






**MPI-1 2ième partie :**  
**Programmation « non bloquante » et**  
**communications de groupe**

Stéphane Vialle





Stephane.Vialle@supelec.fr  
<http://www.metz.supelec.fr/~vialle>

---

---

---

---

---

---

---

---



**MPI-1 2ième partie :**  
**Programmation « non bloquante »**  
**et communications de groupe**

6 – Communications point à point non bloquantes

7 – Communications de groupe

8 – Produit de matrices denses sur anneau avec recouvrement

9 – Bilan de programmation MPI

A – Comparaisons performances/coûts de 4 clusters

B – Produit de matrices denses sur tore

---

---

---


---

---

---

---

---



MPI-1 2ième partie :  
Programmation « non bloquante » et communications de groupe

**6 – Communications point à point**  
**non bloquantes**

- Principes
- Ibsend-Irecv
- Issend-Irecv
- Irsend-Irecv
- Identificateurs persistents

---

---

---

---

---

---

---

---



Communications point à point non bloquantes

## Principes des comm. Non bloquantes

### Principes des communications non-bloquantes

`MPI_Ixxxx(..., MPI_Request *request)`

`Ibsend/Irecv - Issend/Irecv - Irsend/Irecv`

- **Isend ET Irecv** non bloquants
- Un paramètre de plus que leurs homologues bloquants :  
→ Id sur la communication demandée (`MPI_Request`)
- Besoin de savoir si les communications sont terminées :  
→ `MPI_Wait(request, ...)` ou `MPI_Test(request, ...)`

### Comparaison aux communications bloquantes :

- Nécessite une synchronisation explicite – Plus complexe
- Permet de réaliser du recouvrement calcul-communication

---

---

---

---

---

---

---

---

---

---



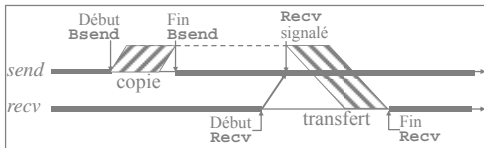
Communications point à point non bloquantes

## Ibsend – Irecv

### Mode *bufferisé* et non-bloquant : `MPI_Ibsend – MPI_Irecv`

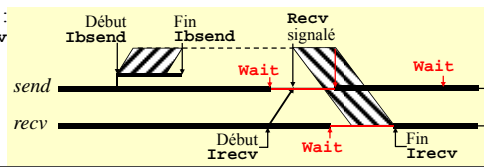
Rappel :

`Ssend-Recv`



Non-bloquant :

`Ibsend-Irecv`




---

---

---

---

---

---

---

---

---

---



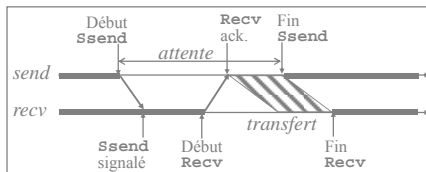
Communications point à point non bloquantes

## Issend – Irecv

### Mode *synchrone* et non-bloquant : `MPI_Issend – MPI_Irecv`

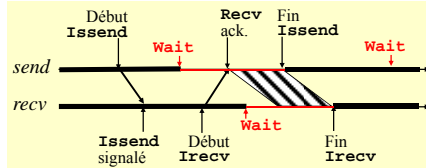
Rappel :

`Ssend-Recv`



Non-bloquant :

`Issend-Irecv`




---

---

---

---

---

---

---

---

---

---

Supélec

Communications point à point non bloquantes

## Irsend – Irecv

**Mode ready et non-bloquant : MPI\_Irsend – MPI\_Irecv**

Rappel : Rsend-Recv

Non-bloquant : Irsend-Irecv

---

---

---

---

---

---

---

---

---

---

Supélec

Communications point à point non bloquantes

## Identificateurs persistents

**Autre mode d'utilisation des comm. non bloquantes :**

```

Loop:
  MPI_Issend(...,&s_request); MPI_Irecv(...,&r_request);
  ..... // calculs
  MPI_Wait(&s_request,...); MPI_Wait(&r_request,...);
  
```

↓

```

MPI_Ssend_init(...,&s_request); MPI_Recv_init(...,&r_request);
Loop:
  MPI_Start(&s_request); MPI_Start(&r_request);
  ..... // calculs
  MPI_Wait(&s_request,...); MPI_Wait(&r_request,...);
MPI_Request_free(&s_request);
MPI_Request_free(&r_request);
  
```

→ Evite de réinitialiser des communications non-bloquantes

---

---

---

---

---

---

---

---

---

---

Supélec

MPI-1 2ième partie :

Programmation « non bloquante » et communications de groupe

## 7 – Communications de groupe

- Principes des communications de groupe
- Broadcast
- Scatter
- Gather
- Reduce

---

---

---

---

---

---

---

---

---

---

Supélec

Communications de groupe

## Principes des comm. de groupe

5 types principaux :

**Broadcast**      **Scatter**      **Gather**      **Reduce(op)**

=  $op(op(\dots, op(\dots, op(\dots, op(\dots))))$

+ les barrières !

