Les bases de MPI (Message Passing Interface)

Atelier T8.AP03

Thierry GARCIA

Les bases de MPI (Message Passing Interface)

Sitographie :

- http://www.idris.fr/formations/mpi/
- MPI forum : http://www.mpi-forum.org/
- http://perso.ensta-paristech.fr/~ciarlet/A1-2/A1-2-Introduction-MPI.pdf
- https://www.unilim.fr/pages_perso/jean.debord/math/matrices/matrices.htm
- http://calcul.math.cnrs.fr/Documents/Ecoles/LEM2I/Mod3/paral.pdf

Les bases de MPI (Message Passing Interface)

PRINCIPAUX OBJECTIFS

- Introduction et utilisation de la librairie MPI
- Mécanismes de communication parallèle.

Les bases de MPI (Message Passing Interface)

DESCRIPTION :

- aborder la simulation parallèle à travers les communications entre machines ;
- primitives de communication qui peuvent être utilisées pour la programmation parallèle ;
- exemple d'utilisation de ces primitives.

La simulation parallèle

L'informatique parallèle (high-performance computing) regroupe :

- conception d'algorithmes efficaces en temps,
- architecture des ordinateurs,
- mise en œuvre des programmes,
- analyse des performances.

La programmation parallèle

- Résolution de problèmes de calcul scientifique, de traitement d'images, de bases de données
  - gourmands en ressources CPU et en temps de calcul important
- Passer outre les limites des architectures séquentielles en terme :
  - de performance
  - d’accès à la mémoire
  - de tolérance aux pannes

Les machines parallèles

- machines avec une architecture parallèle ayant plusieurs processeurs qui permettent le traitement d'une application :

  dépasser les limites mémoire locale

  dépasser les limites processeurs machine

  accélérer les calculs (par exemple calcul matriciel, simulation numérique, ...)

Les machines parallèles

Classification (Taxonomie de Flynn[1] - 1966) :

- SISD (Single Instruction Single Data) : chaque processeur exécute en même temps la même instruction et une seule donnée est traitée. Machine séquentielle sans parallélisme - architecture de von Neumann.

  Par exemple : int A[100]; for (i=0;i<100;i++) A[i]=A[i]+A[i+1]; s'exécute sur une machine séquentielle en faisant A[0]+A[1] puis A[1]+A[2], … ;

- SIMD (Single Instruction Multiple Data) : chaque processeur exécute en même temps la même instruction sur des données différentes comme par exemple les machines systoliques ;

1 http://www.enseignement.polytechnique.fr/profs/informatique/Eric.Goubault/Cours05html/poly002.html

https://fr.wikipedia.org/wiki/Taxonomie_de_Flynn

Les machines parallèles

Classification (Taxonomie de Flynn[1] - 1966) :

- MISD (Multiple Instruction Single Data) : peut exécuter plusieurs instructions en même temps sur la même donnée. Ordinateur dans lequel une même donnée est traitée par plusieurs unités de calcul en parallèle (peu d'implémentations);

1 http://www.enseignement.polytechnique.fr/profs/informatique/Eric.Goubault/Cours05html/poly002.html

https://fr.wikipedia.org/wiki/Taxonomie_de_Flynn

Les machines parallèles

Classification (Taxonomie de Flynn[1] - 1966) :

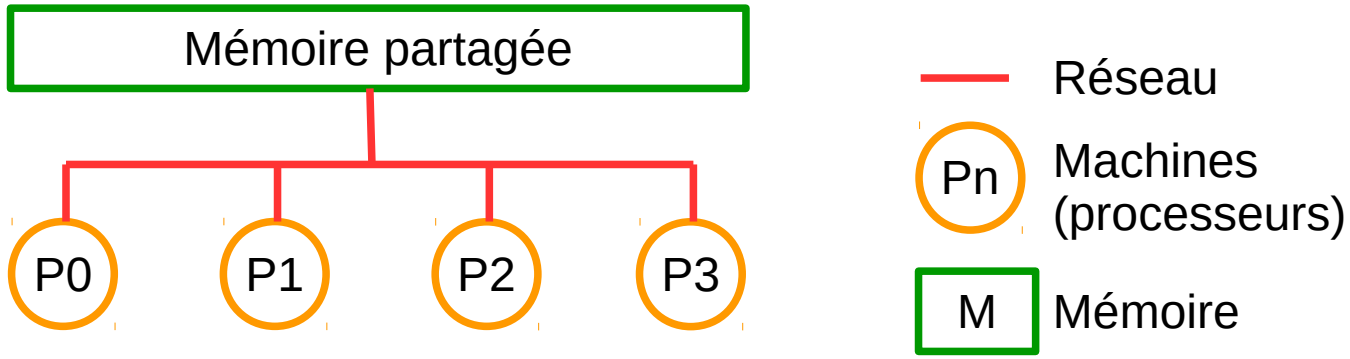
- MIMD (Multiple Instruction Multiple Data) : chaque processeur peut exécuter un programme différent sur des données différentes. On a plusieurs types d'architecture possibles (mémoire partagée ou mémoire locale + réseau de comunication)
- SPMD (Single Program Multiple Data) : souvent on exécute souvent le même code sur les processeurs d'une machine MIMD. Les processeurs exécutent en parallèle le même programme sur des données différentes.

1 http://www.enseignement.polytechnique.fr/profs/informatique/Eric.Goubault/Cours05html/poly002.html

https://fr.wikipedia.org/wiki/Taxonomie_de_Flynn

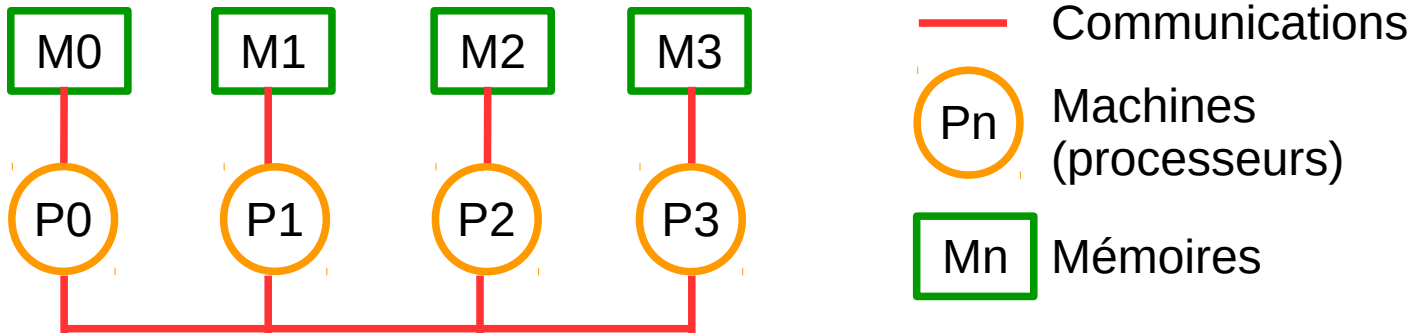
Les machines parallèles

- 2 types :
  - à mémoire partagée



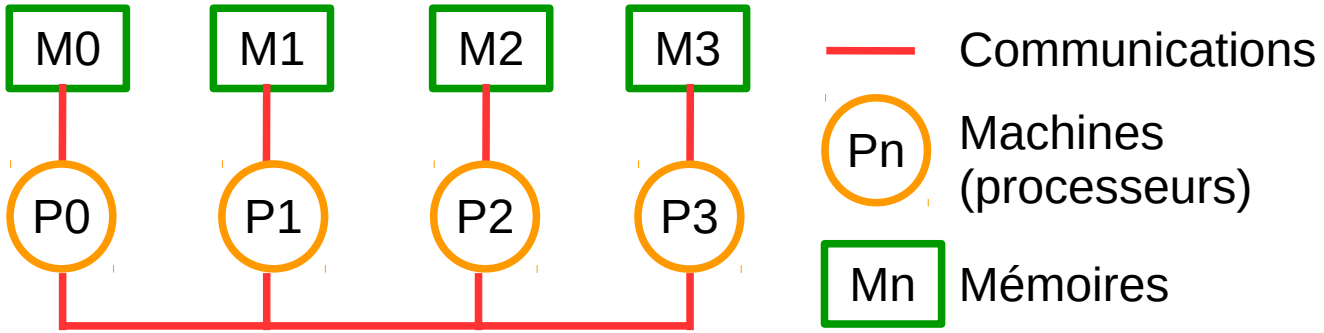
Les machines parallèles

- 2 types :
  - à mémoire distribuée



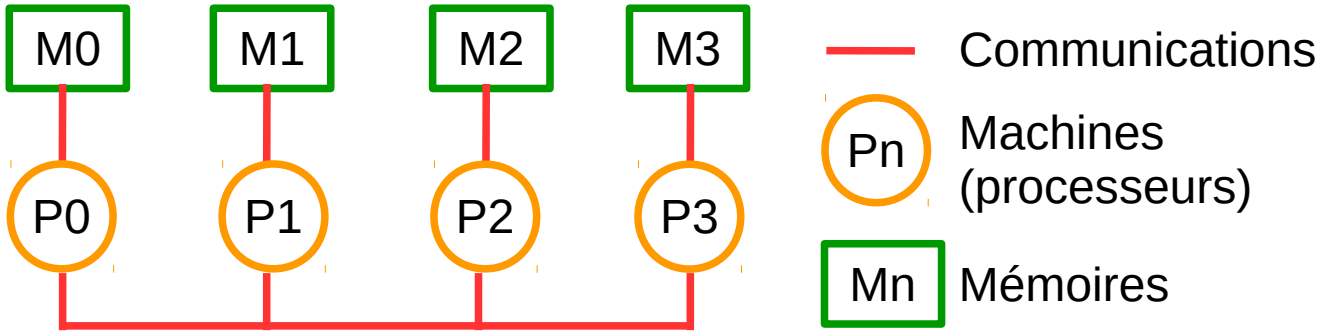
Les machines parallèles

- Caractéristiques :
  - interconnexion de machines indépendantes ;
  - mémoire locale à chaque processeur ;
  - chaque processeur exécute des instructions identiques sur des données identiques (SPMD) ;
  - parallélisme par échange de messages.



Les machines parallèles

- Caractéristiques :
  - interconnexion de machines indépendantes ;
  - mémoire locale à chaque processeur ;
  - chaque processeur exécute des instructions identiques sur des données identiques (SPMD) ;
  - parallélisme par échange de messages.



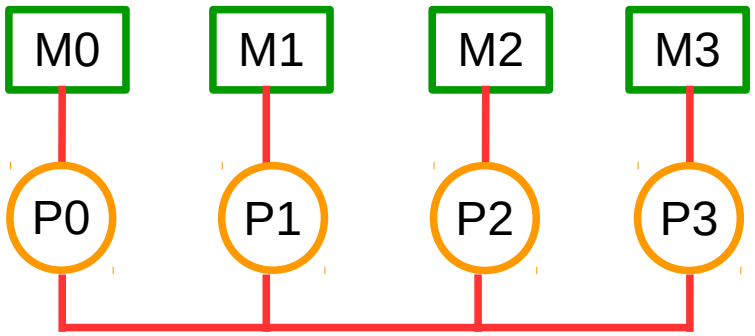
Parallélisme par échange de message

- Echange de messages entre processus (données, synchronisation, ...) ;
- Un processus dispose de ses données sans pouvoir accéder à celles des autres ;
- Gestion des échanges grâce à la librairie MPI (Message Passing Interface) ;
- MPI (norme) prend comme modèle le SPMD.

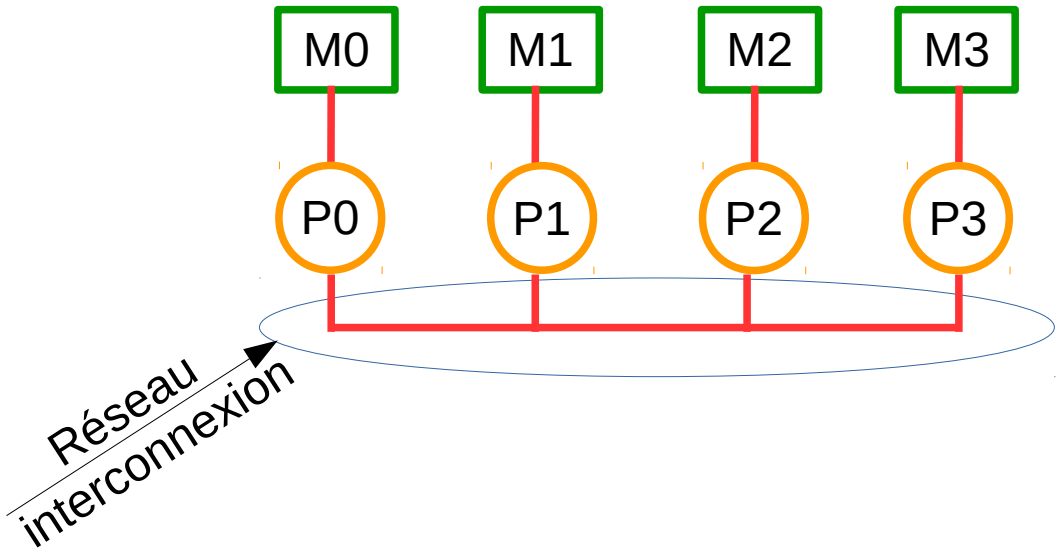
Parallélisme par échange de message

- MPI est une librairie mais pas un langage
- Elle permet le développement d'applications parallèles
- Peut importe la technologie réseau (Ethernet Gigabit, infiniband, ...),
- Il existe plusieurs implémentations de MPI
  - OpenMPI
  - MPICH
  - MVAPICH
  - IntelMPI
  - DeinoMPI

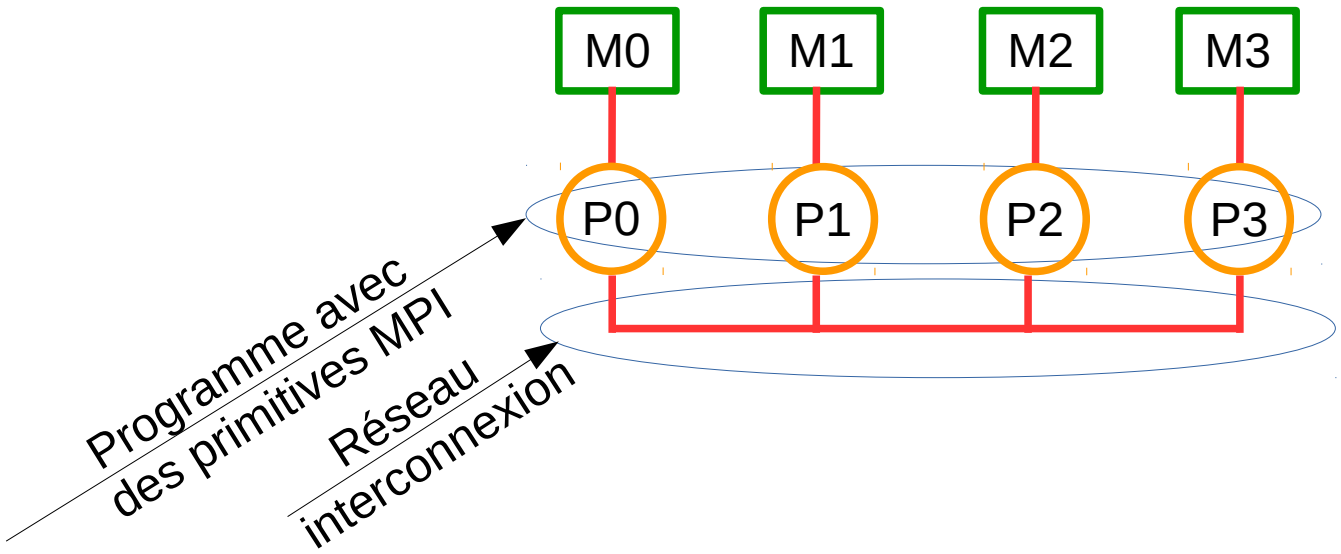
Parallélisme par échange de message



Parallélisme par échange de message



Parallélisme par échange de message



Librairie MPI

Exécution d'un programme parallèle :

- Une primitive MPI permet de dire que le programme qui va s'exécuter est une application parallèle ;
- Elle permet d'initialiser et de terminer les processus parallèles ;

```
mpirun -np <nombre de processus> -machinefile <liste de machines> nom du programme parallèle
```

Librairie MPI

Exemples :

mpirun -np 4 -machinefile machinefile bonjour

mpirun -np 4 bonjour (mpi crée 4 processus indépendants représentant les machines) ← utile pour débugger

Librairie MPI

Exercices (pour ceux qui ont des machines) :

télécharger T8_AP03.zip sur (mettre l'adresse)

décompresser le fichier

Librairie MPI

Regardons bonjour.c :

```
#include <stdio.h>
#include <mpi.h>
// inclusion de l'entête MPI (mpif.h en fortran, mpi.h en C/C++ )

int main(int argc,char *argv[]) {

return(0);
}
```

Librairie MPI

Regardons bonjour.c :

```
#include <mpi.h>
#include <stdio.h>
int main(int argc,char *argv[]) {

MPI_Init(&argc,&argv) ;



return(0);
}
```

Les noms commencent par MPI_

Librairie MPI

Regardons bonjour.c :

```
#include <mpi.h>
#include <stdio.h>
int main(int argc,char *argv[]) {
int error ;
error = MPI_Init(&argc,&argv) ;
```

On peut gérer les erreurs (si pas d'erreur error=MPI_SUCCESS)

```
return(0);
}
```

Librairie MPI

Regardons bonjour.c :

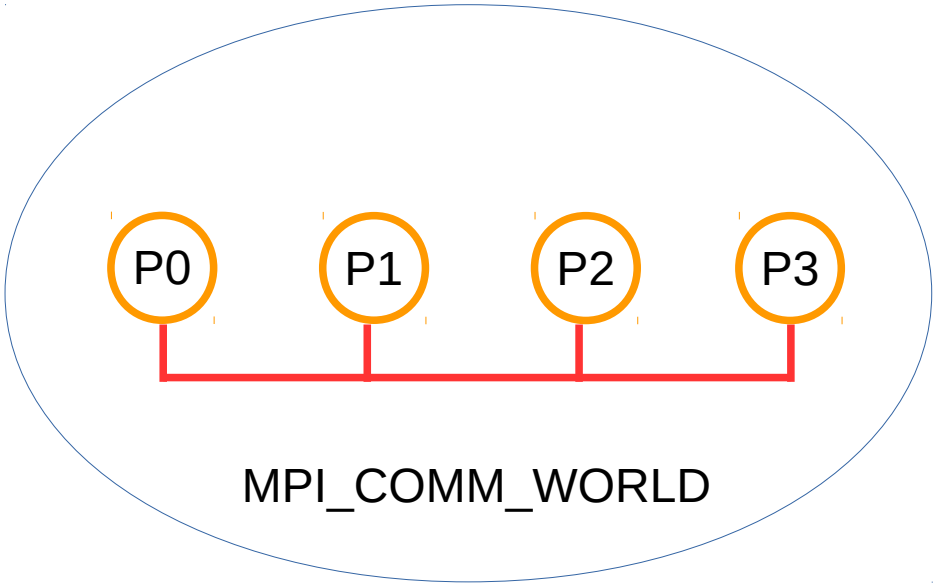
```
#include <mpi.h>
#include <stdio.h>
int main(int argc,char *argv[]) {
int error ;
error = MPI_Init(&argc,&argv) ;
```

← Initialise l'environnement // d'un processeur (communicateur)

```
MPI_Finalize() ;
return(0);
}
```

← Désactive l'environnement // d'un processeur

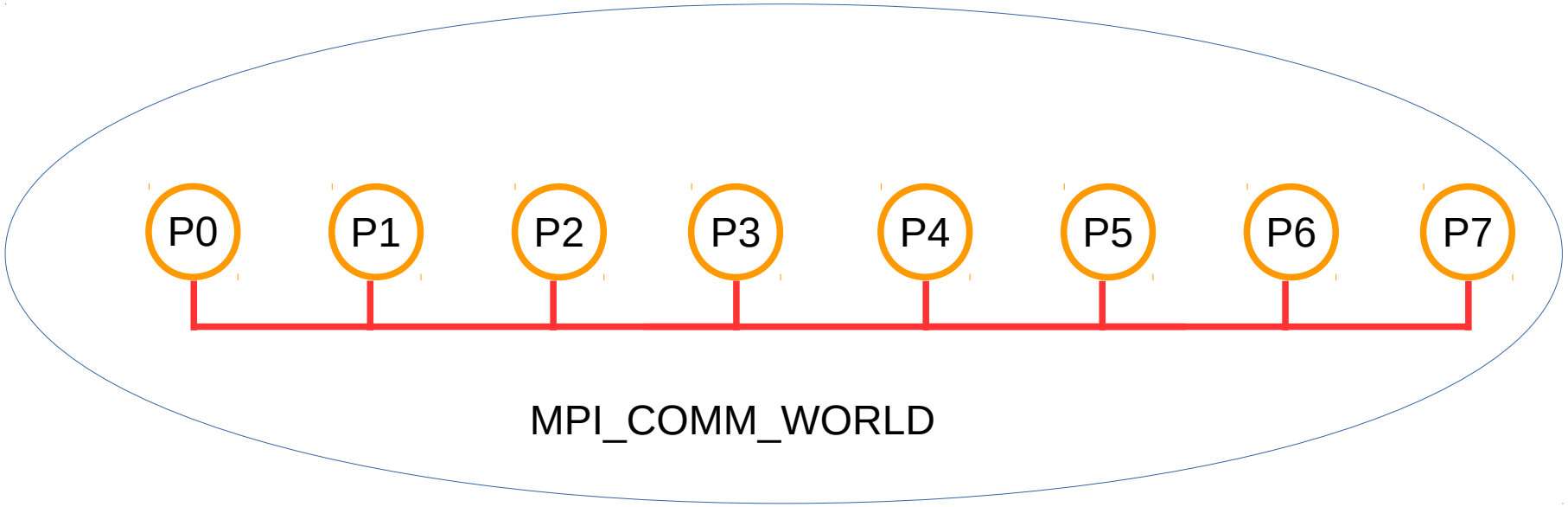
Librairie MPI



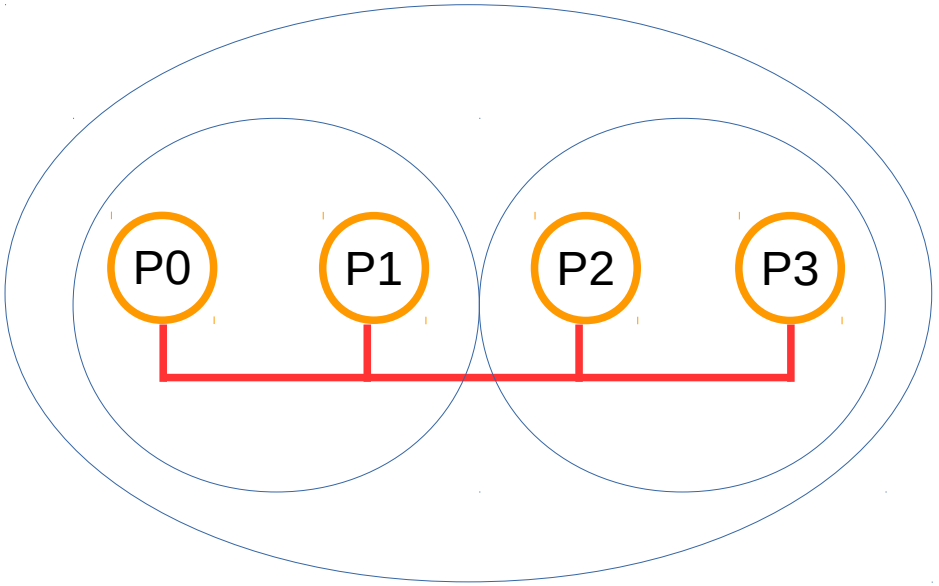
MPI utilise des communicateurs.

