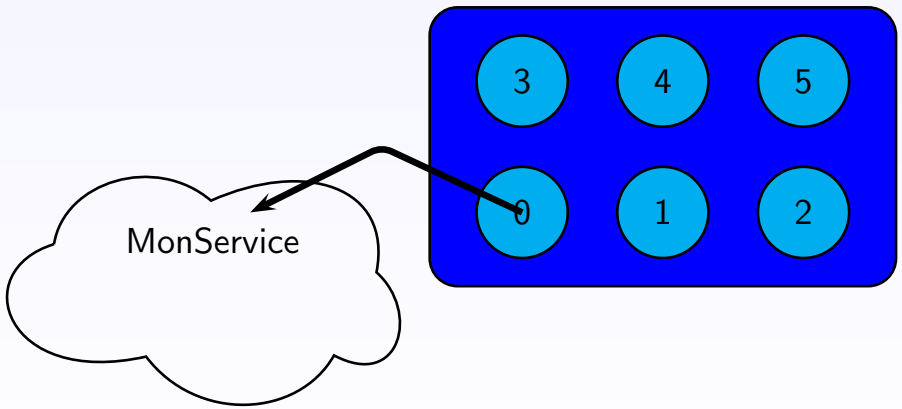


Les deux modèles

Le mode client/Serveur

Un (ensemble) de processus serveur va proposer un service qui pourra être utilisé par des clients. Modèle des sockets TCP/IP

1. Un processus particulier gère les services
2. Le(s) processus serveur enregistre son service
3. Le(s) processus client recherche le service et font une demande
4. Le serveur accepte la requête $\Rightarrow$ création d'un intercommunicateur entre les deux ensembles de processus

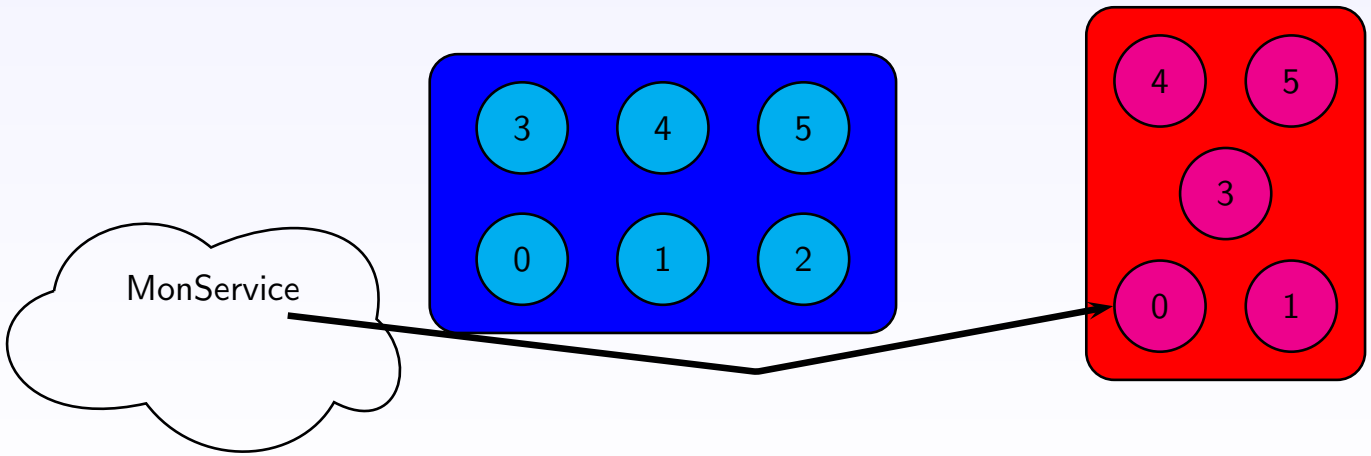


Les deux modèles

Le mode client/Serveur

Un (ensemble) de processus serveur va proposer un service qui pourra être utilisé par des clients. Modèle des sockets TCP/IP