- Utilisent les *communicator* et les groupes de processus
- Opérations bloquantes
- Des variantes existent : *all-reduce*, *all-to-all*, *scatterv*, ...
- **Routage optimisé selon le réseau sous-jacent**
  - arborescent – linéaire – sur bus – ...
  - parfois plus lent que des Send/Recv !!

---

---

---

---

---

---

---

---

---

---

Supélec

Communications de groupe

## Broadcast

```
int MPI_Bcast(buffer, count, datatype, root, comm )
void *buffer; /* Starting address of buffer */
int count; /* Number of entries in buffer (integer) */
MPI_Datatype datatype; /* Data type of buffer */
int root; /* Rank of broadcast root (integer) */
MPI_Comm comm; /* Communicator */
```

- Chaque processus exécute `MPI_Bcast` (différemment !)
- Chaque processus utilise sa propre adresse de buffer

Généralisation :  
MPI\_Alltoall et MPI\_Alltoallv

---

---

---

---

---

---

---

---

---

---

Supélec

Communications de groupe

## Scatter

```
int MPI_Scatter(sendbuf, sendcnt, sendtype,
               recvbuf, recvcnt, recvtype,
               root, comm)
void *sendbuf; /* Address of send buffer */
int sendcnt; /* Number of elements sent to each process */
MPI_Datatype sendtype; /* Data type of send buffer elements */
void *recvbuf; /* Address of receive buffer */
int recvcnt; /* Number of elements in receive buffer */
MPI_Datatype recvtype; /* Data type of receive buffer elements */
int root; /* Rank of sending process */
MPI_Comm comm; /* Communicator */
```

- Chaque processus exécute `MPI_Scatter` (différemment)
- Chaque processus utilise sa propre adresse de `recvbuf`

Variante : `MPI_scatterv` (avec décalage explicite des données)

---

---

---

---

---

---

---

---

---

---

Supélec

Communications de groupe

## Gather

```

int MPI_Gather(sendbuf, sendcnt, sendtype,
              recvbuf, recvcnt, recvtype, root, comm)
void *sendbuf; /* Starting address of send buffer */
int sendcnt; /* Number of elements in send buffer */
MPI_Datatype sendtype; /* Data type of send buffer elements */
void *recvbuf; /* Address of receive buffer */
int recvcnt; /* Number of elements for any single receive */
MPI_Datatype recvtype; /* Data type of recv buffer elements */
int root; /* Rank of receiving process */
MPI_Comm comm; /* Communicator */

```

- Chaque processus exécute `MPI_Gather` (différemment)
- Chaque processus utilise sa propre adresse de `sendbuf`

Généralisation : `MPI_Gatherv`, `MPI_Allgather`, `MPI_Allgatherv`

---

---

---

---

---

---

---

---

---

---

---

---

Supélec

Communications de groupe

## Reduce

```

int MPI_Reduce(sendbuf, recvbuf, count, datatype, op, root, comm)
void *sendbuf; /* Address of send buffer */
void *recvbuf; /* Address of receive buffer */
int count; /* Number of elements in send buffer */
MPI_Datatype datatype; /* Data type of elements of send buffer */
MPI_Op op; /* Reduce operation */
int root; /* Rank of root process */
MPI_Comm comm; /* Communicator */

```

**Reduce operations :**

`MPI_MAX`, `MPI_MIN`, `MPI_SUM`, `MPI_PROD`  
`MPI_LAND`, `MPI_BAND`, `MPI_LOR`, `MPI_BOR`  
`MPI_LXOR`, `MPI_BXOR`, `MPI_MINLOC`

+ définition de nouvelles opérations par `MPI_Op_create()`

**Généralisation :**

`MPI_Allreduce`, `MPI_Reduce_scatter`  
 → les résultats sont redistribués

---

---

---

---

---

---

---

---

---

---

---

---

Supélec

MPI-1 2ième partie :

Programmation « non bloquante » et communications de groupe

## 8 – Produit de matrices denses sur anneau avec recouvrement

- Algorithme distribué
- Modélisation des performances
- Exemples de performances