Par défaut il y a MPI_COMM_WORLD qui inclut tous les processeurs actifs.

Librairie MPI



Librairie MPI

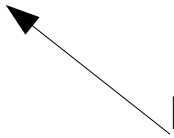


On peut aussi créer des groupes de communicateurs.

Librairie MPI

Regardons bonjour.c :

```
#include <mpi.h>
#include <stdio.h>
int main(int argc,char *argv[]) {
int nbproc ;
MPI_Init(&argc,&argv) ;
MPI_Comm_size(MPI_COMM_WORLD, &nbproc) ;
```



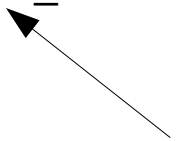
permet à un processeur de connaître le nombre de processeurs dans un communicateur

```
MPI_Finalize() ;
return(0);
}
```

Librairie MPI

Regardons bonjour.c :

```
#include <mpi.h>
#include <stdio.h>
int main(int argc,char *argv[]) {
int nbproc, rang ;
MPI_Init(&argc,&argv) ;
MPI_Comm_size(MPI_COMM_WORLD, &nbproc) ;
MPI_Comm_rank(MPI_COMM_WORLD, &rang) ;
```



permet à un processeur de connaître son rang (0 à nbproc-1)

```
MPI_Finalize() ;
return(0);
}
```

Librairie MPI

Regardons bonjour.c :

```
#include <mpi.h>
#include <stdio.h>
int main(int argc,char *argv[]) {
int nbproc, rang ;
MPI_Init(&argc,&argv) ;
MPI_Comm_size(MPI_COMM_WORLD, &nbproc) ;
MPI_Comm_rank(MPI_COMM_WORLD, &rang) ;
printf("Je suis le processeur de rang =%d\n",rang) ;
MPI_Finalize() ;
return(0);
}
```

Librairie MPI

La version fortran fbonjour.f90 :

```
IMPLICIT NONE
INCLUDE "mpif.h"          ! entête mpi fortran (obligatoire)
INTEGER :: nbproc, infompi, rang
CALL MPI_Init(infompi)
CALL MPI_Comm_size(MPI_COMM_WORLD,nbproc,infompi)
CALL MPI_Comm_rank(MPI_COMM_WORLD,rang,infompi)
WRITE(*,*) "Je suis le processeur de rang = ",rang
CALL MPI_Finalize(infompi)
STOP
END PROGRAM
```

Librairie MPI

Pour compiler, il faut faire :

Pour la version C :

mpicc bonjour.c -o bonjour (mpic++)

Pour la version Fortran :

mpif90 fbonjour.f90 -o fbonjour (mpifort)

Librairie MPI

Si on lance l'execution : mpirun -np 4 ./bonjour

on obtient la première fois :

Je suis le processeur de rang = 1

Je suis le processeur de rang = 2

Je suis le processeur de rang = 3

Je suis le processeur de rang = 0

si on relance :

Je suis le processeur de rang = 3

Je suis le processeur de rang = 0

Je suis le processeur de rang = 1

Je suis le processeur de rang = 2

Librairie MPI

Si on lance l'execution : mpirun -np 4 ./bonjour

on obtient la première fois :

Je suis le processeur de rang = 1

Je suis le processeur de rang = 2

Je suis le processeur de rang = 3

Je suis le processeur de rang = 0

si on relance :

Je suis le processeur de rang = 3

Je suis le processeur de rang = 0

Je suis le processeur de rang = 1

Je suis le processeur de rang = 2

Les processeurs vivent leur vie.

Ils finissent suivant leur vitesse d'exécution.

L'affichage est séquentiel.

Librairie MPI

Regardons bonjour1.c :

```
#include <mpi.h>
#include <stdio.h>
int main(int argc,char *argv[]) {
int nbproc, rang ;
MPI_Init(&argc,&argv) ;
MPI_Comm_size(MPI_COMM_WORLD, &nbproc) ;
MPI_Comm_rank(MPI_COMM_WORLD, &rang) ;
printf("%d dit : Il y a %d processeurs\n",rang,nbproc);
printf("Je suis le processeur de rang =%d\n",rang) ;
MPI_Finalize() ;
return(0);
}
```

Librairie MPI

La version fortran fbonjour1.f90 :

```
IMPLICIT NONE
INCLUDE "mpif.h"          ! entête mpi fortran (obligatoire)
INTEGER :: nbproc, infompi, rang
CALL MPI_Init(infompi)
CALL MPI_Comm_size(MPI_COMM_WORLD,nbproc,infompi)
CALL MPI_Comm_rank(MPI_COMM_WORLD,rang,infompi)
WRITE(*,*) rang,"dit : Il y a ",nbproc," processeurs"
WRITE(*,*) "Je suis le processeur de rang = ",rang
CALL MPI_Finalize(infompi)
STOP
END PROGRAM
```

Librairie MPI

Si on lance l'execution : mpirun -np 4 ./bonjour1

0 dit : Il y a 4 processeurs

Je suis le processeur de rang = 0

1 dit : Il y a 4 processeurs

3 dit : Il y a 4 processeurs

Je suis le processeur de rang = 3

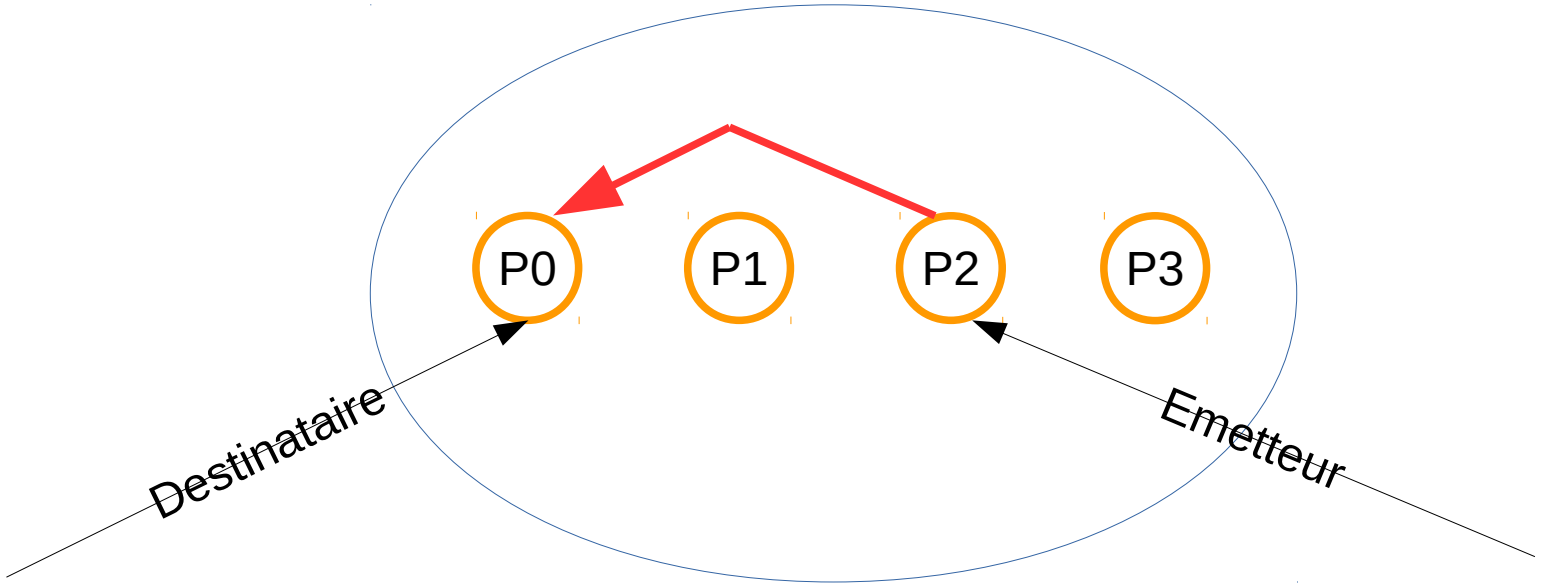
2 dit : Il y a 4 processeurs

Je suis le processeur de rang = 2

Je suis le processeur de rang = 1

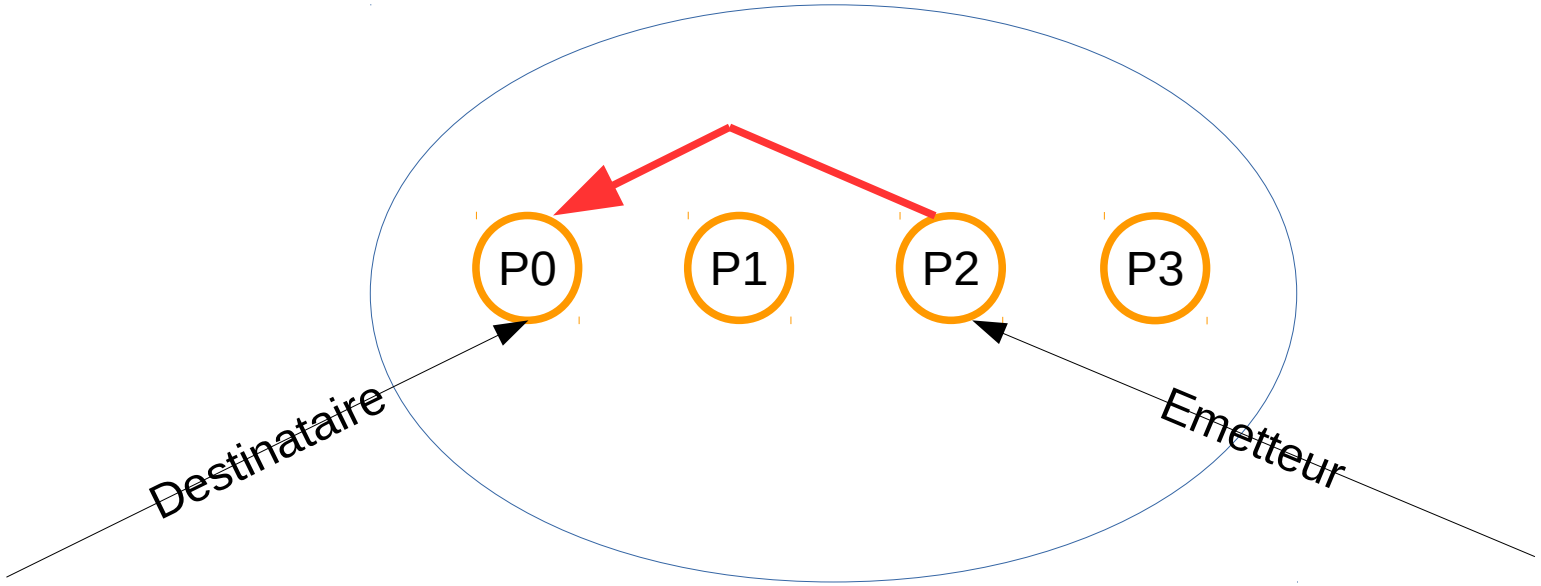
MPI : communications point à point

Une communication est dite point à point si la communication a lieu entre deux processeurs (un émetteur et un destinataire).



MPI : communications point à point

Message est constitué des n° (rang) de l'émetteur et du destinataire, d'une étiquette, d'une donnée (type) et du communicateur.



Plusieurs protocoles de communication

MPI : communications point à point

Une primitive d'envoi ou de réception est dite :

- bloquante si l'espace mémoire servant à la communication peut être réutilisé immédiatement à la fin de son utilisation ;
- non bloquante si elle rend la main immédiatement à la fin de son utilisation. Dans ce cas, pour réutiliser l'espace mémoire servant à la communication, il faudra finaliser la communication (exemple plus loin).

MPI : communications point à point

En C, on a les primitives bloquantes suivantes :

int MPI_Send(void* valeur, int nbvaleur, MPI_Datatype type, int destinataire, int etiquette, MPI_Comm comm)

- valeur : valeur à envoyer
- nbvaleur : nombre de valeur à envoyer
- type : le type de la valeur
- destinataire : le rang du processus destinataire
- etiquette : une étiquette pour identifier le message
- comm : le communicateur

MPI : communications point à point

int MPI_Recv(void* valeur, int nbvaleur, MPI_Datatype type, int emetteur, int etiquette, MPI_Comm comm, MPI_Status status)

- valeur : valeur à recevoir
- nbvaleur : nombre de valeur à recevoir
- type : le type de la valeur
- emetteur : le rang du processus émetteur
- etiquette : une étiquette pour identifier le message
- comm : le communicateur
- status : structure de type MPI_Status avec 3 champs : status.MPI_SOURCE (rang émetteur), status.MPI_TAG (étiquette) et status.MPI_ERROR (code d'erreur)

MPI : communications point à point

En Fortran :

CALL MPI_SEND(type valeur, integer :: nbvaleur, MPI_Datatype type, integer :: destinataire, integer :: etiquette, integer :: comm, integer :: ierr)

- valeur : valeur à recevoir (donner le type de la donnée)
- nbvaleur : nombre de valeur à recevoir
- type : le type de la valeur
- destinataire : le rang du processus destinataire
- etiquette : une étiquette pour identifier le message
- comm : le communicateur
- ierr : gestion des erreurs

MPI : communications point à point

CALL MPI_RECV(type valeur, integer :: nbvaleur, MPI_Datatype type, integer :: emetteur, integer :: etiquette, integer :: comm, dimension(MPI_STATUS_SIZE) :: status, integer :: ierr)

- valeur : valeur à recevoir (donner le type de la donnée)
- nbvaleur : nombre de valeur à recevoir
- type : le type de la valeur
- émetteur : le rang du processus émetteur
- etiquette : une étiquette pour identifier le message
- comm : le communicateur
- status : informations
- ierr : gestion des erreurs

MPI : communications point à point

Type MPI	Type FORTRAN
MPI_INTEGER	INTEGER
MPI_REAL	REAL
MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_COMPLEX	COMPLEX
MPI_DOUBLE_COMPLEX	DOUBLE COMPLEX
MPI_LOGICAL	LOGICAL
MPI_CHARACTER	CHARACTER

Type MPI	Type C
MPI_CHAR	signed char
MPI_SHORT	signed short
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double

MPI : communications point à point

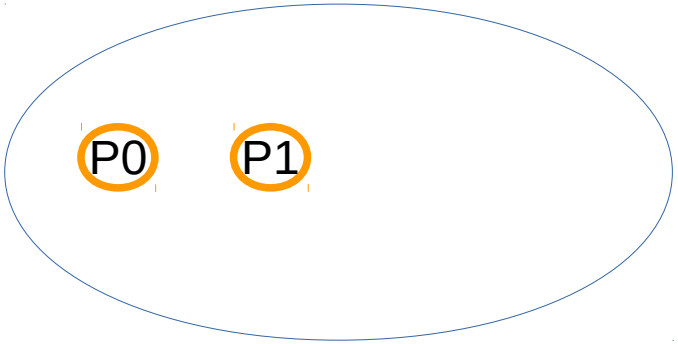
On a la possibilité de recevoir des données

- de n'importe quel processeur : MPI_ANY_SOURCE
- et/ou avec n'importe quelle étiquette : MPI ANY TAG

Il est aussi possible de créer des structures de données complexe.

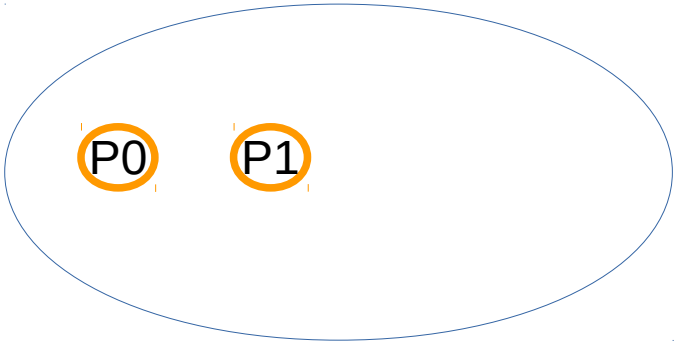
MPI : communications point à point

```
IMPLICIT NONE
INCLUDE "mpif.h"          ! entete mpi fortran (obligatoire)
INTEGER :: nbproc, infompi, rang, status(MPI_STATUS_SIZE),valeur
CALL MPI_Init(infompi)
CALL MPI_Comm_size(MPI_COMM_WORLD,nbproc,infompi)
CALL MPI_Comm_rank(MPI_COMM_WORLD,rang,infompi)
IF (rang .EQ. 0) THEN
  valeur=2017
  WRITE (*,*) rang, " envoie au processeur 1 : ",valeur
  call MPI_SEND(valeur,1,MPI_INTEGER,1,100,MPI_COMM_WORLD,infompi)
ELSE
  call MPI_RECV(valeur,1,MPI_INTEGER,0,100,MPI_COMM_WORLD,status,infompi)
  WRITE (*,*) rang, " vient de recevoir du processeur 0 : ",valeur
ENDIF
CALL MPI_Finalize(infompi)
STOP
END PROGRAM
```



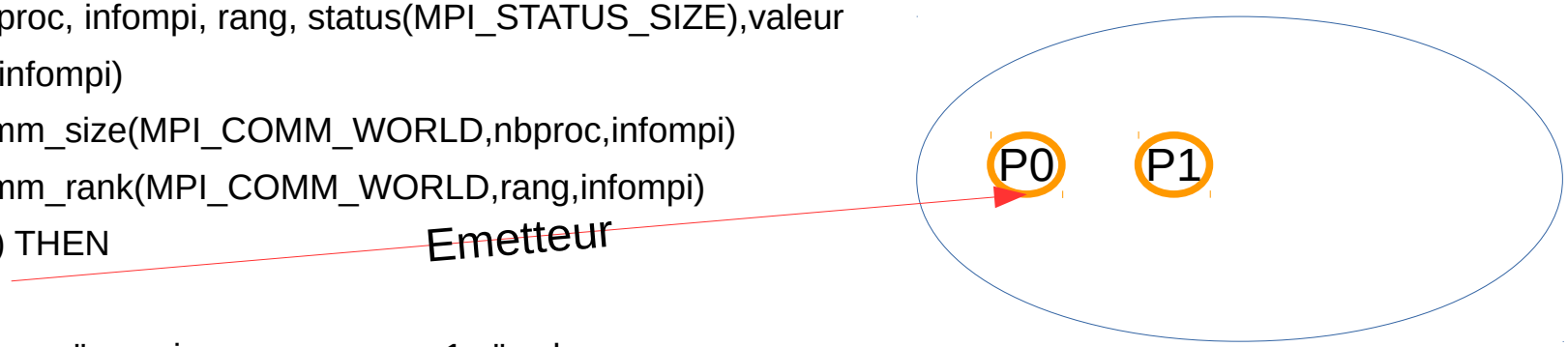
MPI : communications point à point

```
IMPLICIT NONE
INCLUDE "mpif.h"          ! entete mpi fortran (obligatoire)
INTEGER :: nbproc, infompi, rang, status(MPI_STATUS_SIZE),valeur
CALL MPI_Init(infompi)
CALL MPI_Comm_size(MPI_COMM_WORLD,nbproc,infompi)
CALL MPI_Comm_rank(MPI_COMM_WORLD,rang,infompi)
IF (rang .EQ. 0) THEN
  valeur=2017
  WRITE (*,*) rang, " envoie au processeur 1 : ",valeur
  call MPI_SEND(valeur,1,MPI_INTEGER,1,100,MPI_COMM_WORLD,infompi)
ELSE
  call MPI_RECV(valeur,1,MPI_INTEGER,0,100,MPI_COMM_WORLD,status,infompi)
  WRITE (*,*) rang, " vient de recevoir du processeur 0 : ",valeur
ENDIF
CALL MPI_Finalize(infompi)
STOP
END PROGRAM
```