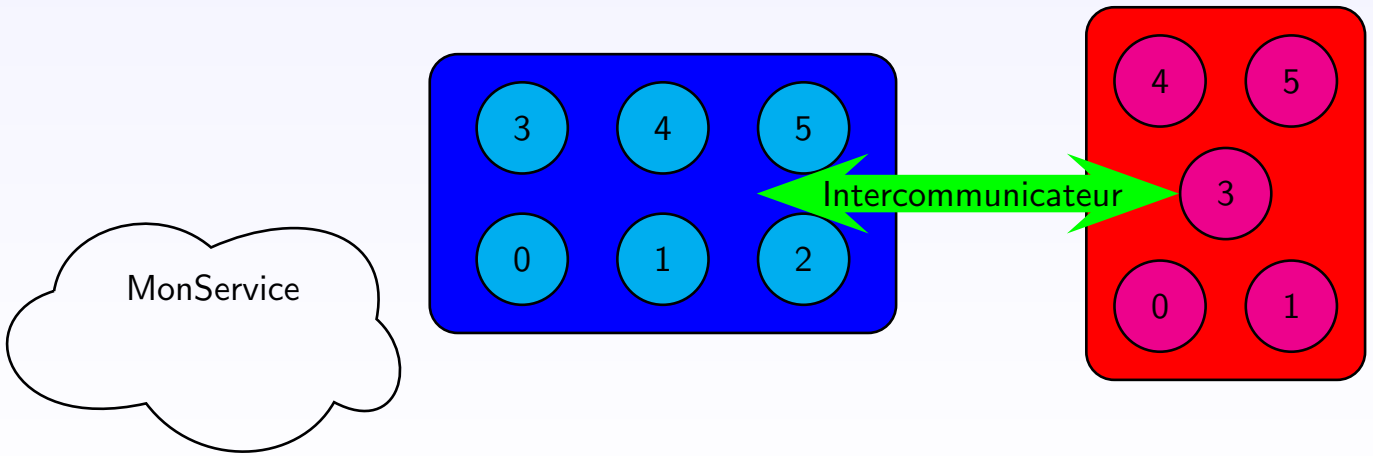
1. Un processus particulier gère les services
2. Le(s) processus serveur enregistre son service
3. Le(s) processus client recherche le service et font une demande
4. Le serveur accepte la requête ⇒ création d'un intercommunicateur entre les deux ensembles de processus



Le mode client/Serveur

Un (ensemble) de processus serveur va proposer un service qui pourra être utilisé par des clients. Modèle des sockets TCP/IP

1. Un processus particulier gère les services
2. Le(s) processus serveur enregistre son service
3. Le(s) processus client recherche le service et font une demande
4. Le serveur accepte la requête ⇒ création d'un intercommunicateur entre les deux ensembles de processus



Maître/esclave: Mise en œuvre

Coté maître

- Commande `MPI_Comm_spawn`
  - nom de l'exécutable
  - liste des arguments de l'exécutable
  - nombre d'instances à lancer
  - `MPI_INFO`
  - rang du processus maître qui effectue réellement le spawn
  - communicateur des maîtres
  - intercommunicateur
  - tableau des erreurs

Coté esclave

Il s'agit principalement de récupérer l'intercommunicateur

- Commande `MPI_Comm_get_parent`
  - intercommunicateur vers les parents

Maître/esclave: Mise en œuvre

Coté maître

- Commande `MPI_Comm_spawn_multiple`
  - nombre de commandes
  - tableau de noms d'exécutable
  - tableau de liste des arguments des exécutables
  - tableau contenant le nombre d'instances pour chaque commande
  - liste de `MPI_INFO`
  - rang du processus maître qui effectue réellement le spawn
  - communicateur des maîtres
  - tableau d'intercommunicateurs
  - tableau de tableaux des erreurs

Coté esclave

Il s'agit principalement de récupérer l'intercommunicateur

- Commande `MPI_Comm_get_parent`
  - intercommunicateur vers les parents

Exemples

Client/Serveur: Mise en œuvre

Coté Serveur

- `MPI_Open_port` Ouverture du port du service
  - `MPI_INFO`
  - nom du port (c'est un paramètre résultat)
- `MPI_Publish_name` publication du service
  - nom du service
  - `MPI_INFO`
  - nom du port
- `MPI_Comm_accept` attente de connexions
  - nom du port
  - `MPI_INFO`
  - rang du processus qui gère les connexions
  - communicateur du serveur
  - intercommunicateur

Client/Serveur: Mise en œuvre

Coté client

- `MPI_Lookup_name` recherche du service
  - nom du service
  - `MPI_INFO`
  - nom du port de connexion
- `MPI_Comm_connect` connexion au service
  - nom du port
  - `MPI_INFO`
  - rang du client qui gère vraiment la demande
  - communicateur des clients
  - intercommunicateur

Client/Serveur: Mise en œuvre

Lancement de l'application

- Lancer le serveur de nom
  `ompi-server -r /tmp/ompi-server.txt`
- Lancer le processus qui offre le service
  `mpirun --ompi-server file:/tmp/ompi-server.txt -np 4 serveur`
- Lancer le processus client
  `mpirun --ompi-server file:/tmp/ompi-server.txt -np 10 client`

Exemples

Plan