---

---

---

---

---

---

---

---

---

---

---

---

Supélec

Produit de matrices denses sur anneau

## Rappel d'algorithmique distribuée

Partitionnement sur un anneau de processeurs :

- A partitionnée en blocs de lignes
- B et C partitionnées en blocs de colonnes
- circulation de A
- B et C statiques

Topologie

Partitionnement et circulation de A

Partitionnement statique de B

Partitionnement statique de C

---

---

---

---

---

---

---

---

---

---

---

---

Supélec

Produit de matrices denses sur anneau

## Rappel d'algorithmique distribuée

Exemple de déroulement de l'algorithme sur PE-2, avec  $P = 4$  :

Étape 1 C

Étape 2 C

Étape 3 C

Étape 4 C

- Calcul de **toutes les colonnes** et **roulage de toutes les lignes** de A en **parallèle**, en  $P = 4$  étapes.
- Résultats après  $P$  étapes (seulement) :

Topologie

Partitionnement statique de C

---

---

---

---

---

---

---

---

---

---

---

---

Supélec

Produit de matrices denses sur anneau avec recouvrement

## Algorithme distribué

Stratégies d'implantation sur un anneau de  $P$  processeurs :

Sans recouvrement :

```
for (i=0; i<P; i++)
calcul();
barriere();
circulation();
barriere();
}
```

Avec recouvrement :

```
for (i=0; i<P; i++)
par {
calcul();
circulation();
}
barriere();
buffer_permut();
}
```

**1<sup>er</sup> algorithme : avec barrières conceptuelles**

Les barrières de synchronisations sont parfois implicites ... } Voir à  
 Les barrières apparaissent parfois inutiles ... } l'implant.

---

---

---

---

---

---

---

---

---

---

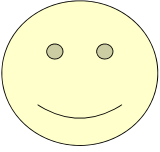
---

---

Produit de matrices denses sur anneau  
**Implant. en MPI-1 à comm. non-bloquantes**

Implantation MPI-1 par communications Non-bloquantes et recouv.

→ voir BE-4!




---

---

---

---

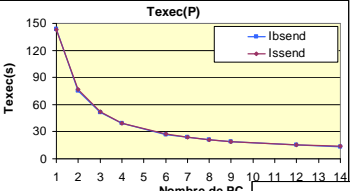
---

---

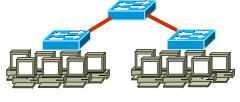
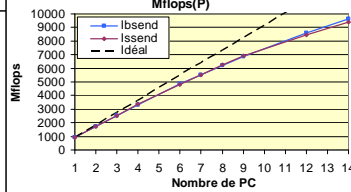
---

---

Produit de matrices denses sur anneau avec recouvrement  
**Exemple de performances**



- 12 x P4- 3GHz - 2Go Ram
- Ethernet Gigabit
- 3 switches

La perte de performances persiste :

- Elle n'est pas due au comm.
- Amdahl ?
- Calculs locaux trop petits ?

---

---

---

---

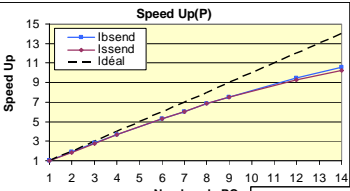
---

---

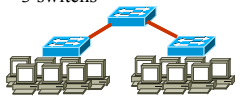
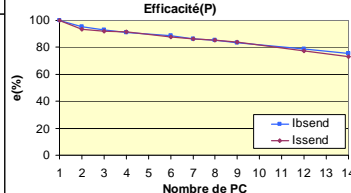
---

---

Produit de matrices denses sur anneau avec recouvrement  
**Exemple de performances**



- 12 x P4- 3GHz - 2Go Ram
- Ethernet Gigabit
- 3 switches

Les communications sont négligeables dans cette implantation!

Donc le recouvrement apporte peu sur un bon réseau.