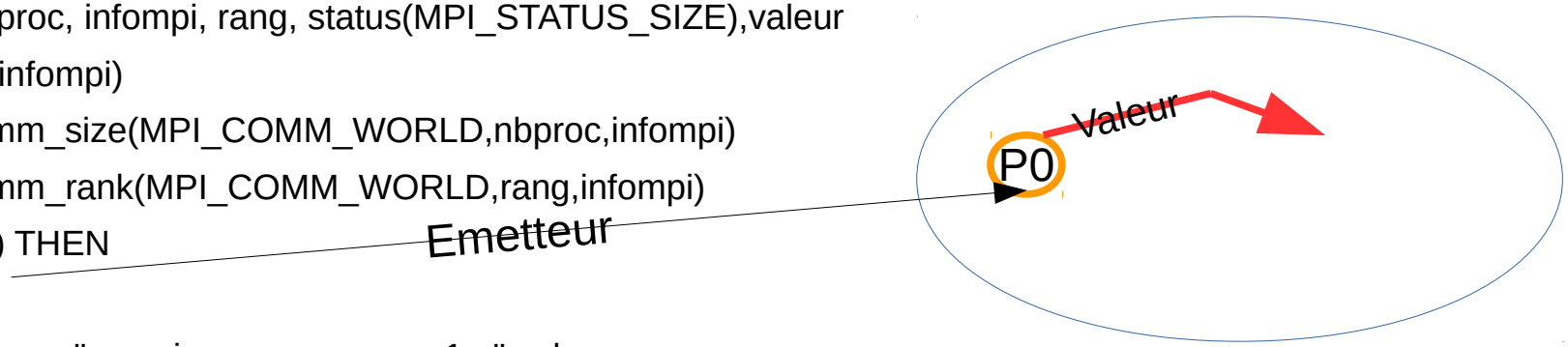
MPI : communications point à point

```
IMPLICIT NONE
INCLUDE "mpif.h"          ! entete mpi fortran (obligatoire)
INTEGER :: nbproc, infompi, rang, status(MPI_STATUS_SIZE),valeur
CALL MPI_Init(infompi)
CALL MPI_Comm_size(MPI_COMM_WORLD,nbproc,infompi)
CALL MPI_Comm_rank(MPI_COMM_WORLD,rang,infompi)
IF (rang .EQ. 0) THEN
  valeur=2017
  WRITE (*,*) rang, " envoie au processeur 1 : ",valeur
  call MPI_SEND(valeur,1,MPI_INTEGER,1,100,MPI_COMM_WORLD,infompi)
ELSE
  call MPI_RECV(valeur,1,MPI_INTEGER,0,100,MPI_COMM_WORLD,status,infompi)
  WRITE (*,*) rang, " vient de recevoir du processeur 0 : ",valeur
ENDIF
CALL MPI_Finalize(infompi)
STOP
END PROGRAM
```



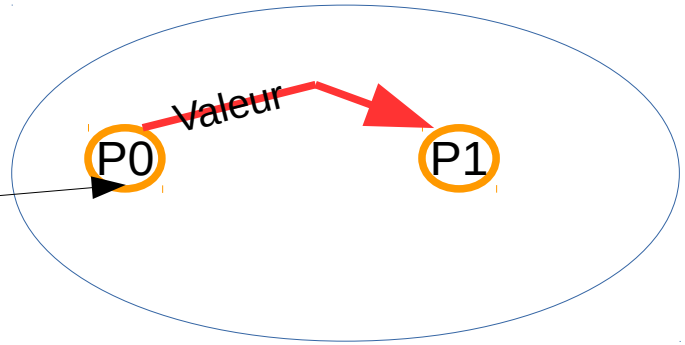
MPI : communications point à point

```
IMPLICIT NONE
INCLUDE "mpif.h"          ! entete mpi fortran (obligatoire)
INTEGER :: nbproc, infompi, rang, status(MPI_STATUS_SIZE),valeur
CALL MPI_Init(infompi)
CALL MPI_Comm_size(MPI_COMM_WORLD,nbproc,infompi)
CALL MPI_Comm_rank(MPI_COMM_WORLD,rang,infompi)
IF (rang .EQ. 0) THEN
  valeur=2017
  WRITE (*,*) rang, " envoie au processeur 1 : ",valeur
  call MPI_SEND(valeur,1,MPI_INTEGER,1,100,MPI_COMM_WORLD,infompi)
ELSE
  call MPI_RECV(valeur,1,MPI_INTEGER,0,100,MPI_COMM_WORLD,status,infompi)
  WRITE (*,*) rang, " vient de recevoir du processeur 0 : ",valeur
ENDIF
CALL MPI_Finalize(infompi)
STOP
END PROGRAM
```



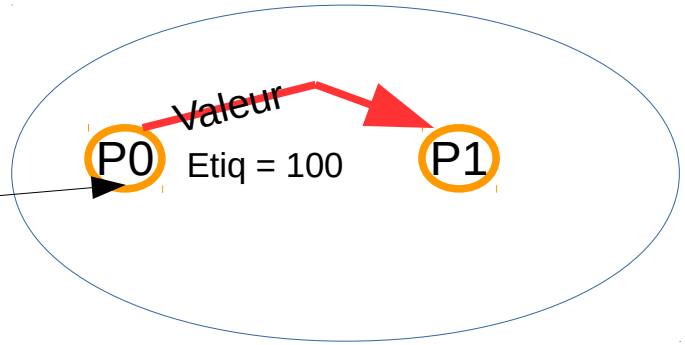
MPI : communications point à point

```
IMPLICIT NONE
INCLUDE "mpif.h"          ! entete mpi fortran (obligatoire)
INTEGER :: nbproc, infompi, rang, status(MPI_STATUS_SIZE),valeur
CALL MPI_Init(infompi)
CALL MPI_Comm_size(MPI_COMM_WORLD,nbproc,infompi)
CALL MPI_Comm_rank(MPI_COMM_WORLD,rang,infompi)
IF (rang .EQ. 0) THEN
  valeur=2017
  WRITE (*,*) rang, " envoie au processeur 1 : ",valeur
  call MPI_SEND(valeur,1,MPI_INTEGER,1,100,MPI_COMM_WORLD,infompi)
ELSE
  call MPI_RECV(valeur,1,MPI_INTEGER,0,100,MPI_COMM_WORLD,status,infompi)
  WRITE (*,*) rang, " vient de recevoir du processeur 0 : ",valeur
ENDIF
CALL MPI_Finalize(infompi)
STOP
END PROGRAM
```



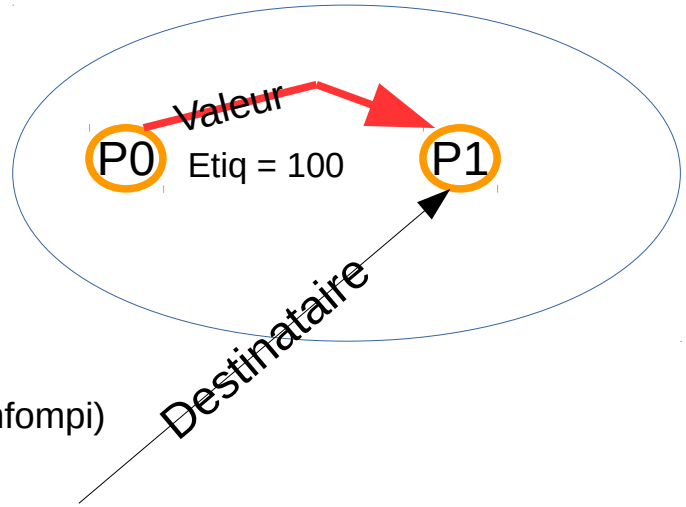
MPI : communications point à point

```
IMPLICIT NONE
INCLUDE "mpif.h"          ! entete mpi fortran (obligatoire)
INTEGER :: nbproc, infompi, rang, status(MPI_STATUS_SIZE),valeur
CALL MPI_Init(infompi)
CALL MPI_Comm_size(MPI_COMM_WORLD,nbproc,infompi)
CALL MPI_Comm_rank(MPI_COMM_WORLD,rang,infompi)
IF (rang .EQ. 0) THEN
  valeur=2017
  WRITE (*,*) rang, " envoie au processeur 1 : ",valeur
  call MPI_SEND(valeur,1,MPI_INTEGER,1,100,MPI_COMM_WORLD,infompi)
ELSE
  call MPI_RECV(valeur,1,MPI_INTEGER,0,100,MPI_COMM_WORLD,status,infompi)
  WRITE (*,*) rang, " vient de recevoir du processeur 0 : ",valeur
ENDIF
CALL MPI_Finalize(infompi)
STOP
END PROGRAM
```



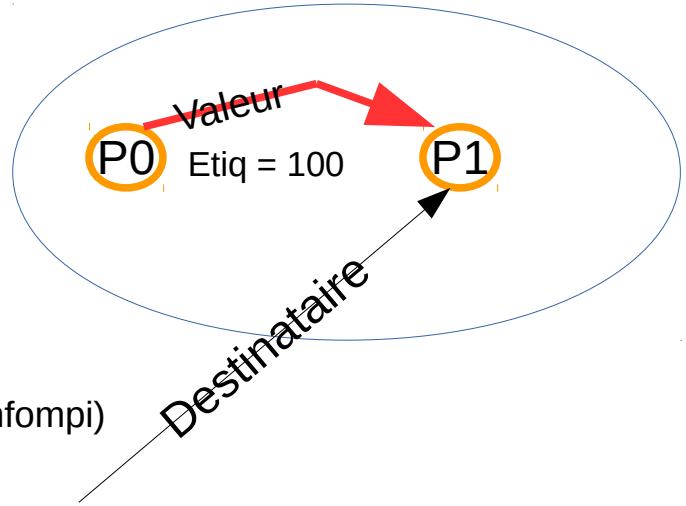
MPI : communications point à point

```
IMPLICIT NONE
INCLUDE "mpif.h"          ! entete mpi fortran (obligatoire)
INTEGER :: nbproc, infompi, rang, status(MPI_STATUS_SIZE),valeur
CALL MPI_Init(infompi)
CALL MPI_Comm_size(MPI_COMM_WORLD,nbproc,infompi)
CALL MPI_Comm_rank(MPI_COMM_WORLD,rang,infompi)
IF (rang .EQ. 0) THEN
  valeur=2017
  WRITE (*,*) rang, " envoie au processeur 1 : ",valeur
  call MPI_SEND(valeur,1,MPI_INTEGER,1,100,MPI_COMM_WORLD,infompi)
ELSE
  call MPI_RECV(valeur,1,MPI_INTEGER,0,100,MPI_COMM_WORLD,status,infompi)
  WRITE (*,*) rang, " vient de recevoir du processeur 0 : ",valeur
ENDIF
CALL MPI_Finalize(infompi)
STOP
END PROGRAM
```



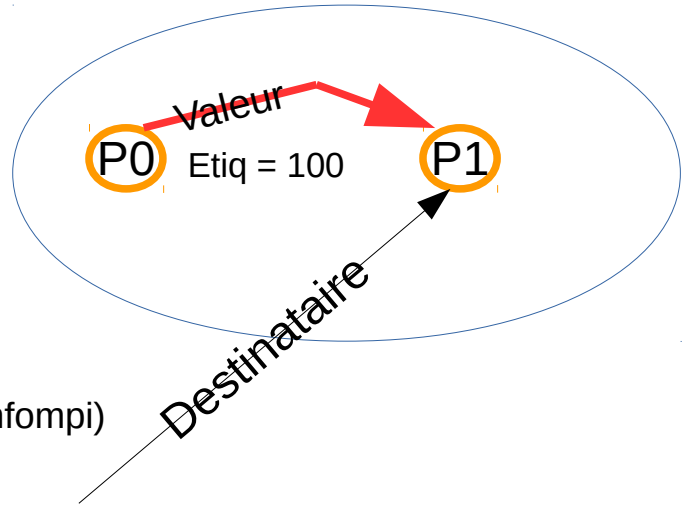
MPI : communications point à point

```
IMPLICIT NONE
INCLUDE "mpif.h"          ! entete mpi fortran (obligatoire)
INTEGER :: nbproc, infompi, rang, status(MPI_STATUS_SIZE),valeur
CALL MPI_Init(infompi)
CALL MPI_Comm_size(MPI_COMM_WORLD,nbproc,infompi)
CALL MPI_Comm_rank(MPI_COMM_WORLD,rang,infompi)
IF (rang .EQ. 0) THEN
  valeur=2017
  WRITE (*,*) rang, " envoie au processeur 1 : ",valeur
  call MPI_SEND(valeur,1,MPI_INTEGER,1,100,MPI_COMM_WORLD,infompi)
ELSE
  call MPI_RECV(valeur,1,MPI_INTEGER,0,100,MPI_COMM_WORLD,status,infompi)
  WRITE (*,*) rang, " vient de recevoir du processeur 0 : ",valeur
ENDIF
CALL MPI_Finalize(infompi)
STOP
END PROGRAM
```



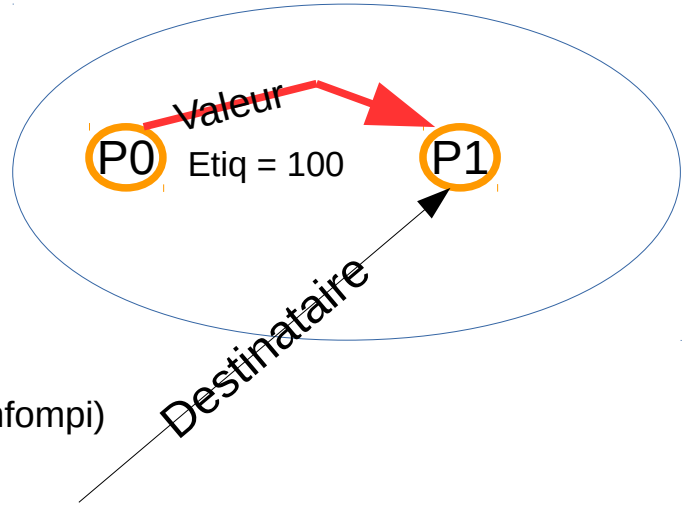
MPI : communications point à point

```
IMPLICIT NONE
INCLUDE "mpif.h"          ! entete mpi fortran (obligatoire)
INTEGER :: nbproc, infompi, rang, status(MPI_STATUS_SIZE),valeur
CALL MPI_Init(infompi)
CALL MPI_Comm_size(MPI_COMM_WORLD,nbproc,infompi)
CALL MPI_Comm_rank(MPI_COMM_WORLD,rang,infompi)
IF (rang .EQ. 0) THEN
  valeur=2017
  WRITE (*,*) rang, " envoie au processeur 1 : ",valeur
  call MPI_SEND(valeur,1,MPI_INTEGER,1,100,MPI_COMM_WORLD,infompi)
ELSE
  call MPI_RECV(valeur,1,MPI_INTEGER,0,100,MPI_COMM_WORLD,status,infompi)
  WRITE (*,*) rang, " vient de recevoir du processeur 0 : ",valeur
ENDIF
CALL MPI_Finalize(infompi)
STOP
END PROGRAM
```



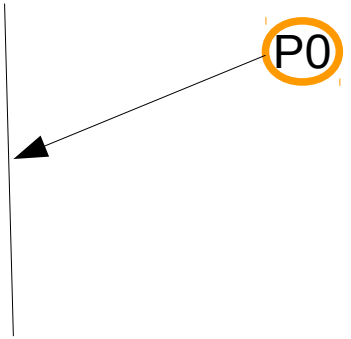
MPI : communications point à point

```
IMPLICIT NONE
INCLUDE "mpif.h"          ! entete mpi fortran (obligatoire)
INTEGER :: nbproc, infompi, rang, status(MPI_STATUS_SIZE),valeur
CALL MPI_Init(infompi)
CALL MPI_Comm_size(MPI_COMM_WORLD,nbproc,infompi)
CALL MPI_Comm_rank(MPI_COMM_WORLD,rang,infompi)
IF (rang .EQ. 0) THEN
  valeur=2017
  WRITE (*,*) rang, " envoie au processeur 1 : ",valeur
  call MPI_SEND(valeur,1,MPI_INTEGER,1,100,MPI_COMM_WORLD,infompi)
ELSE
  call MPI_RECV(valeur,1,MPI_INTEGER,0,100,MPI_COMM_WORLD,status,infompi)
  WRITE (*,*) rang, " vient de recevoir du processeur 0 : ",valeur
ENDIF
CALL MPI_Finalize(infompi)
STOP
END PROGRAM
```



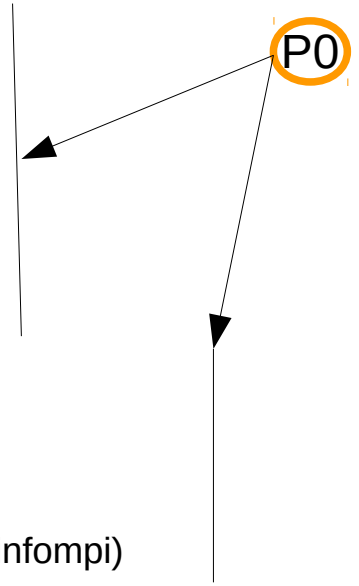
MPI : communications point à point

```
IMPLICIT NONE
INCLUDE "mpif.h"          ! entete mpi fortran (obligatoire)
INTEGER :: nbproc, infompi, rang, status(MPI_STATUS_SIZE),valeur
CALL MPI_Init(infompi)
CALL MPI_Comm_size(MPI_COMM_WORLD,nbproc,infompi)
CALL MPI_Comm_rank(MPI_COMM_WORLD,rang,infompi)
IF (rang .EQ. 0) THEN
  valeur=2017
  WRITE (*,*) rang, " envoie au processeur 1 : ",valeur
  call MPI_SEND(valeur,1,MPI_INTEGER,1,100,MPI_COMM_WORLD,infompi)
ELSE
  call MPI_RECV(valeur,1,MPI_INTEGER,0,100,MPI_COMM_WORLD,status,infompi)
  WRITE (*,*) rang, " vient de recevoir du processeur 0 : ",valeur
ENDIF
CALL MPI_Finalize(infompi)
STOP
END PROGRAM
```



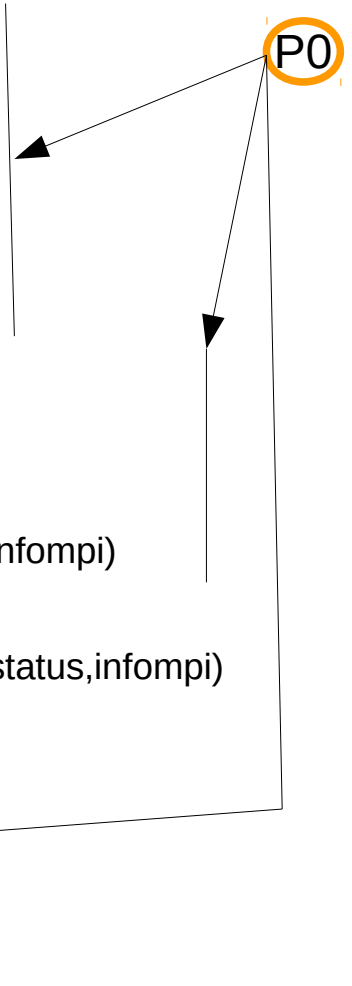
MPI : communications point à point

```
IMPLICIT NONE
INCLUDE "mpif.h"          ! entete mpi fortran (obligatoire)
INTEGER :: nbproc, infompi, rang, status(MPI_STATUS_SIZE),valeur
CALL MPI_Init(infompi)
CALL MPI_Comm_size(MPI_COMM_WORLD,nbproc,infompi)
CALL MPI_Comm_rank(MPI_COMM_WORLD,rang,infompi)
IF (rang .EQ. 0) THEN
  valeur=2017
  WRITE (*,*) rang, " envoie au processeur 1 : ",valeur
  call MPI_SEND(valeur,1,MPI_INTEGER,1,100,MPI_COMM_WORLD,infompi)
ELSE
  call MPI_RECV(valeur,1,MPI_INTEGER,0,100,MPI_COMM_WORLD,status,infompi)
  WRITE (*,*) rang, " vient de recevoir du processeur 0 : ",valeur
ENDIF
CALL MPI_Finalize(infompi)
STOP
END PROGRAM
```



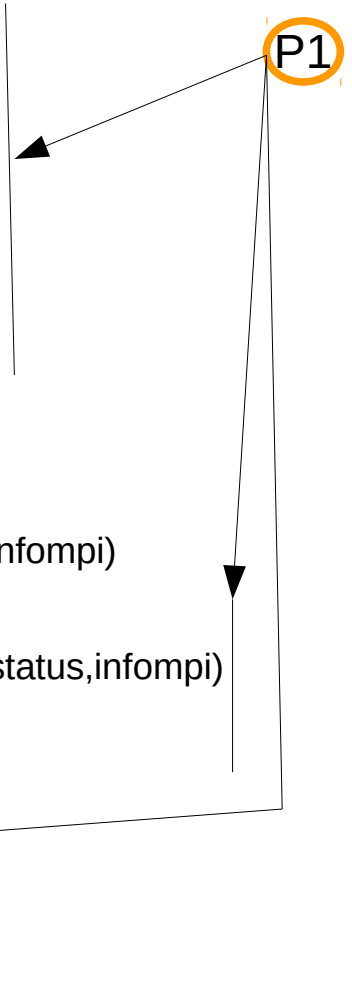
MPI : communications point à point

```
IMPLICIT NONE
INCLUDE "mpif.h"          ! entete mpi fortran (obligatoire)
INTEGER :: nbproc, infompi, rang, status(MPI_STATUS_SIZE),valeur
CALL MPI_Init(infompi)
CALL MPI_Comm_size(MPI_COMM_WORLD,nbproc,infompi)
CALL MPI_Comm_rank(MPI_COMM_WORLD,rang,infompi)
IF (rang .EQ. 0) THEN
  valeur=2017
  WRITE (*,*) rang, " envoie au processeur 1 : ",valeur
  call MPI_SEND(valeur,1,MPI_INTEGER,1,100,MPI_COMM_WORLD,infompi)
ELSE
  call MPI_RECV(valeur,1,MPI_INTEGER,0,100,MPI_COMM_WORLD,status,infompi)
  WRITE (*,*) rang, " vient de recevoir du processeur 0 : ",valeur
ENDIF
CALL MPI_Finalize(infompi)
STOP
END PROGRAM
```



MPI : communications point à point

```
IMPLICIT NONE
INCLUDE "mpif.h"          ! entete mpi fortran (obligatoire)
INTEGER :: nbproc, infompi, rang, status(MPI_STATUS_SIZE),valeur
CALL MPI_Init(infompi)
CALL MPI_Comm_size(MPI_COMM_WORLD,nbproc,infompi)
CALL MPI_Comm_rank(MPI_COMM_WORLD,rang,infompi)
IF (rang .EQ. 0) THEN
  valeur=2017
  WRITE (*,*) rang, " envoie au processeur 1 : ",valeur
  call MPI_SEND(valeur,1,MPI_INTEGER,1,100,MPI_COMM_WORLD,infompi)
ELSE
  call MPI_RECV(valeur,1,MPI_INTEGER,0,100,MPI_COMM_WORLD,status,infompi)
  WRITE (*,*) rang, " vient de recevoir du processeur 0 : ",valeur
ENDIF
CALL MPI_Finalize(infompi)
STOP
END PROGRAM
```



MPI : communications point à point

```
#include <mpi.h>
#include <stdio.h>
int main(int argc,char *argv[]) {
int nbproc, rang, valeur;
MPI_Status status;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD, &nbproc);
MPI_Comm_rank(MPI_COMM_WORLD, &rang);
if (rang == 0)  {
  valeur=2017;
  printf("%d envoie au processeur 1 : %d \n",rang,valeur);
  MPI_Send(&valeur,1,MPI_INT,1,100,MPI_COMM_WORLD); }
else if (rang == 1) {
  MPI_Recv(&valeur,1,MPI_INT,0,100,MPI_COMM_WORLD,&status);
  printf("%d vient de recevoir du processeur 0 : %d \n",rang,valeur); }
MPI_Finalize();
return(0); }
```

MPI : communications point à point

mpirun -np 2 ./commptapt ou ./fcommptapt

on obtient :

0 envoie au processeur 1 : 2017

1 vient de recevoir du processeur 0 : 2017

MPI : communications point à point

```
...
char valeur[5000];
...
if (rang == 0) {
   valeur[0]='a' ;
   printf("%d envoie au processeur 1 : %c \n",rang,valeur[0]);
   MPI_Send(valeur,5000,MPI_CHAR,1,100,MPI_COMM_WORLD);
   MPI_Recv(valeur,5000,MPI_CHAR,1,100,MPI_COMM_WORLD,&status);
   printf("%d vient de recevoir du processeur 1 : %c \n",rang,valeur[0]);
}
else if (rang == 1) {
   valeur[0]='b';
   printf("%d envoie au processeur 0 : %c \n",rang,valeur[0]);
   MPI_Send(valeur,5000,MPI_CHAR,0,100,MPI_COMM_WORLD);
   MPI_Recv(valeur,5000,MPI_CHAR,0,100,MPI_COMM_WORLD,&status);
   printf("%d vient de recevoir du processeur 0 : %c \n",rang,valeur[0]);
}
...
```

MPI : communications point à point

mpirun -np 2 ./commptapt1 ou ./fcommptapt1

on obtient :

0 envoie au processeur 1 : a

1 envoie au processeur 0 : b

puis blocage …. pourquoi ?

MPI : communications point à point

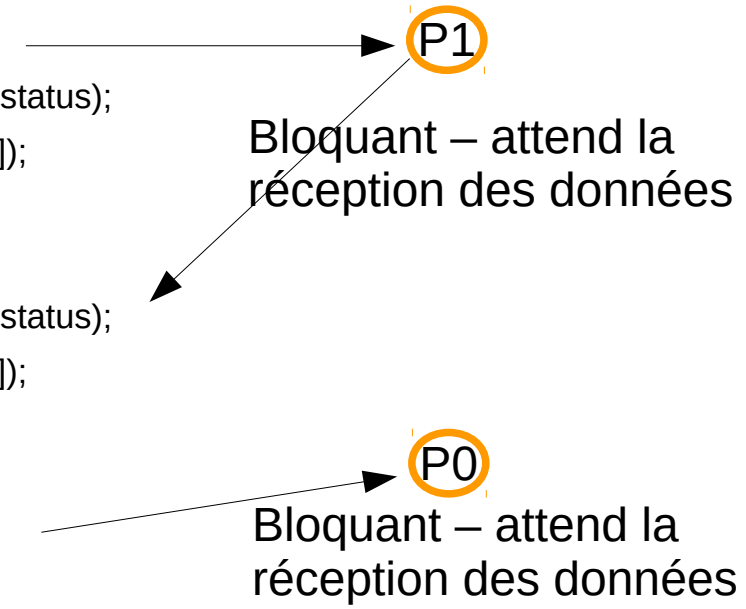
```
...
char valeur[5000];
...
if (rang == 0) {
   valeur[0]='a' ;
   printf("%d envoie au processeur 1 : %c \n",rang,valeur[0]);
   MPI_Send(valeur,5000,MPI_CHAR,1,100,MPI_COMM_WORLD);
   MPI_Recv(valeur,5000,MPI_CHAR,1,100,MPI_COMM_WORLD,&status);
   printf("%d vient de recevoir du processeur 1 : %c \n",rang,valeur[0]);
}
else if (rang == 1) {
   valeur[0]='b';
   printf("%d envoie au processeur 0 : %c \n",rang,valeur[0]);
   MPI_Send(valeur,5000,MPI_CHAR,0,100,MPI_COMM_WORLD);
   MPI_Recv(valeur,5000,MPI_CHAR,0,100,MPI_COMM_WORLD,&status);
   printf("%d vient de recevoir du processeur 0 : %c \n",rang,valeur[0]);
}
...
```

Bloquant – attend la réception des données

Bloquant – attend la réception des données

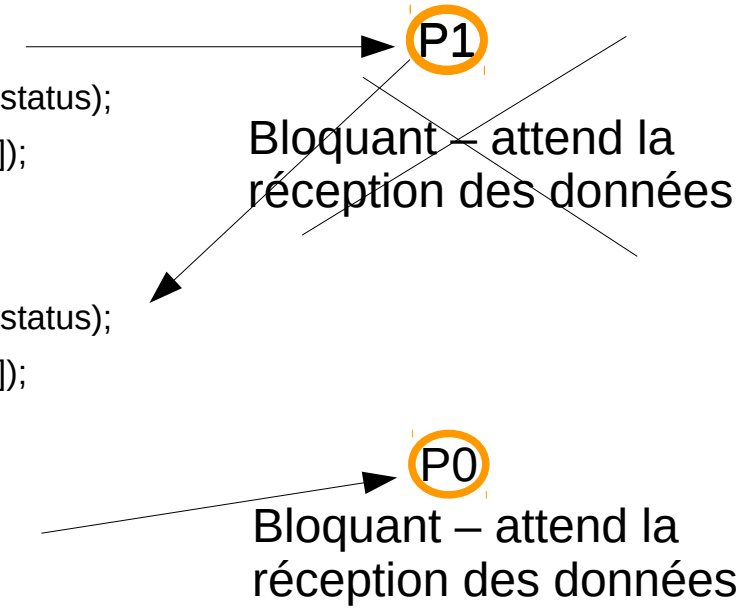
MPI : communications point à point

```
...
char valeur[5000];
...
if (rang == 0) {
   valeur[0]='a' ;
   printf("%d envoie au processeur 1 : %c \n",rang,valeur[0]);
   MPI_Send(valeur,5000,MPI_CHAR,1,100,MPI_COMM_WORLD);
   MPI_Recv(valeur,5000,MPI_CHAR,1,100,MPI_COMM_WORLD,&status);
   printf("%d vient de recevoir du processeur 1 : %c \n",rang,valeur[0]);
}
else if (rang == 1) {
   MPI_Recv(valeur,5000,MPI_CHAR,0,100,MPI_COMM_WORLD,&status);
   printf("%d vient de recevoir du processeur 0 : %c \n",rang,valeur[0]);
   valeur[0]='b';
   printf("%d envoie au processeur 0 : %c \n",rang,valeur[0]);
   MPI_Send(valeur,5000,MPI_CHAR,0,100,MPI_COMM_WORLD) ;
}
...
```



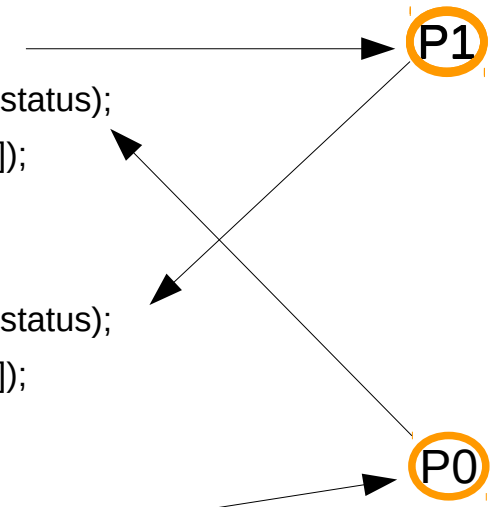
MPI : communications point à point

```
...
char valeur[5000];
...
if (rang == 0) {
   valeur[0]='a' ;
   printf("%d envoie au processeur 1 : %c \n",rang,valeur[0]);
   MPI_Send(valeur,5000,MPI_CHAR,1,100,MPI_COMM_WORLD);
   MPI_Recv(valeur,5000,MPI_CHAR,1,100,MPI_COMM_WORLD,&status);
   printf("%d vient de recevoir du processeur 1 : %c \n",rang,valeur[0]);
}
else if (rang == 1) {
   MPI_Recv(valeur,5000,MPI_CHAR,0,100,MPI_COMM_WORLD,&status);
   printf("%d vient de recevoir du processeur 0 : %c \n",rang,valeur[0]);
   valeur[0]='b';
   printf("%d envoie au processeur 0 : %c \n",rang,valeur[0]);
   MPI_Send(valeur,5000,MPI_CHAR,0,100,MPI_COMM_WORLD) ;
}
...
```



MPI : communications point à point

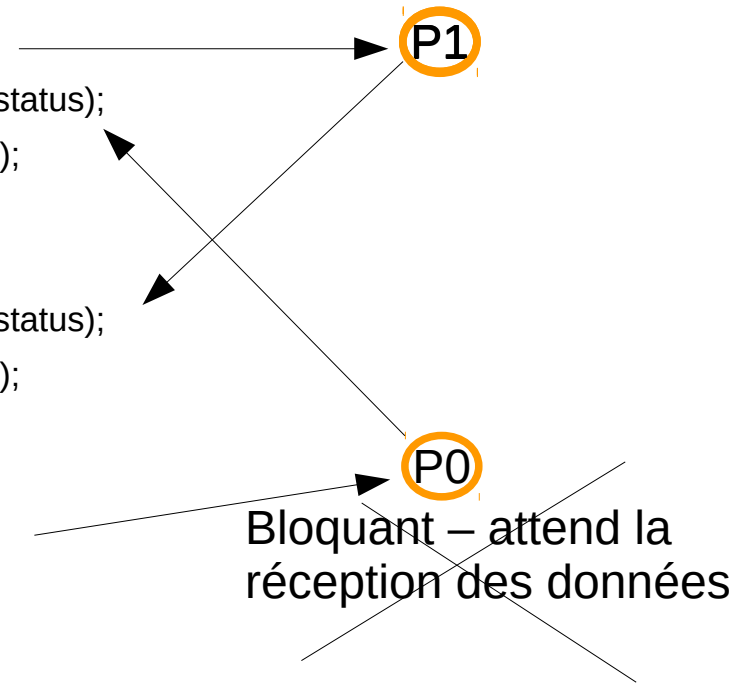
```
...
char valeur[5000];
...
if (rang == 0) {
   valeur[0]='a' ;
   printf("%d envoie au processeur 1 : %c \n",rang,valeur[0]);
   MPI_Send(valeur,5000,MPI_CHAR,1,100,MPI_COMM_WORLD);
   MPI_Recv(valeur,5000,MPI_CHAR,1,100,MPI_COMM_WORLD,&status);
   printf("%d vient de recevoir du processeur 1 : %c \n",rang,valeur[0]);
}
else if (rang == 1) {
   MPI_Recv(valeur,5000,MPI_CHAR,0,100,MPI_COMM_WORLD,&status);
   printf("%d vient de recevoir du processeur 0 : %c \n",rang,valeur[0]);
   valeur[0]='b';
   printf("%d envoie au processeur 0 : %c \n",rang,valeur[0]);
   MPI_Send(valeur,5000,MPI_CHAR,0,100,MPI_COMM_WORLD) ;
}
...
```



Bloquant – attend la réception des données

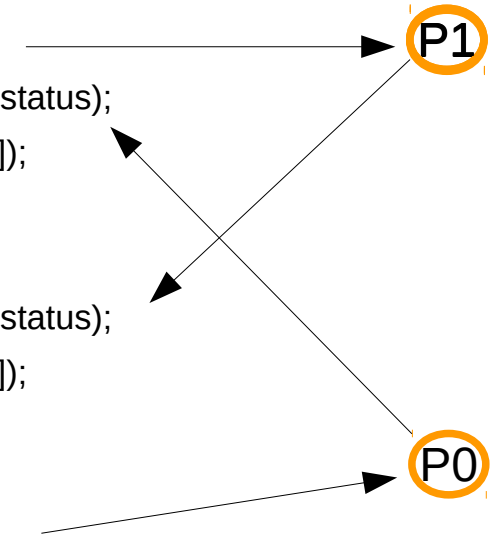
MPI : communications point à point

```
...
char valeur[5000];
...
if (rang == 0) {
   valeur[0]='a' ;
   printf("%d envoie au processeur 1 : %c \n",rang,valeur[0]);
   MPI_Send(valeur,5000,MPI_CHAR,1,100,MPI_COMM_WORLD);
   MPI_Recv(valeur,5000,MPI_CHAR,1,100,MPI_COMM_WORLD,&status);
   printf("%d vient de recevoir du processeur 1 : %c \n",rang,valeur[0]);
}
else if (rang == 1) {
   MPI_Recv(valeur,5000,MPI_CHAR,0,100,MPI_COMM_WORLD,&status);
   printf("%d vient de recevoir du processeur 0 : %c \n",rang,valeur[0]);
   valeur[0]='b';
   printf("%d envoie au processeur 0 : %c \n",rang,valeur[0]);
   MPI_Send(valeur,5000,MPI_CHAR,0,100,MPI_COMM_WORLD) ;
}
...
```



MPI : communications point à point

```
...
char valeur[5000];
...
if (rang == 0) {
   valeur[0]='a' ;
   printf("%d envoie au processeur 1 : %c \n",rang,valeur[0]);
   MPI_Send(valeur,5000,MPI_CHAR,1,100,MPI_COMM_WORLD);
   MPI_Recv(valeur,5000,MPI_CHAR,1,100,MPI_COMM_WORLD,&status);
   printf("%d vient de recevoir du processeur 1 : %c \n",rang,valeur[0]);
}
else if (rang == 1) {
   MPI_Recv(valeur,5000,MPI_CHAR,0,100,MPI_COMM_WORLD,&status);
   printf("%d vient de recevoir du processeur 0 : %c \n",rang,valeur[0]);
   valeur[0]='b';
   printf("%d envoie au processeur 0 : %c \n",rang,valeur[0]);
   MPI_Send(valeur,5000,MPI_CHAR,0,100,MPI_COMM_WORLD) ;
}
...
```



MPI : communications point à point

mpirun -np 2 ./commptapt2 ou ./fcommptapt2

on obtient :

0 envoie au processeur 1 : a

1 vient de recevoir du processeur 0 : a

1 envoie au processeur 0 : b

0 vient de recevoir du processeur 1 : b

MPI : communications point à point

- Il est important de tenir compte de l'ordre des opérations d'envoi ou de réception car on peut arriver à une situation d'interblocage !!!

MPI : communications point à point

Acheminement des messages

MPI garantit qu'entre un même émetteur et un même destinataire, les messages sont transmis dans l'ordre. Les opérations de réception pour un même émetteur sont aussi traitées dans l'ordre.

Cependant le réseau peut mélanger des messages d'émetteurs ou de destinataires différents.

MPI : communications point à point

Envoi et réception combinés

On peut effectuer un envoi et une réception en une seule communication avec la primitive MPI_SENDRECV

On peut échanger des données en une seule communication MPI_SENDRECV_REPLACE

MPI : communications point à point

Envoi et réception combinés

MPI SENDRECV exécute une envoi bloquant et une réception bloquante. Les deux doivent utiliser le même groupe de processus, mais doivent avoir des étiquettes différentes.

Le buffer d'envoi et le buffer de réception doivent être disjoints et peuvent avoir des longueurs et types différents.

MPI : communications point à point

Mesure du temps d'exécution

double MPI_Wtime (void) – MPI_Wtime()

Retourne le nombre de secondes passées depuis un moment arbitraire dans le passé.

La différence entre deux valeurs retournées par cette fonction à deux endroits différents dans le code donne le nombre de secondes qui ont passé entre ces deux endroits.

MPI : communications point à point

```
CALL MPI_Init(infompi)
CALL MPI_Comm_size(MPI_COMM_WORLD,nbproc,infompi)
CALL MPI_Comm_rank(MPI_COMM_WORLD,rang,infompi)

IF (rang .EQ. 0) THEN
  valeur(1)='a'
  WRITE (*,*) rang, " envoie au processeur 1 : ",valeur(1)
  call MPI_SEND(valeur(1),5000,MPI_CHARACTER,1,100,MPI_COMM_WORLD,infompi)
  call MPI_RECV(valeur(1),5000,MPI_CHARACTER,1,101,MPI_COMM_WORLD,status,infompi)
  WRITE (*,*) rang, " vient de recevoir du processeur 1 : ",valeur(1)
ELSE
   call MPI_RECV(valeur(1),5000,MPI_CHARACTER,0,100,MPI_COMM_WORLD,status,infompi)
  WRITE (*,*) rang, " vient de recevoir du processeur 0 : ",valeur(1)
  valeur(1)='b'
  WRITE (*,*) rang, " envoie au processeur 0 : ",valeur(1)
  call MPI_SEND(valeur(1),5000,MPI_CHARACTER,0,101,MPI_COMM_WORLD,infompi)
ENDIF


CALL MPI_Finalize(infompi)
```

MPI : communications point à point

```
REAL*8 :: debut, duree
CALL MPI_Init(infompi)
CALL MPI_Comm_size(MPI_COMM_WORLD,nbproc,infompi)
CALL MPI_Comm_rank(MPI_COMM_WORLD,rang,infompi)

IF (rang .EQ. 0) THEN
  valeur(1)='a'
  WRITE (*,*) rang, " envoie au processeur 1 : ",valeur(1)
  call MPI_SEND(valeur(1),5000,MPI_CHARACTER,1,100,MPI_COMM_WORLD,infompi)
  call MPI_RECV(valeur(1),5000,MPI_CHARACTER,1,101,MPI_COMM_WORLD,status,infompi)
  WRITE (*,*) rang, " vient de recevoir du processeur 1 : ",valeur(1)
ELSE
  call MPI_RECV(valeur(1),5000,MPI_CHARACTER,0,100,MPI_COMM_WORLD,status,infompi)
  WRITE (*,*) rang, " vient de recevoir du processeur 0 : ",valeur(1)
  valeur(1)='b'
  WRITE (*,*) rang, " envoie au processeur 0 : ",valeur(1)
  call MPI_SEND(valeur(1),5000,MPI_CHARACTER,0,101,MPI_COMM_WORLD,infompi)
ENDIF


CALL MPI_Finalize(infompi)
```

MPI : communications point à point

```
REAL*8 :: debut, duree
CALL MPI_Init(infompi)
CALL MPI_Comm_size(MPI_COMM_WORLD,nbproc,infompi)
CALL MPI_Comm_rank(MPI_COMM_WORLD,rang,infompi)
debut=MPI_WTIME()
IF (rang .EQ. 0) THEN
  valeur(1)='a'
  WRITE (*,*) rang, " envoie au processeur 1 : ",valeur(1)
  call MPI_SEND(valeur(1),5000,MPI_CHARACTER,1,100,MPI_COMM_WORLD,infompi)
  call MPI_RECV(valeur(1),5000,MPI_CHARACTER,1,101,MPI_COMM_WORLD,status,infompi)
  WRITE (*,*) rang, " vient de recevoir du processeur 1 : ",valeur(1)
ELSE
   call MPI_RECV(valeur(1),5000,MPI_CHARACTER,0,100,MPI_COMM_WORLD,status,infompi)
  WRITE (*,*) rang, " vient de recevoir du processeur 0 : ",valeur(1)
  valeur(1)='b'
  WRITE (*,*) rang, " envoie au processeur 0 : ",valeur(1)
  call MPI_SEND(valeur(1),5000,MPI_CHARACTER,0,101,MPI_COMM_WORLD,infompi)
ENDIF



CALL MPI_Finalize(infompi)
```

MPI : communications point à point

```
REAL*8 :: debut, duree
CALL MPI_Init(infompi)
CALL MPI_Comm_size(MPI_COMM_WORLD,nbproc,infompi)
CALL MPI_Comm_rank(MPI_COMM_WORLD,rang,infompi)
debut=MPI_WTIME()
IF (rang .EQ. 0) THEN
  valeur(1)='a'
  WRITE (*,*) rang, " envoie au processeur 1 : ",valeur(1)
  call MPI_SEND(valeur(1),5000,MPI_CHARACTER,1,100,MPI_COMM_WORLD,infompi)
  call MPI_RECV(valeur(1),5000,MPI_CHARACTER,1,101,MPI_COMM_WORLD,status,infompi)
  WRITE (*,*) rang, " vient de recevoir du processeur 1 : ",valeur(1)
ELSE
   call MPI_RECV(valeur(1),5000,MPI_CHARACTER,0,100,MPI_COMM_WORLD,status,infompi)
  WRITE (*,*) rang, " vient de recevoir du processeur 0 : ",valeur(1)
  valeur(1)='b'
  WRITE (*,*) rang, " envoie au processeur 0 : ",valeur(1)
  call MPI_SEND(valeur(1),5000,MPI_CHARACTER,0,101,MPI_COMM_WORLD,infompi)
ENDIF
duree=MPI_WTIME()

CALL MPI_Finalize(infompi)
```

MPI : communications point à point

```
REAL*8 :: debut, duree
CALL MPI_Init(infompi)
CALL MPI_Comm_size(MPI_COMM_WORLD,nbproc,infompi)
CALL MPI_Comm_rank(MPI_COMM_WORLD,rang,infompi)
debut=MPI_WTIME()
IF (rang .EQ. 0) THEN
  valeur(1)='a'
  WRITE (*,*) rang, " envoie au processeur 1 : ",valeur(1)
  call MPI_SEND(valeur(1),5000,MPI_CHARACTER,1,100,MPI_COMM_WORLD,infompi)
  call MPI_RECV(valeur(1),5000,MPI_CHARACTER,1,101,MPI_COMM_WORLD,status,infompi)
  WRITE (*,*) rang, " vient de recevoir du processeur 1 : ",valeur(1)
ELSE
   call MPI_RECV(valeur(1),5000,MPI_CHARACTER,0,100,MPI_COMM_WORLD,status,infompi)
  WRITE (*,*) rang, " vient de recevoir du processeur 0 : ",valeur(1)
  valeur(1)='b'
  WRITE (*,*) rang, " envoie au processeur 0 : ",valeur(1)
  call MPI_SEND(valeur(1),5000,MPI_CHARACTER,0,101,MPI_COMM_WORLD,infompi)
ENDIF
duree=MPI_WTIME()
WRITE(6,'(a,F8.6)') "temps d'execution : ",duree-debut
CALL MPI_Finalize(infompi)
```