---

---

---

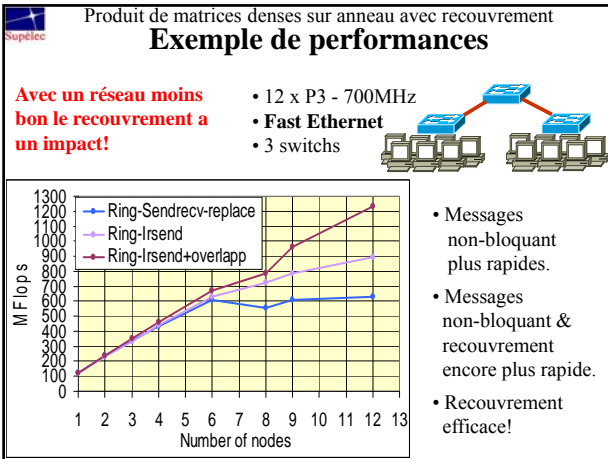
---

---

---

---

---




---

---

---

---

---

---

---

---

Supélec

MPI-1 2ième partie :  
Programmation « non bloquante » et communications de groupe

## 9 – Bilan de programmation MPI

---

---

---

---

---

---

---

---

Supélec

MPI : Programmation et algorithmique distribuée

### Bilan de programmation MPI

**Avantages :**

- Création de P processus dès le lancement
- Porté sur la plus large gamme d'architecture possible
- Une grande variété d'envois de messages possibles
- Aujourd'hui : *thread-safe*,  
- ex : mixable avec OpenMP sur cluster de PC bi-pro.

**Inconvénients :**

- Des différences de performances selon
  - le type d'envois de messages
  - l'application (volume et fréquence des comms)
  - l'architecture employée
- Des synchronisations parfois complexes
- Demande une modification importante du code séquentiel
  - conception parallèle à part entière

---

---

---

---

---

---

---

---

Supélec

## A – Comparaisons performances/coûts de 4 clusters

- Comparaison de speedup de Jacobi
- Mflops/Euros

---

---

---

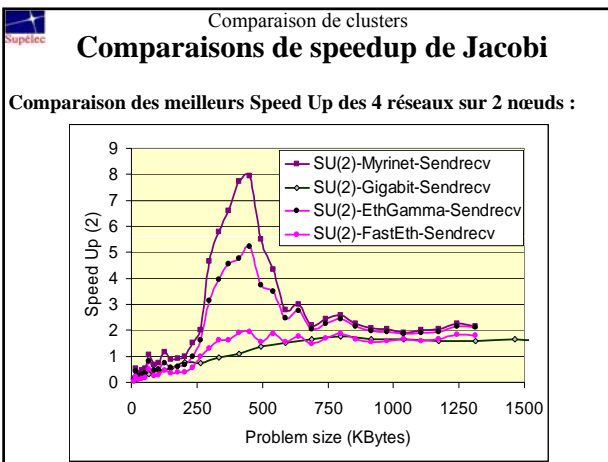
---

---

---

---

---




---

---

---

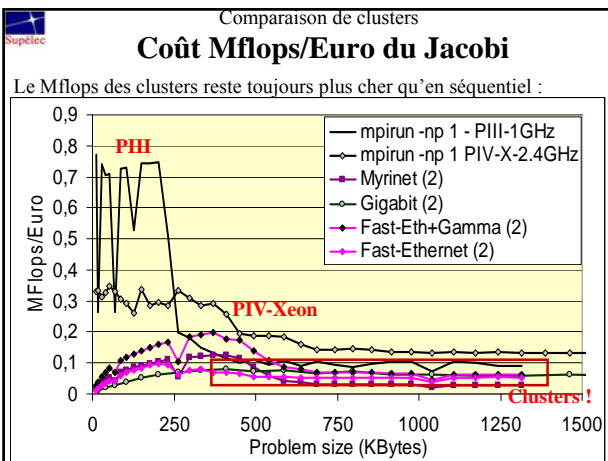
---

---

---

---

---




---

---

---

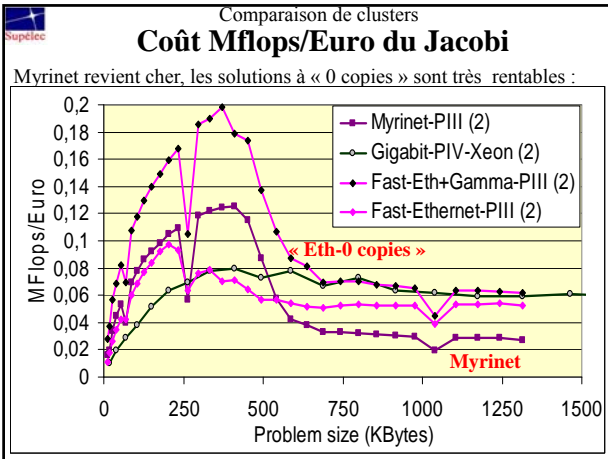
---

---

---

---

---




---

---

---

---

---

---

---

---

Supélec

MPI-1 2ième partie :  
 Programmation « non bloquante » et  
 communications de groupe

**FIN**

---

---

---

---

---

---

---

---