MPI : communications point à point

```
REAL*8 :: debut, duree
CALL MPI_Init(infompi)
CALL MPI_Comm_size(MPI_COMM_WORLD,nbproc,infompi)
CALL MPI_Comm_rank(MPI_COMM_WORLD,rang,infompi)
debut=MPI_WTIME()
IF (rang .EQ. 0) THEN
  valeur(1)='a'
  WRITE (*,*) rang, " envoie au processeur 1 : ",valeur(1)
  call MPI_SEND(valeur(1),5000,MPI_CHARACTER,1,100,MPI_COMM_WORLD,infompi)
  call MPI_RECV(valeur(1),5000,MPI_CHARACTER,1,101,MPI_COMM_WORLD,status,infompi)
  WRITE (*,*) rang, " vient de recevoir du processeur 1 : ",valeur(1)
ELSE
   call MPI_RECV(valeur(1),5000,MPI_CHARACTER,0,100,MPI_COMM_WORLD,status,infompi)
  WRITE (*,*) rang, " vient de recevoir du processeur 0 : ",valeur(1)
  valeur(1)='b'
  WRITE (*,*) rang, " envoie au processeur 0 : ",valeur(1)
  call MPI_SEND(valeur(1),5000,MPI_CHARACTER,0,101,MPI_COMM_WORLD,infompi)
ENDIF
CALL sleep(3) ! Pour la démo
duree=MPI_WTIME()
WRITE(6,'(a,F8.6)') "temps d'execution : ",duree-debut
CALL MPI_Finalize(infompi)
```

MPI : communications point à point

```
mpirun -np 2  ./fcommptapt2_t
0  envoie au processeur 1 : a
1  vient de recevoir du processeur 0 : a
1  envoie au processeur 0 : b
0  vient de recevoir du processeur 1 : b
temps d'execution : 3.000470
temps d'execution : 3.000560
```

MPI : communications point à point

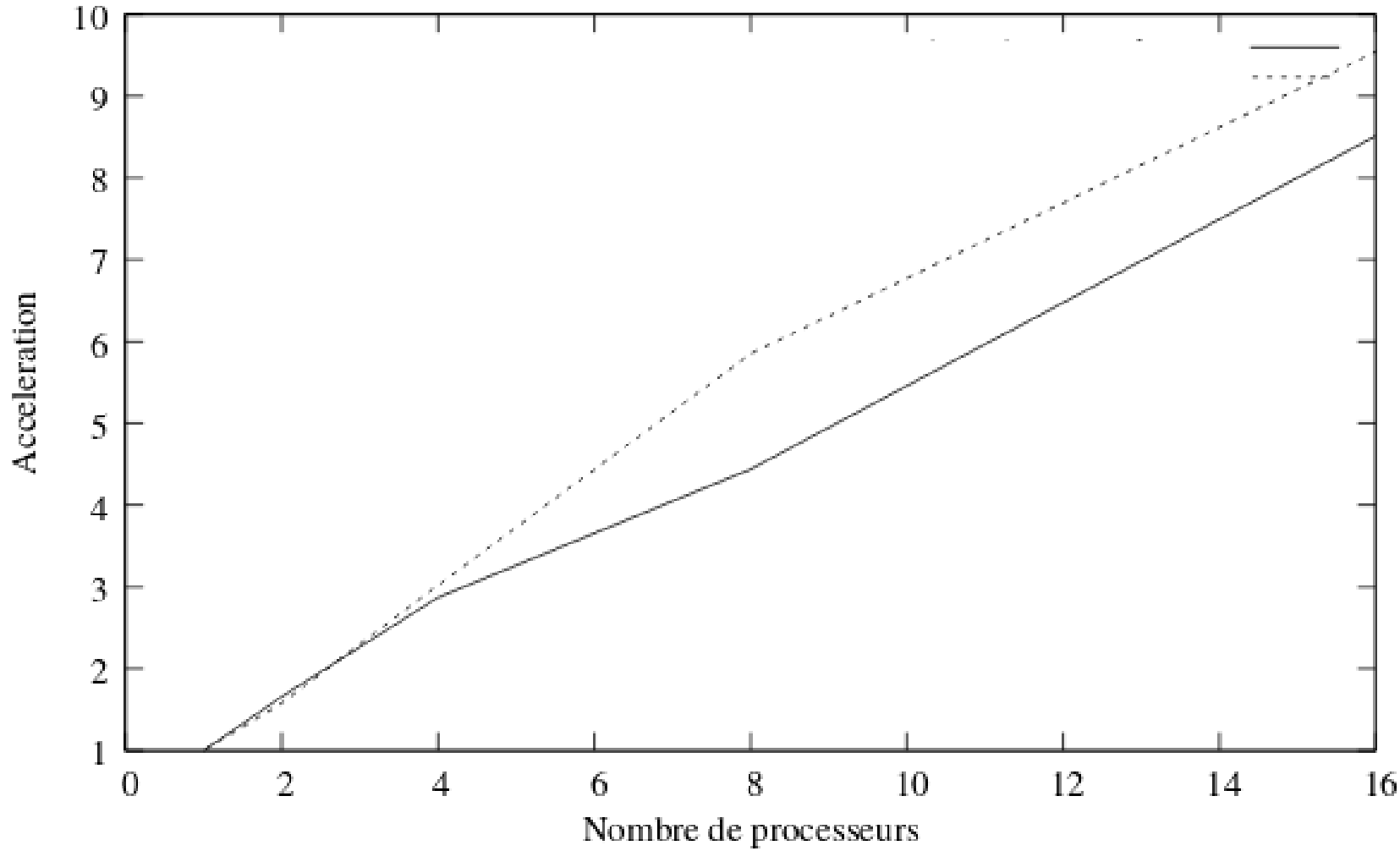
L'accélération (speed-up) et l'efficacité sont deux mesures de la qualité de la parallélisation.

Soit t_1 et t_p les temps d'exécution sur 1 et p processeurs :

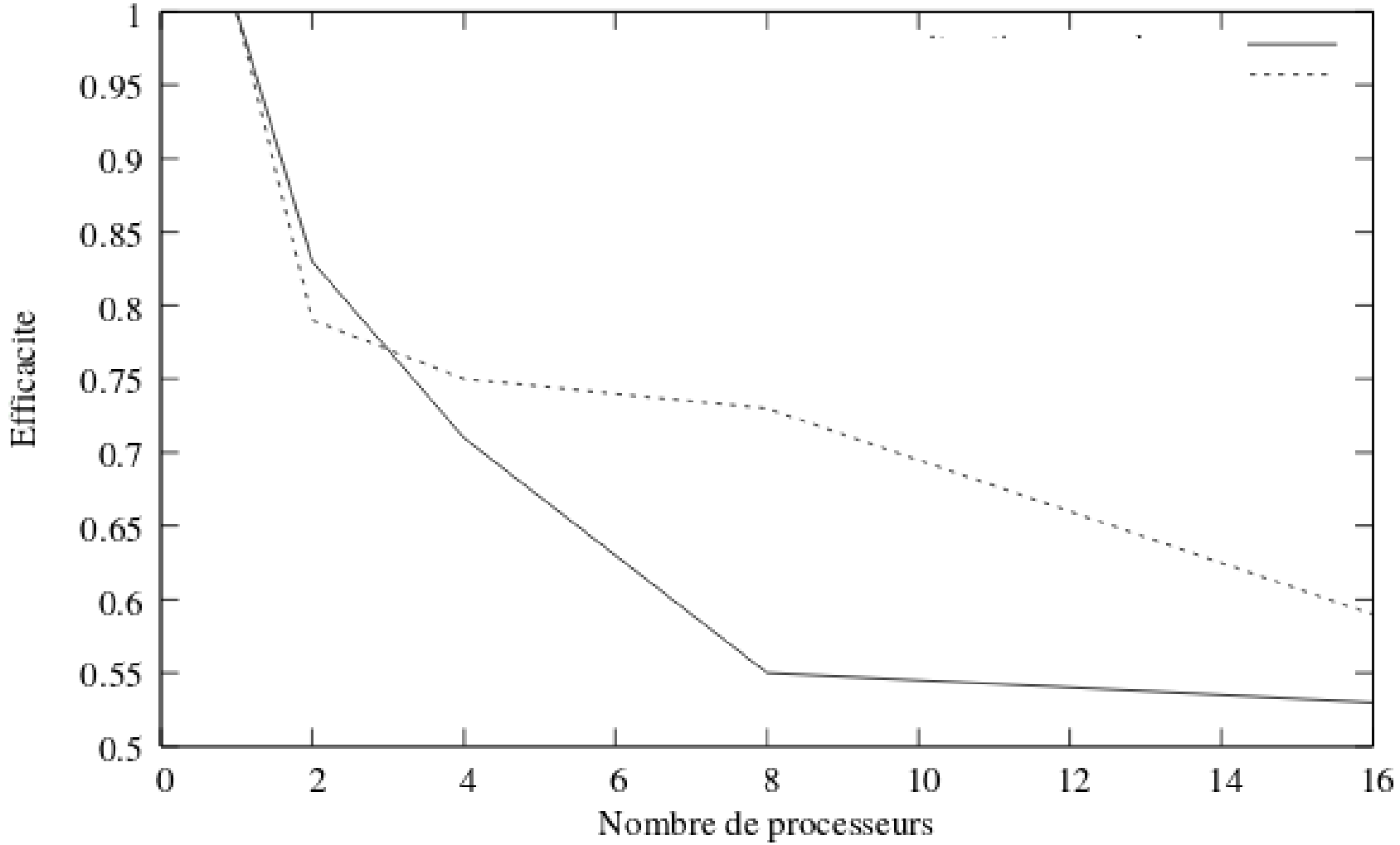
$$accélération = S(p) = t_1 / t_p$$

$$efficacité = E(p) = S(p) / p$$

MPI : communications point à point



MPI : communications point à point



MPI : communications point à point

Une primitive d'envoi ou de réception est dite :

- bloquante si l'espace mémoire servant à la communication peut être réutilisé immédiatement à la fin de son utilisation ;
- non bloquante si elle rend la main immédiatement à la fin de son utilisation. Dans ce cas, pour réutiliser l'espace mémoire servant à la communication, il faudra finaliser la communication.

MPI : communications point à point

Les communications non bloquantes :

- laissent le programme poursuivre son exécution, que la communication soit réellement réalisée ou non;
- présentent un avantage indéniable au niveau du temps d'exécution puisqu'un processeur peut travailler tout en envoyant des données.

MPI : communications point à point

En C, on a les primitives non bloquantes suivantes :

int MPI_Isend(void* valeur, int nbvaleur, MPI_Datatype type, int destinataire, int etiquette, MPI_Comm comm, MPI_Request *req)

− valeur : valeur à envoyer

− nbvaleur : nombre de valeur à envoyer

− type : le type de la valeur

− destinataire : le rang du processus destinataire

− etiquette : une étiquette pour identifier le message

− comm : le communicateur

− req : contient des informations sur la communication (utilisé par les primitives MPI_WAIT et MPI_WAITALL)

MPI : communications point à point

int MPI_Irecv(void* valeur, int nbvaleur, MPI_Datatype type, int emetteur, int etiquette, MPI_Comm comm, MPI_Request *req)

- valeur : valeur à recevoir
- nbvaleur : nombre de valeur à recevoir
- type : le type de la valeur
- emetteur : le rang du processus émetteur
- etiquette : une étiquette pour identifier le message
- comm : le communicateur
- req : contient des informations sur la communication (utilisé par les primitives MPI_WAIT et MPI_WAITALL)

MPI : communications point à point

En Fortran :

CALL MPI_ISEND(type valeur, integer :: nbvaleur, MPI_Datatype type, integer :: destinataire, integer :: etiquette, integer :: comm, integer :: req, integer :: ierr)

− valeur : valeur à recevoir (donner le type de la donnée)

− nbvaleur : nombre de valeur à recevoir

− type : le type de la valeur

− destinataire : le rang du processus destinataire

− etiquette : une étiquette pour identifier le message

− comm : le communicateur

− req : contient des informations sur la communication (utilisé par les primitives MPI_WAIT et MPI_WAITALL)

− ierr : gestion des erreurs

MPI : communications point à point

CALL MPI_IRECV(type valeur, integer :: nbvaleur, MPI_Datatype type, integer :: emetteur, integer :: etiquette, integer :: comm, integer :: req, integer :: ierr)

avec :

− valeur : valeur à recevoir (donner le type de la donnée)

− nbvaleur : nombre de valeur à recevoir

− type : le type de la valeur

− émetteur : le rang du processus émetteur

− etiquette : une étiquette pour identifier le message

− comm : le communicateur

− req : contient des informations sur la communication (utilisé par les primitives MPI_WAIT et MPI_WAITALL)

− ierr : gestion des erreurs

MPI : communications point à point

```
IMPLICIT NONE
INCLUDE "mpif.h"          ! entete mpi fortran (obligatoire)
INTEGER :: nbproc, infompi, rang, req,valeur
CALL MPI_Init(infompi)
CALL MPI_Comm_size(MPI_COMM_WORLD,nbproc,infompi)
CALL MPI_Comm_rank(MPI_COMM_WORLD,rang,infompi)
IF (rang .EQ. 0) THEN
  valeur=2017
  WRITE (*,*) rang, " envoie au processeur 1 : ",valeur
  call MPI_ISEND(valeur,1,MPI_INTEGER,1,100,MPI_COMM_WORLD,req,infompi)
ELSE
  call MPI_IRECV(valeur,1,MPI_INTEGER,0,100,MPI_COMM_WORLD,req,infompi)
  WRITE (*,*) rang, " vient de recevoir du processeur 0 : ",valeur
ENDIF
CALL MPI_Finalize(infompi)
STOP
END PROGRAM
```

MPI : communications point à point

```
IMPLICIT NONE
INCLUDE "mpif.h"          ! entete mpi fortran (obligatoire)
INTEGER :: nbproc, infompi, rang, req,valeur
CALL MPI_Init(infompi)
CALL MPI_Comm_size(MPI_COMM_WORLD,nbproc,infompi)
CALL MPI_Comm_rank(MPI_COMM_WORLD,rang,infompi)
IF (rang .EQ. 0) THEN
  valeur=2017
  WRITE (*,*) rang, " envoie au processeur 1 : ",valeur
  call MPI_ISEND(valeur,1,MPI_INTEGER,1,100,MPI_COMM_WORLD,req,infompi)
ELSE
  call MPI_IRECV(valeur,1,MPI_INTEGER,0,100,MPI_COMM_WORLD,req,infompi)
  WRITE (*,*) rang, " vient de recevoir du processeur 0 : ",valeur
ENDIF
CALL MPI_Finalize(infompi)
STOP
END PROGRAM
```

MPI : communications point à point

mpirun -np 2 ./commptaptNB ou ./fcommptaptNB

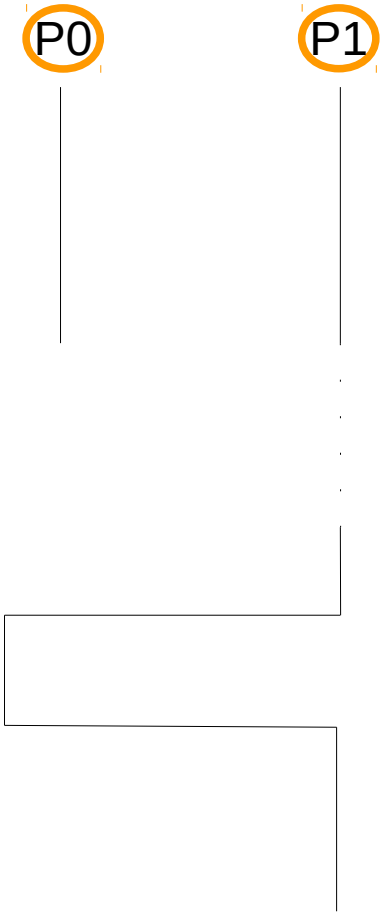
on obtient :

0 envoie au processeur 1 : 2017

1 vient de recevoir du processeur 0 : 32765

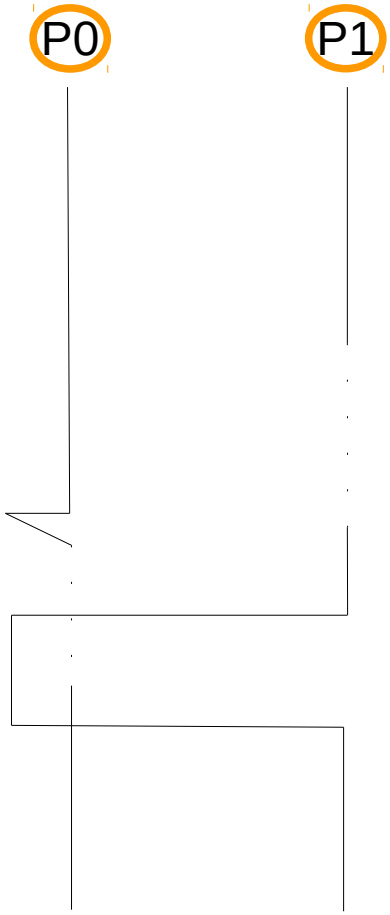
MPI : communications point à point

```
IMPLICIT NONE
INCLUDE "mpif.h"          ! entete mpi fortran (obligatoire)
INTEGER :: nbproc, infompi, rang, req,valeur
CALL MPI_Init(infompi)
CALL MPI_Comm_size(MPI_COMM_WORLD,nbproc,infompi)
CALL MPI_Comm_rank(MPI_COMM_WORLD,rang,infompi)
IF (rang .EQ. 0) THEN
  valeur=2017
  WRITE (*,*) rang, " envoie au processeur 1 : ",valeur
  call MPI_ISEND(valeur,1,MPI_INTEGER,1,100,MPI_COMM_WORLD,req,infompi)
ELSE
  call MPI_IRECV(valeur,1,MPI_INTEGER,0,100,MPI_COMM_WORLD,req,infompi)
  WRITE (*,*) rang, " vient de recevoir du processeur 0 : ",valeur
ENDIF
CALL MPI_Finalize(infompi)
STOP
END PROGRAM
```



MPI : communications point à point

```
IMPLICIT NONE
INCLUDE "mpif.h"          ! entete mpi fortran (obligatoire)
INTEGER :: nbproc, infompi, rang, req,valeur
CALL MPI_Init(infompi)
CALL MPI_Comm_size(MPI_COMM_WORLD,nbproc,infompi)
CALL MPI_Comm_rank(MPI_COMM_WORLD,rang,infompi)
IF (rang .EQ. 0) THEN
  valeur=2017
  WRITE (*,*) rang, " envoie au processeur 1 : ",valeur
  call MPI_ISEND(valeur,1,MPI_INTEGER,1,100,MPI_COMM_WORLD,req,infompi)
ELSE
  call MPI_IRECV(valeur,1,MPI_INTEGER,0,100,MPI_COMM_WORLD,req,infompi)
  WRITE (*,*) rang, " vient de recevoir du processeur 0 : ",valeur
ENDIF
CALL MPI_Finalize(infompi)
STOP
END PROGRAM
```



MPI : communications point à point

```
...
char valeur[5000];
...
if (rang == 0) {
   valeur[0]='a' ;
   printf("%d envoie au processeur 1 : %c \n",rang,valeur[0]);
   MPI_Isend(valeur,5000,MPI_CHAR,1,100,MPI_COMM_WORLD,&req);
   MPI_Irecv(valeur,5000,MPI_CHAR,1,100,MPI_COMM_WORLD,&req);
   printf("%d vient de recevoir du processeur 1 : %c \n",rang,valeur[0]);
}
else if (rang == 1) {
   MPI_Irecv(valeur,5000,MPI_CHAR,0,100,MPI_COMM_WORLD,&req);
   printf("%d vient de recevoir du processeur 0 : %c \n",rang,valeur[0]);
   valeur[0]='b';
   printf("%d envoie au processeur 0 : %c \n",rang,valeur[0]);
   MPI_Isend(valeur,5000,MPI_CHAR,0,100,MPI_COMM_WORLD,&req);
}
...
```

MPI : communications point à point

mpirun -np 2 ./commptaptNB1 ou ./fcommptaptNB1

on obtient :

1 vient de recevoir du processeur 0 : ?

1 envoie au processeur 0 : b

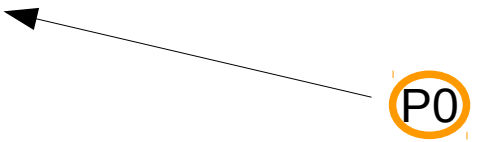
0 envoie au processeur 1 : a

0 vient de recevoir du processeur 1 : a

MPI : communications point à point

```
...
char valeur[5000];
...
if (rang == 0) {
   valeur[0]='a' ;
   printf("%d envoie au processeur 1 : %c \n",rang,valeur[0]);
   MPI_Isend(valeur,5000,MPI_CHAR,1,100,MPI_COMM_WORLD,&req);
   MPI_Irecv(valeur,5000,MPI_CHAR,1,100,MPI_COMM_WORLD,&req);
   printf("%d vient de recevoir du processeur 1 : %c \n",rang,valeur[0]);
}
else if (rang == 1) {
   MPI_Irecv(valeur,5000,MPI_CHAR,0,100,MPI_COMM_WORLD,&req);
   printf("%d vient de recevoir du processeur 0 : %c \n",rang,valeur[0]);
   valeur[0]='b';
   printf("%d envoie au processeur 0 : %c \n",rang,valeur[0]);
   MPI_Isend(valeur,5000,MPI_CHAR,0,100,MPI_COMM_WORLD,&req);
}
...
```

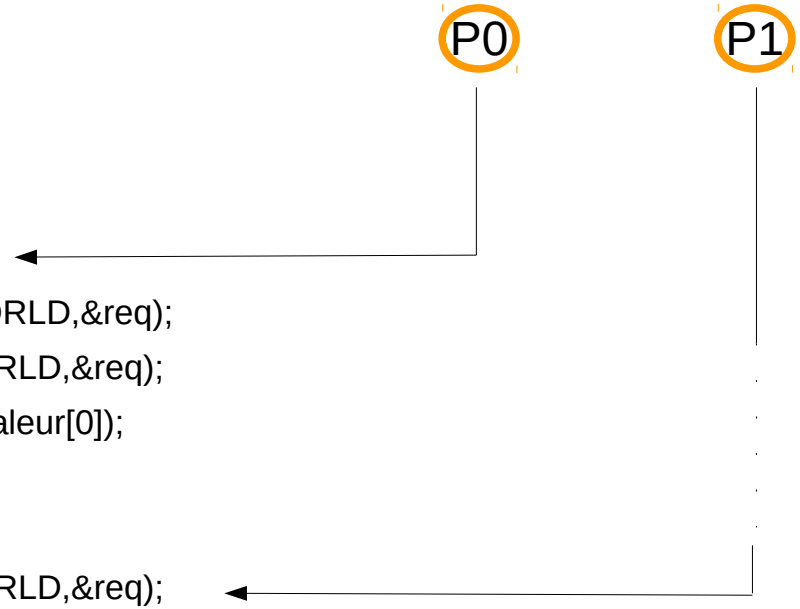
Non bloquant – émet les données et continue



Non bloquant – reçoit les données et continue

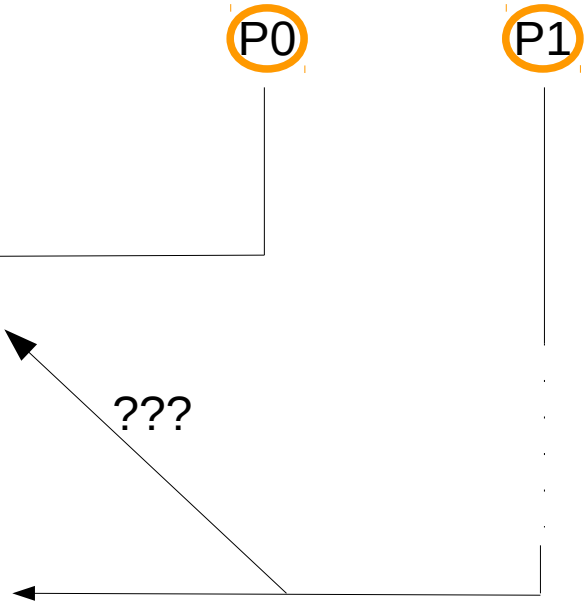
MPI : communications point à point

```
...
char valeur[5000];
...
if (rang == 0) {
   valeur[0]='a' ;
   printf("%d envoie au processeur 1 : %c \n",rang,valeur[0]);
   MPI_Isend(valeur,5000,MPI_CHAR,1,100,MPI_COMM_WORLD,&req);
   MPI_Irecv(valeur,5000,MPI_CHAR,1,100,MPI_COMM_WORLD,&req);
   printf("%d vient de recevoir du processeur 1 : %c \n",rang,valeur[0]);
}
else if (rang == 1) {
   MPI_Irecv(valeur,5000,MPI_CHAR,0,100,MPI_COMM_WORLD,&req);
   printf("%d vient de recevoir du processeur 0 : %c \n",rang,valeur[0]);
   valeur[0]='b';
   printf("%d envoie au processeur 0 : %c \n",rang,valeur[0]);
   MPI_Isend(valeur,5000,MPI_CHAR,0,100,MPI_COMM_WORLD,&req);
}
...
```



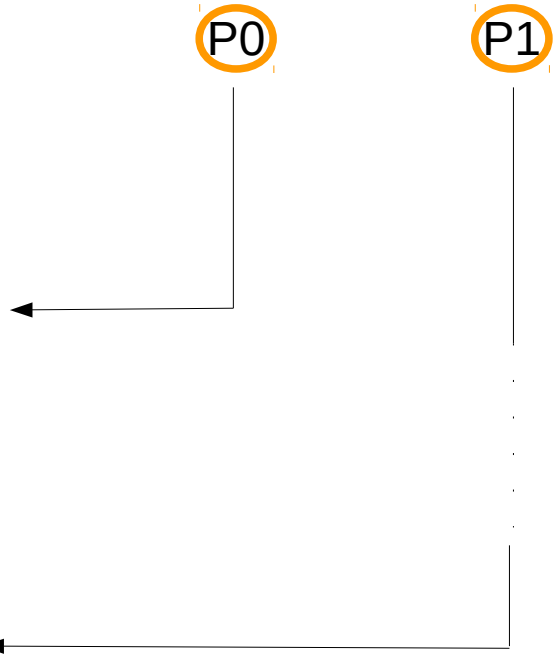
MPI : communications point à point

```
...
char valeur[5000];
...
if (rang == 0) {
   valeur[0]='a' ;
   printf("%d envoie au processeur 1 : %c \n",rang,valeur[0]);
   MPI_Isend(valeur,5000,MPI_CHAR,1,100,MPI_COMM_WORLD,&req);
   MPI_Irecv(valeur,5000,MPI_CHAR,1,100,MPI_COMM_WORLD,&req);
   printf("%d vient de recevoir du processeur 1 : %c \n",rang,valeur[0]);
}
else if (rang == 1) {
   MPI_Irecv(valeur,5000,MPI_CHAR,0,100,MPI_COMM_WORLD,&req);
   printf("%d vient de recevoir du processeur 0 : %c \n",rang,valeur[0]);
   valeur[0]='b';
   printf("%d envoie au processeur 0 : %c \n",rang,valeur[0]);
   MPI_Isend(valeur,5000,MPI_CHAR,0,100,MPI_COMM_WORLD,&req);
}
...
```



MPI : communications point à point

```
...
char valeur[5000];
...
if (rang == 0) {
   valeur[0]='a' ;
   printf("%d envoie au processeur 1 : %c \n",rang,valeur[0]);
   MPI_Isend(valeur,5000,MPI_CHAR,1,100,MPI_COMM_WORLD,&req);
   MPI_Irecv(valeur,5000,MPI_CHAR,1,100,MPI_COMM_WORLD,&req);
   printf("%d vient de recevoir du processeur 1 : %c \n",rang,valeur[0]);
}
else if (rang == 1) {
   MPI_Irecv(valeur,5000,MPI_CHAR,0,100,MPI_COMM_WORLD,&req);
   printf("%d vient de recevoir du processeur 0 : %c \n",rang,valeur[0]);
   valeur[0]='b';
   printf("%d envoie au processeur 0 : %c \n",rang,valeur[0]);
   MPI_Isend(valeur,5000,MPI_CHAR,0,100,MPI_COMM_WORLD,&req);
}
...
```



MPI : communications point à point

On peut attendre la fin de l'opération ou tester son état :

en C

int MPI_Wait (MPI_Request *req, MPI_Status *status)

int MPI_Test (MPI_Request *req, int *flag, MPI_Status *status)

flag est vrai si la communication testée est terminée

en Fortran

MPI_WAIT (handle, status, ierror)

MPI_TEST (handle, flag, status, ierror)

MPI : communications point à point

```
if (rang == 0)  {
   valeur[0]='a';
   printf("%d envoie au processeur 1 : %c \n",rang,valeur[0]);
   MPI_Isend(valeur,5000,MPI_CHAR,1,100,MPI_COMM_WORLD,&req);
   MPI_Wait (&req,&status);
   MPI_Irecv(valeur,5000,MPI_CHAR,1,100,MPI_COMM_WORLD,&req);
   MPI_Wait (&req,&status);
   printf("%d vient de recevoir du processeur 1 : %c \n",rang,valeur[0]);
}
else if (rang == 1) {
   MPI_Irecv(valeur,5000,MPI_CHAR,0,100,MPI_COMM_WORLD,&req);
   MPI_Wait (&req,&status);
   printf("%d vient de recevoir du processeur 0 : %c \n",rang,valeur[0]);
   valeur[0]='b';
   printf("%d envoie au processeur 0 : %c \n",rang,valeur[0]);
   MPI_Isend(valeur,5000,MPI_CHAR,0,100,MPI_COMM_WORLD,&req);
   MPI_Wait (&req,&status);
}
```

MPI : communications point à point

mpirun -np 2 ./commptaptNB2 ou ./fcommptaptNB2

on obtient :

0 envoie au processeur 1 : a

1 vient de recevoir du processeur 0 : a

1 envoie au processeur 0 : b

0 vient de recevoir du processeur 1 : b

MPI : communications point à point

```
if (rang == 0)  {
   valeur[0]='a';
   printf("%d envoie au processeur 1 : %c \n",rang,valeur[0]);
   MPI_Isend(valeur,5000,MPI_CHAR,1,100,MPI_COMM_WORLD,&req);
   flag=0; while (!flag) MPI_Test(&req, &flag, &status);
   MPI_Irecv(valeur,5000,MPI_CHAR,1,100,MPI_COMM_WORLD,&req);
   flag=0; while (!flag) MPI_Test(&req, &flag, &status);
   printf("%d vient de recevoir du processeur 1 : %c \n",rang,valeur[0]);
}
else if (rang == 1) {
   MPI_Irecv(valeur,5000,MPI_CHAR,0,100,MPI_COMM_WORLD,&req);
   flag=0; while (!flag) MPI_Test(&req, &flag, &status);
   printf("%d vient de recevoir du processeur 0 : %c \n",rang,valeur[0]);
   valeur[0]='b';
   printf("%d envoie au processeur 0 : %c \n",rang,valeur[0]);
   MPI_Isend(valeur,5000,MPI_CHAR,0,100,MPI_COMM_WORLD,&req);
   flag=0; while (!flag) MPI_Test(&req, &flag, &status);
}
```

MPI : communications point à point

mpirun -np 2 ./commptaptNB3 ou ./fcommptaptNB3

on obtient :

0 envoie au processeur 1 : a

1 vient de recevoir du processeur 0 : a

1 envoie au processeur 0 : b

0 vient de recevoir du processeur 1 : b

MPI : Exercice Dirigé

Produit Matrice Vecteur

Une matrice n × m est un tableau de nombres à n lignes et m colonnes. On dit que n et m sont les dimensions de la matrice.

Une matrice est une lettre par exemple A.

On note A_{ij} l'élément situé à l'intersection de la ligne i et de la colonne j (la ligne est toujours nommée en premier).

MPI : Exercice Dirigé

On note $[A_{ij}]$ la matrice d'élément général A_{ij}.

On a donc : $A = [A_{ij}]$

$$A = \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1m} \\ A_{21} & A_{22} & \dots & A_{2m} \\ \dots & \dots & \dots & \dots \\ A_{n1} & A_{n2} & \dots & A_{nm} \end{bmatrix}$$

MPI : Exercice Dirigé

Produit Matrice Vecteur

Si m = 1, la matrice est appelée vecteur (plus précisément vecteur-colonne) :

$$u = \begin{bmatrix} u_1 \\ u_2 \\ \dots \\ u_n \end{bmatrix}$$

MPI : Exercice Dirigé

Produit Matrice Vecteur

Si n = m, la matrice est appelée matrice carrée.

Matrice unité

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Parfois notée $\mathbf{I}_n$
n est la dimension de la matrice
(soit $\mathbf{I}_4$ dans cet exemple)

Matrice diagonale

$$\mathbf{D} = \begin{bmatrix} D_{11} & 0 & 0 & 0 \\ 0 & D_{22} & 0 & 0 \\ 0 & 0 & D_{33} & 0 \\ 0 & 0 & 0 & D_{44} \end{bmatrix}$$

notée $\mathrm{diag}(D_{ii})$

MPI : Exercice Dirigé

Produit Matrice Vecteur

Si n = m, la matrice est appelée matrice carrée.

Matrice triangulaire supérieure (*Upper triangular matrix*, **U**)

$$\mathbf{U} = \begin{bmatrix} U_{11} & U_{12} & U_{13} & U_{14} \\ 0 & U_{22} & U_{23} & U_{24} \\ 0 & 0 & U_{33} & U_{34} \\ 0 & 0 & 0 & U_{44} \end{bmatrix}$$

Matrice triangulaire inférieure (*Lower triangular matrix*, **L**)

$$\mathbf{L} = \begin{bmatrix} L_{11} & 0 & 0 & 0 \\ L_{21} & L_{22} & 0 & 0 \\ L_{31} & L_{32} & L_{33} & 0 \\ L_{41} & L_{42} & L_{43} & L_{44} \end{bmatrix}$$

MPI : Exercice Dirigé

Produit Matrice Vecteur

on cherche à faire A x u = b

$$b = \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1m} \\ A_{21} & A_{22} & \dots & A_{2m} \\ \dots & \dots & \dots & \dots \\ A_{n1} & A_{n2} & \dots & A_{nm} \end{bmatrix} \times \begin{bmatrix} u_1 \\ u_2 \\ \dots \\ u_n \end{bmatrix}$$

MPI : Exercice Dirigé

Produit Matrice Vecteur

Alors B est une matrice colonne avec m lignes

$$b = \begin{bmatrix} A_{11}u_1 & + & A_{12}u_2 & + & \dots & + & A_{1m}u_n \\ A_{21}u_1 & + & A_{22}u_2 & + & \dots & + & A_{2m}u_n \\ \dots & + & \dots & + & \dots & + & \dots \\ A_{n1}u_1 & + & A_{n2}u_2 & + & \dots & + & A_{nm}u_n \end{bmatrix}$$

MPI : Exercice Dirigé

Produit Matrice Vecteur (version séquentielle)

```
for ( i = 1; i <= m ; i ++) { b [ i ] = 0.0; }
for ( j = 1; j <= n ; j ++) {
    for ( i = 1; i <= m ; i ++) {
        b [ i ] = b[ i ] + A [ i ][ j ] * u [ j ];
    }
}
```

MPI : Exercice Dirigé

Produit Matrice Vecteur

Alors B est une matrice colonne avec m lignes

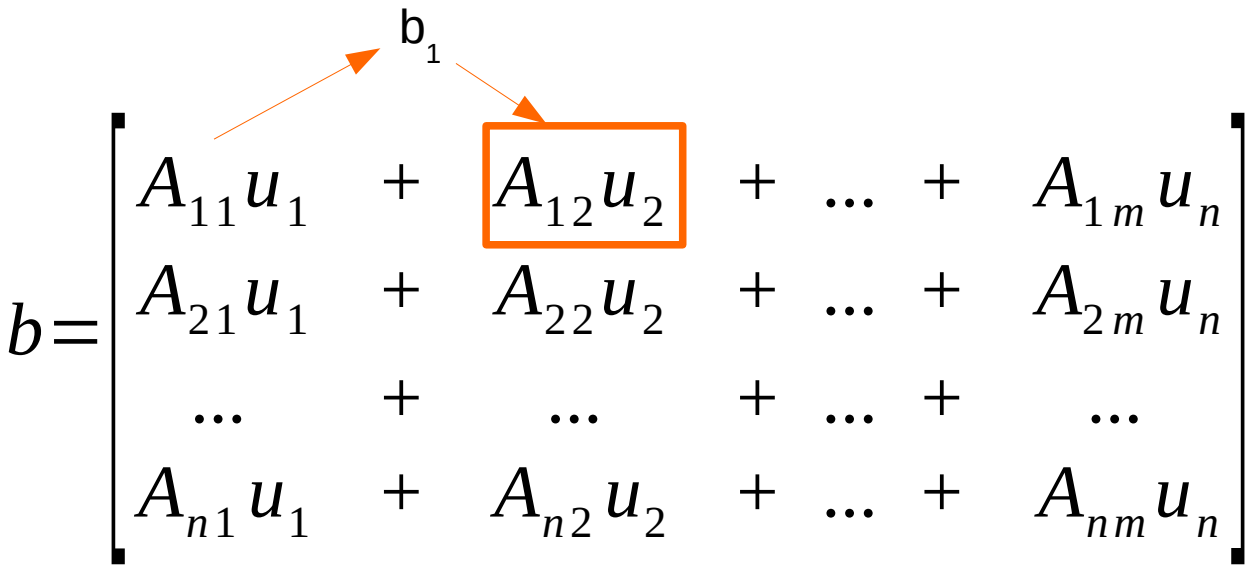
$$b = \begin{bmatrix} \mathbf{A_{11}u_1} & + & A_{12}u_2 & + & \dots & + & A_{1m}u_n \\ A_{21}u_1 & + & A_{22}u_2 & + & \dots & + & A_{2m}u_n \\ \dots & + & \dots & + & \dots & + & \dots \\ A_{n1}u_1 & + & A_{n2}u_2 & + & \dots & + & A_{nm}u_n \end{bmatrix}$$

b_1 →

MPI : Exercice Dirigé

Produit Matrice Vecteur

Alors B est une matrice colonne avec m lignes



MPI : Exercice Dirigé

Produit Matrice Vecteur

Alors B est une matrice colonne avec m lignes

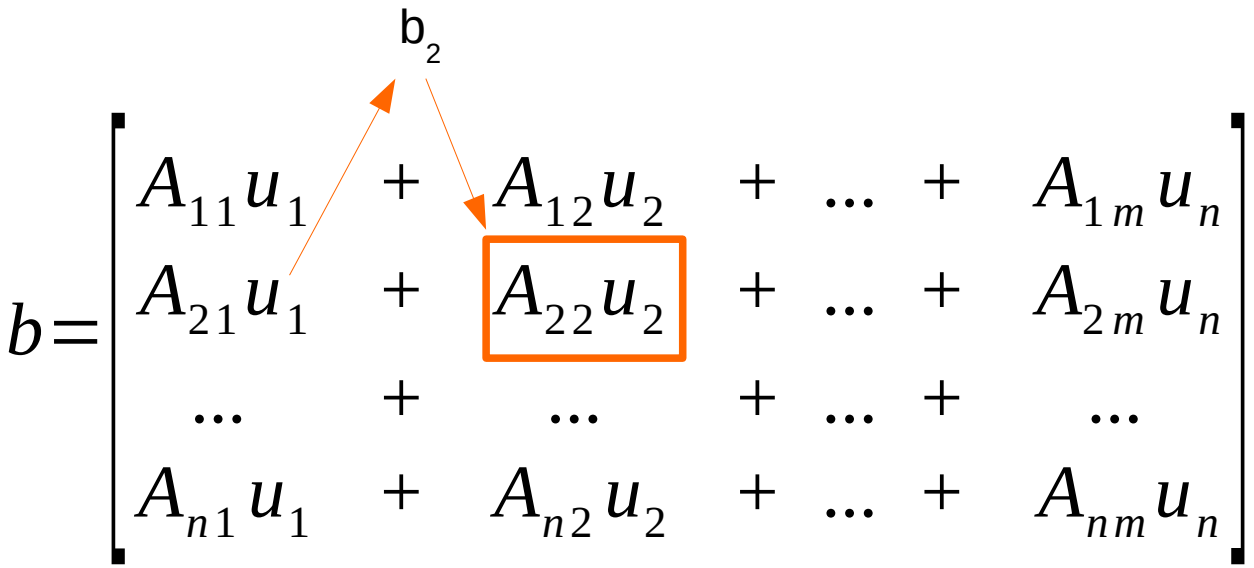
$$b = \begin{bmatrix} A_{11}u_1 & + & A_{12}u_2 & + & \dots & + & A_{1m}u_n \\ A_{21}u_1 & + & A_{22}u_2 & + & \dots & + & A_{2m}u_n \\ \dots & + & \dots & + & \dots & + & \dots \\ A_{n1}u_1 & + & A_{n2}u_2 & + & \dots & + & A_{nm}u_n \end{bmatrix}$$

b_2

MPI : Exercice Dirigé

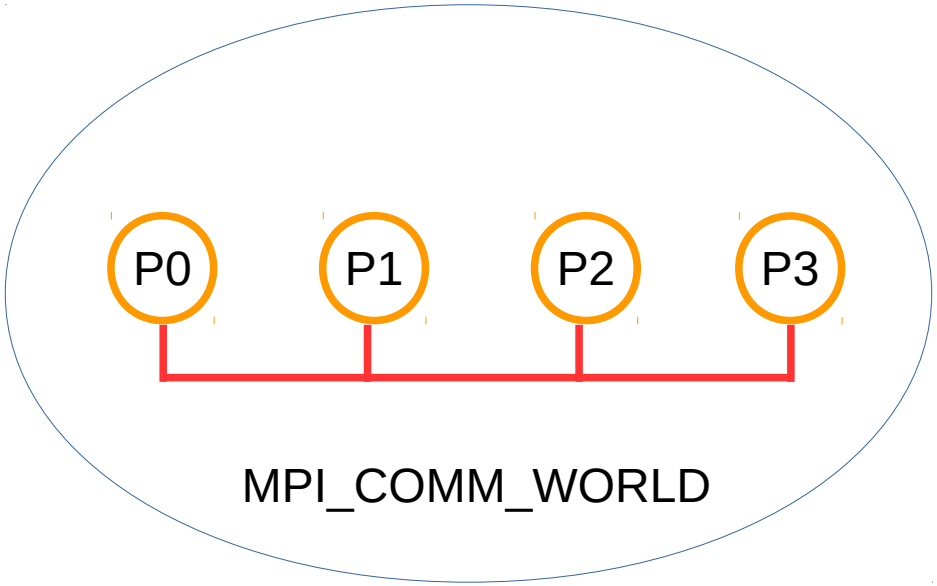
Produit Matrice Vecteur

Alors B est une matrice colonne avec m lignes



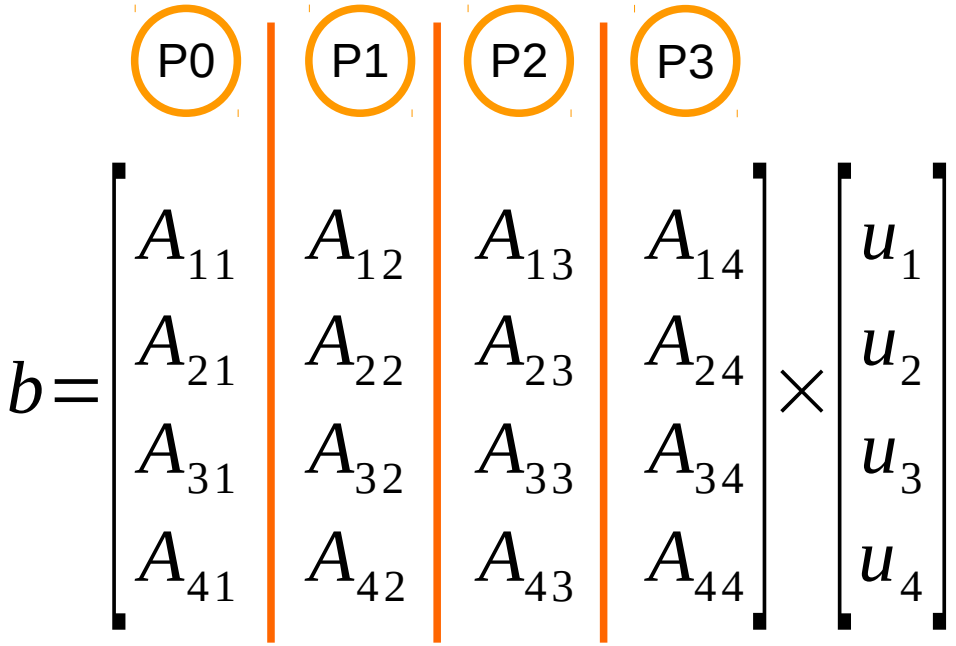
MPI : Exercice Dirigé

Comment partager les calculs sur 4 processeurs ?



MPI : Exercice Dirigé

Découpage



MPI : Exercice Dirigé

Découpage

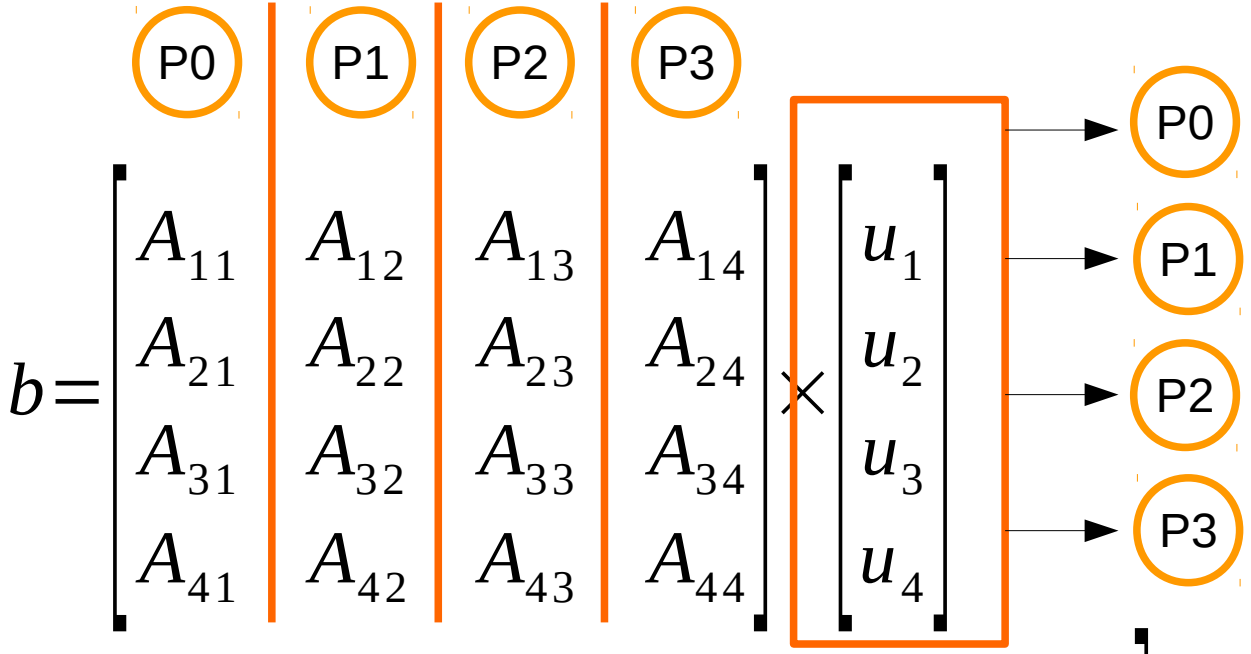
P0 P1 P2 P3

$$b = \left[\begin{array}{c|c|c|c} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{array}\right] \times \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}$$

$$b = \begin{bmatrix} A_{11}u_1 + A_{12}u_2 + A_{13}u_3 + A_{14}u_4 \\ A_{21}u_1 + A_{22}u_2 + A_{23}u_3 + A_{24}u_4 \\ A_{31}u_1 + A_{32}u_2 + A_{33}u_3 + A_{34}u_4 \\ A_{41}u_1 + A_{42}u_2 + A_{43}u_3 + A_{44}u_4 \end{bmatrix}$$

MPI : Exercice Dirigé

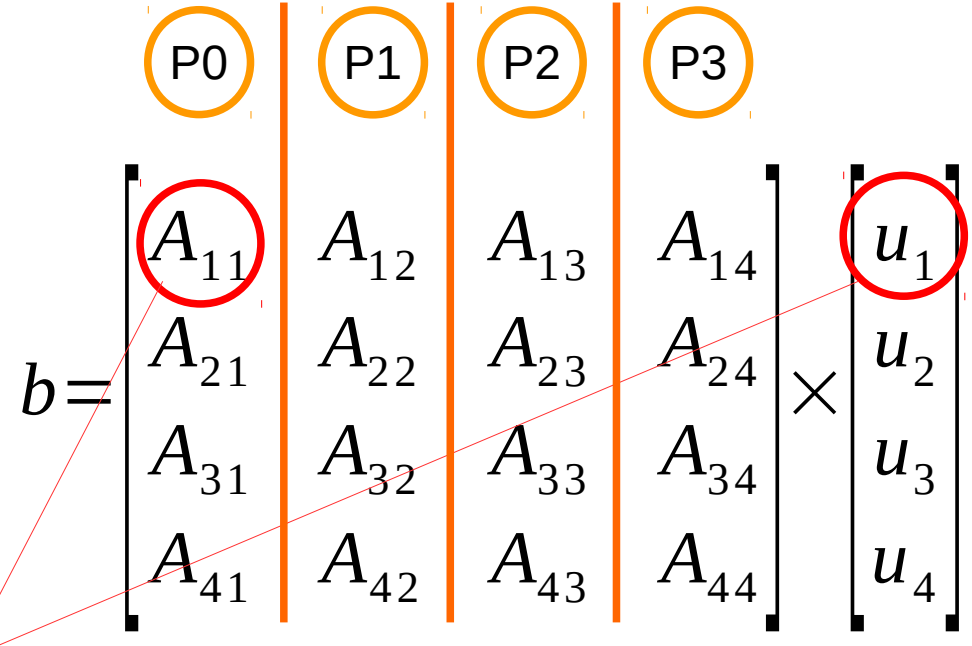
Découpage



$$b = \begin{bmatrix} A_{11}u_1 + A_{12}u_2 + A_{13}u_3 + A_{14}u_4 \\ A_{21}u_1 + A_{22}u_2 + A_{23}u_3 + A_{24}u_4 \\ A_{31}u_1 + A_{32}u_2 + A_{33}u_3 + A_{34}u_4 \\ A_{41}u_1 + A_{42}u_2 + A_{43}u_3 + A_{44}u_4 \end{bmatrix}$$

MPI : Exercice Dirigé

Découpage

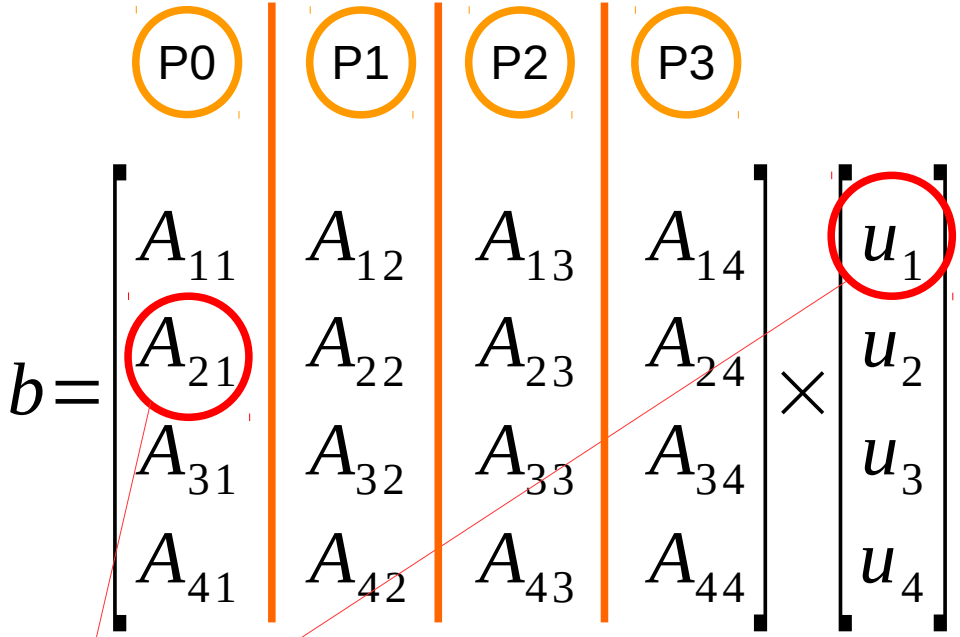


$$b = \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix} \times \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}$$

$$b = \begin{bmatrix} A_{11}u_1 + A_{12}u_2 + A_{13}u_3 + A_{14}u_4 \\ A_{21}u_1 + A_{22}u_2 + A_{23}u_3 + A_{24}u_4 \\ A_{31}u_1 + A_{32}u_2 + A_{33}u_3 + A_{34}u_4 \\ A_{41}u_1 + A_{42}u_2 + A_{43}u_3 + A_{44}u_4 \end{bmatrix}$$

MPI : Exercice Dirigé

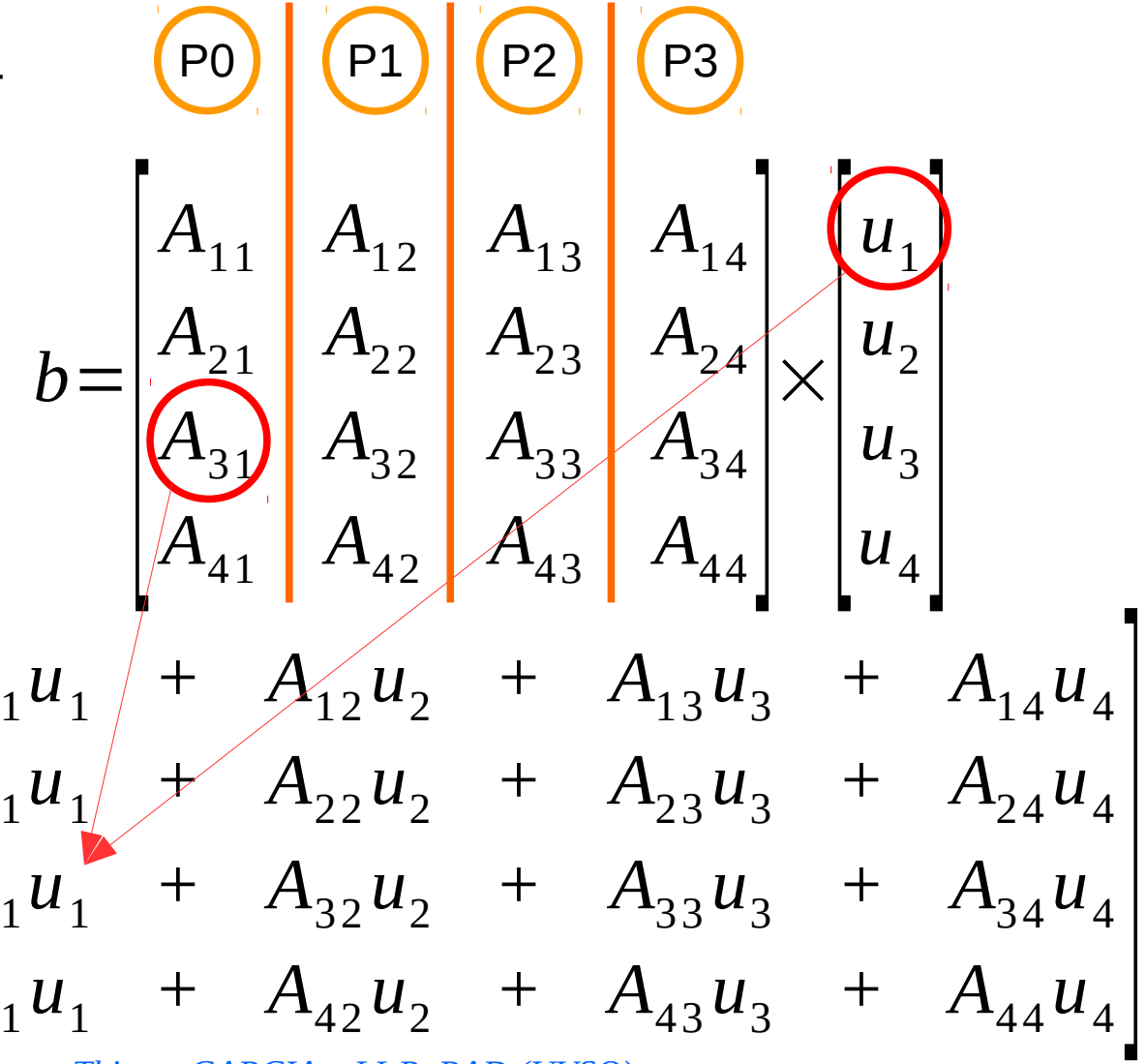
Découpage



$$b = \begin{bmatrix} A_{11}u_1 + A_{12}u_2 + A_{13}u_3 + A_{14}u_4 \\ A_{21}u_1 + A_{22}u_2 + A_{23}u_3 + A_{24}u_4 \\ A_{31}u_1 + A_{32}u_2 + A_{33}u_3 + A_{34}u_4 \\ A_{41}u_1 + A_{42}u_2 + A_{43}u_3 + A_{44}u_4 \end{bmatrix}$$

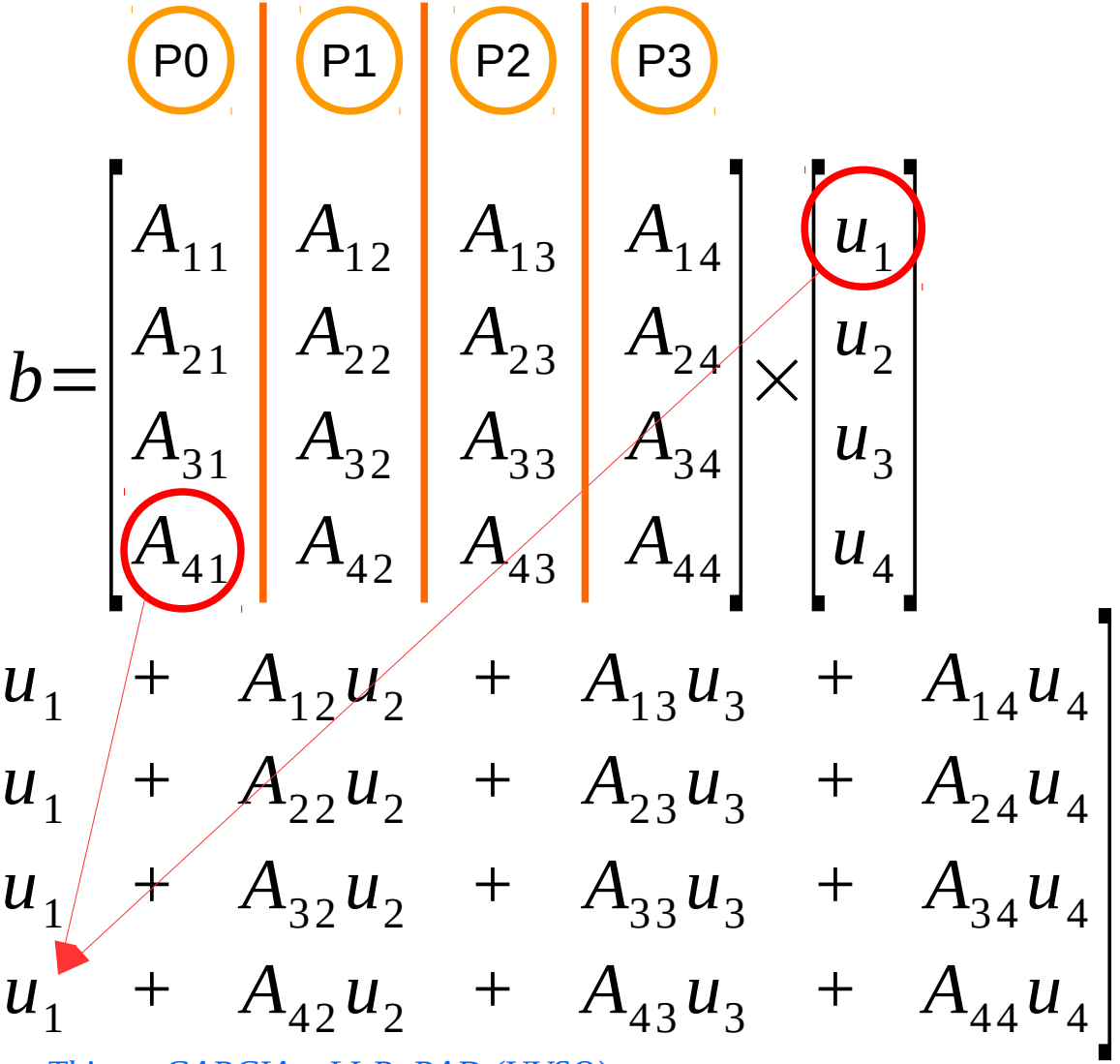
MPI : Exercice Dirigé

Découpage



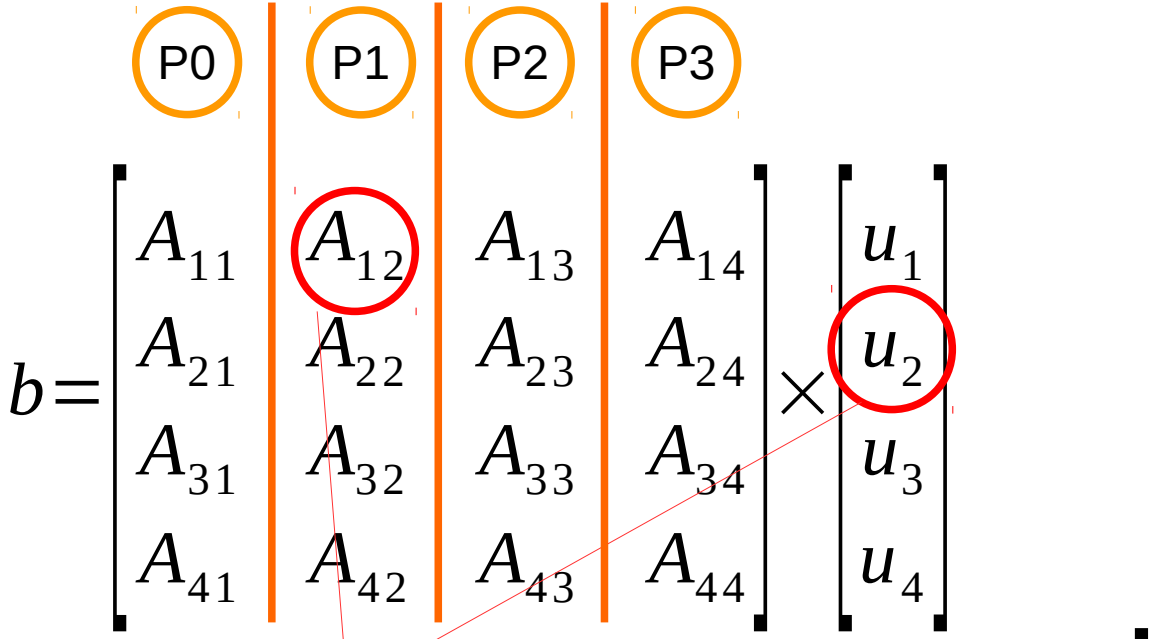
MPI : Exercice Dirigé

Découpage



MPI : Exercice Dirigé

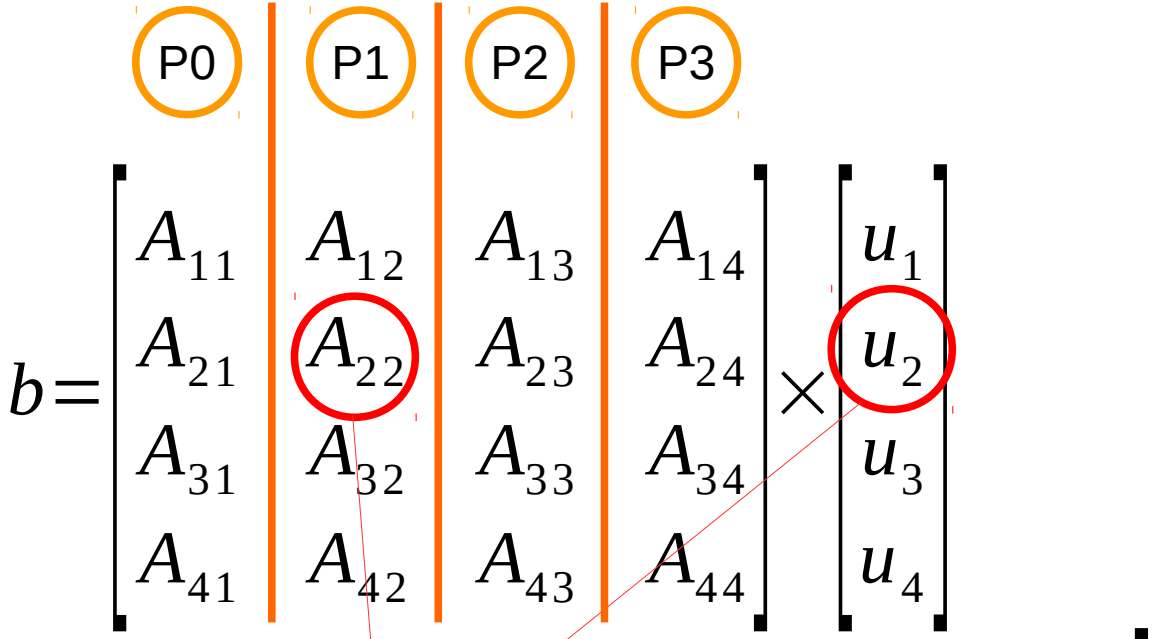
Découpage



$$b = \begin{bmatrix} A_{11}u_1 + A_{12}u_2 + A_{13}u_3 + A_{14}u_4 \\ A_{21}u_1 + A_{22}u_2 + A_{23}u_3 + A_{24}u_4 \\ A_{31}u_1 + A_{32}u_2 + A_{33}u_3 + A_{34}u_4 \\ A_{41}u_1 + A_{42}u_2 + A_{43}u_3 + A_{44}u_4 \end{bmatrix}$$

MPI : Exercice Dirigé

Découpage

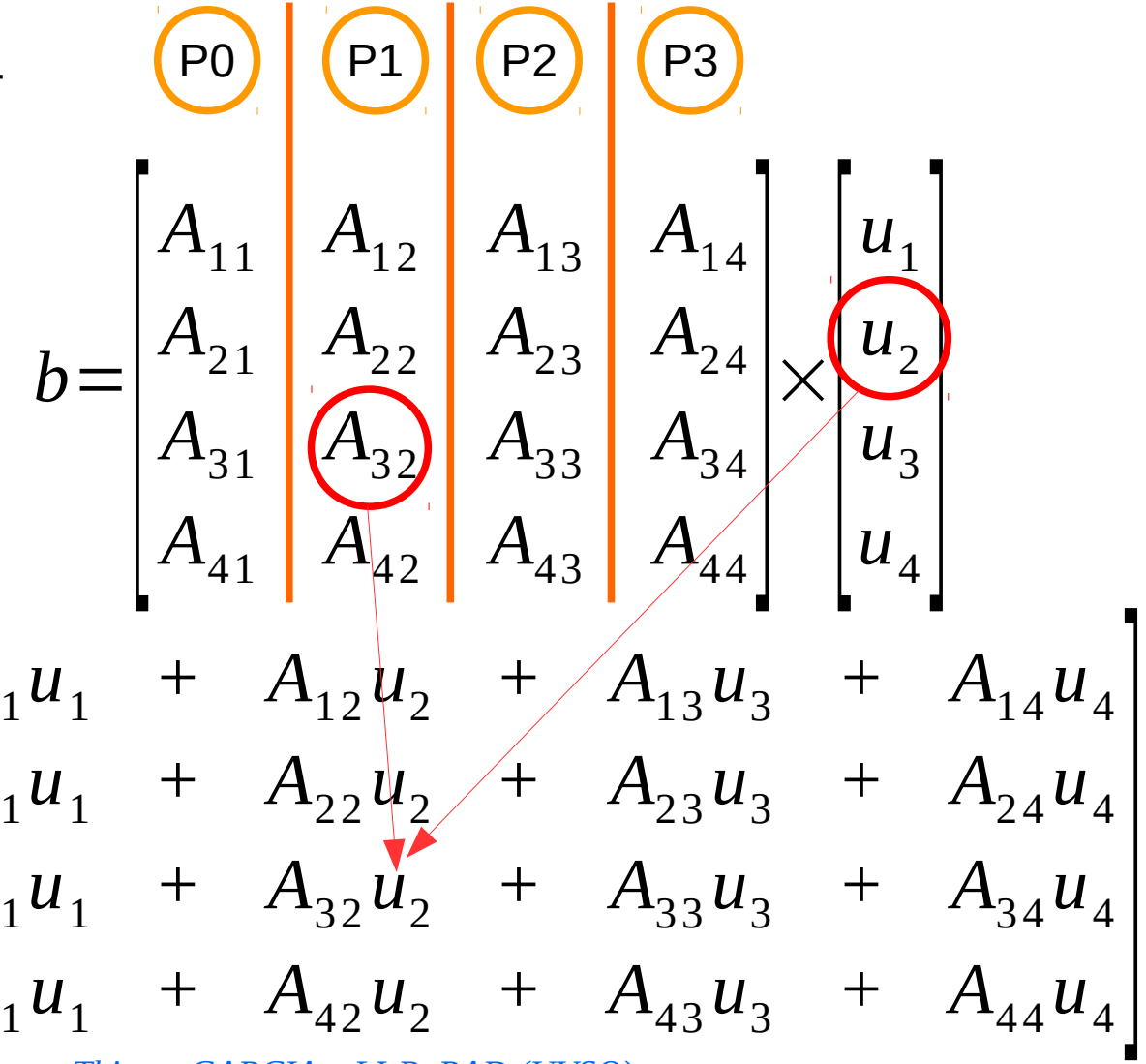


$$b = \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix} \times \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}$$

$$b = \begin{bmatrix} A_{11}u_1 + A_{12}u_2 + A_{13}u_3 + A_{14}u_4 \\ A_{21}u_1 + A_{22}u_2 + A_{23}u_3 + A_{24}u_4 \\ A_{31}u_1 + A_{32}u_2 + A_{33}u_3 + A_{34}u_4 \\ A_{41}u_1 + A_{42}u_2 + A_{43}u_3 + A_{44}u_4 \end{bmatrix}$$

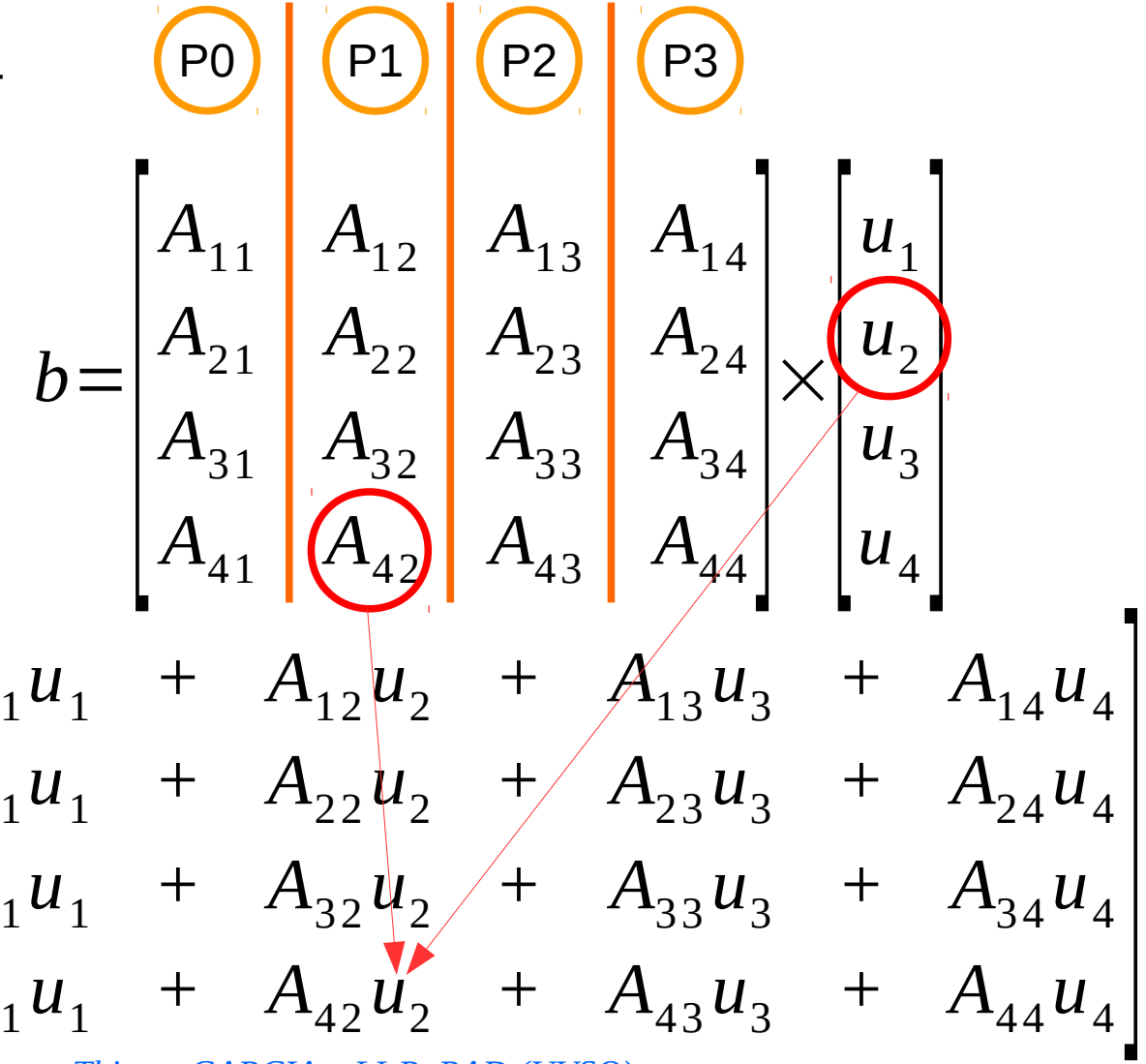
MPI : Exercice Dirigé

Découpage



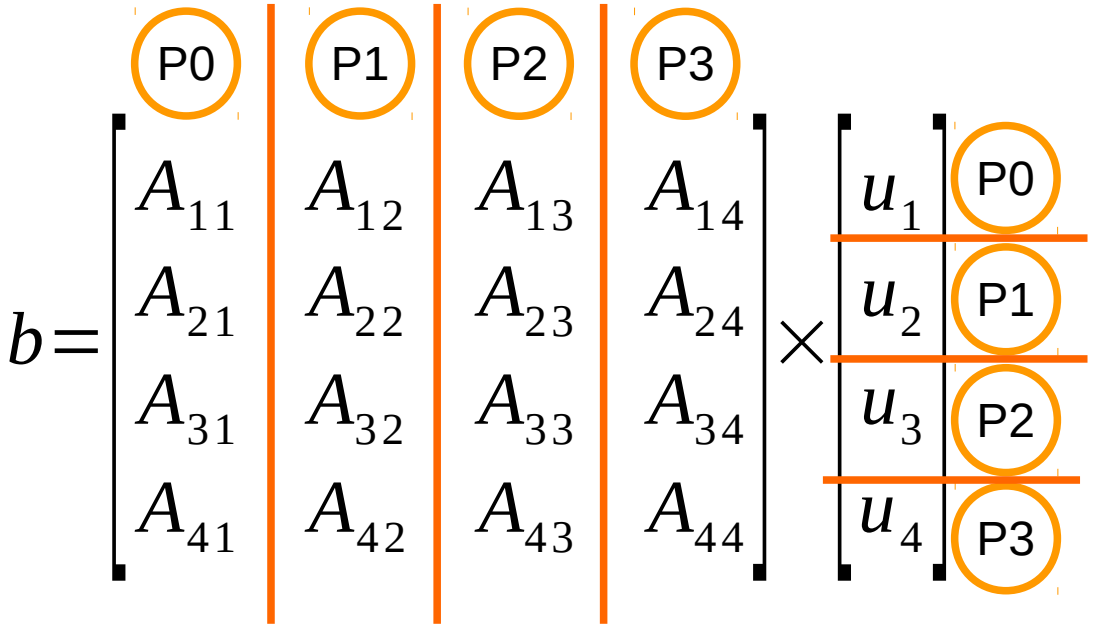
MPI : Exercice Dirigé

Découpage



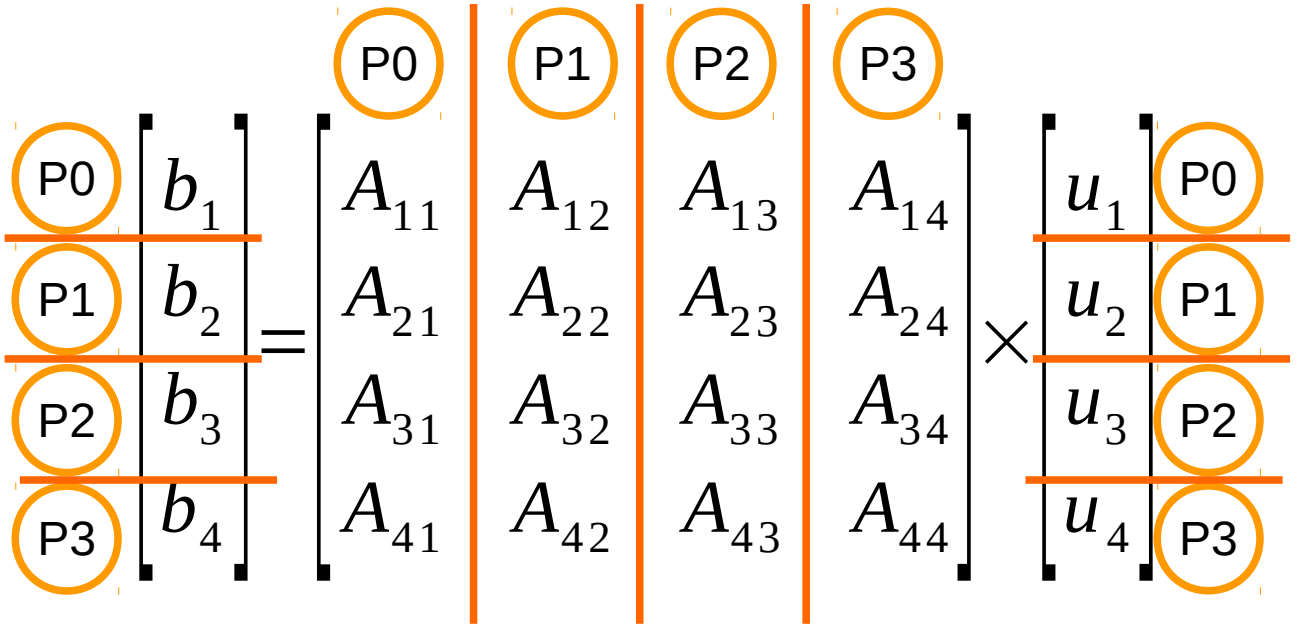
MPI : Exercice Dirigé

Découpage



MPI : Exercice Dirigé

Découpage

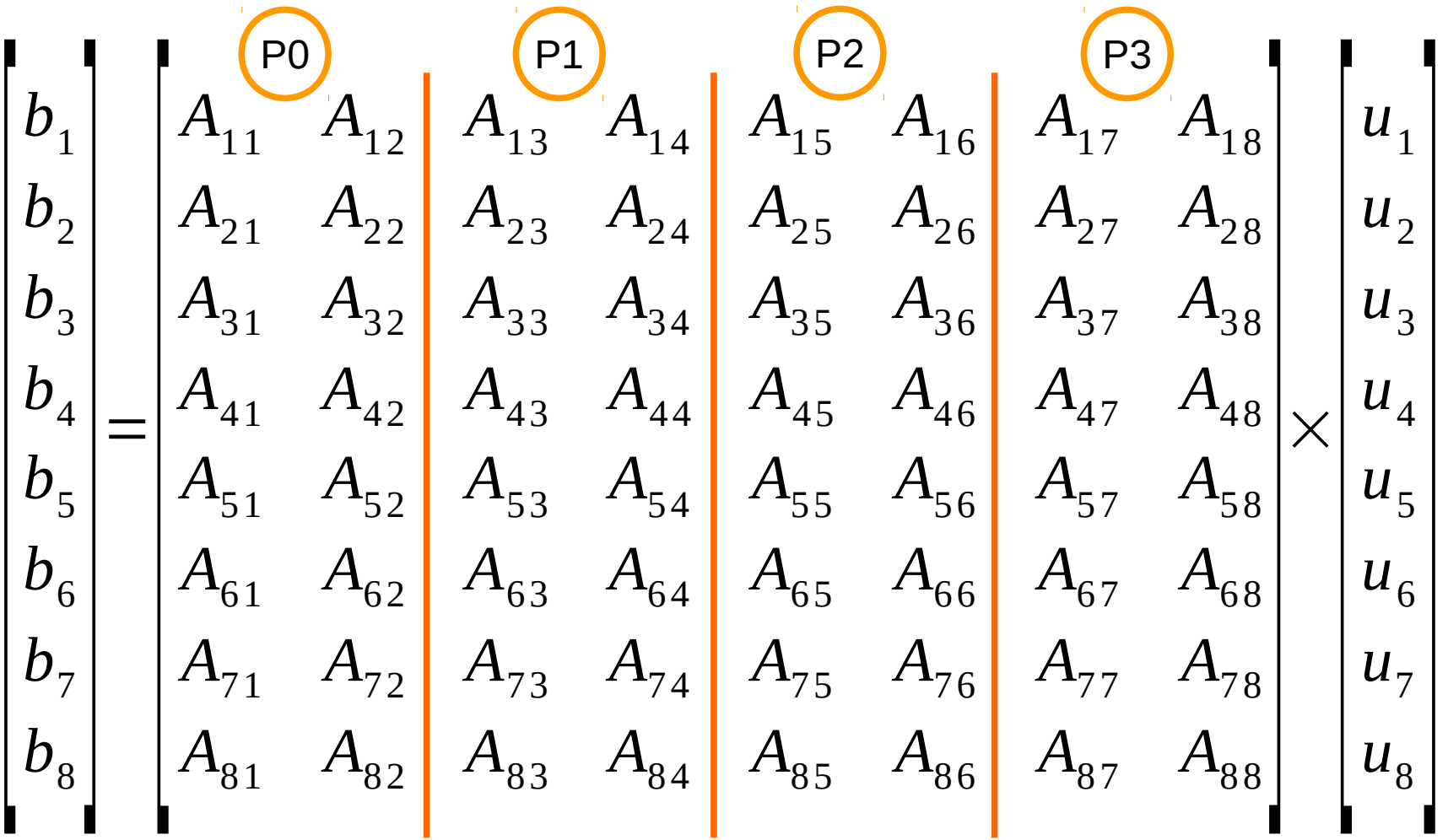


MPI : Exercice Dirigé

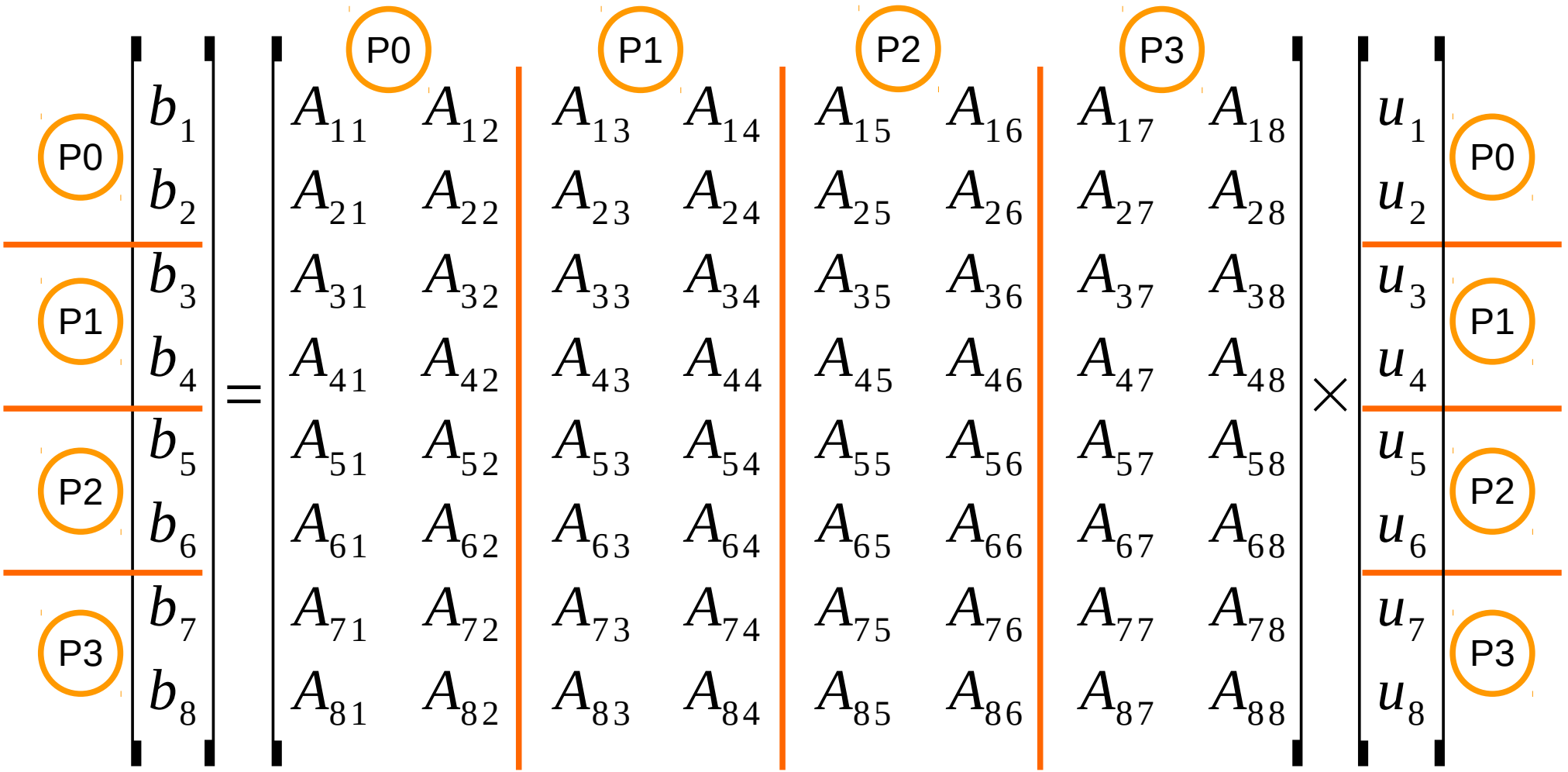
Découpage

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_8 \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} & A_{15} & A_{16} & A_{17} & A_{18} \\ A_{21} & A_{22} & A_{23} & A_{24} & A_{25} & A_{26} & A_{27} & A_{28} \\ A_{31} & A_{32} & A_{33} & A_{34} & A_{35} & A_{36} & A_{37} & A_{38} \\ A_{41} & A_{42} & A_{43} & A_{44} & A_{45} & A_{46} & A_{47} & A_{48} \\ A_{51} & A_{52} & A_{53} & A_{54} & A_{55} & A_{56} & A_{57} & A_{58} \\ A_{61} & A_{62} & A_{63} & A_{64} & A_{65} & A_{66} & A_{67} & A_{68} \\ A_{71} & A_{72} & A_{73} & A_{74} & A_{75} & A_{76} & A_{77} & A_{78} \\ A_{81} & A_{82} & A_{83} & A_{84} & A_{85} & A_{86} & A_{87} & A_{88} \end{bmatrix} \times \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \end{bmatrix}$$

MPI : Exercice Dirigé

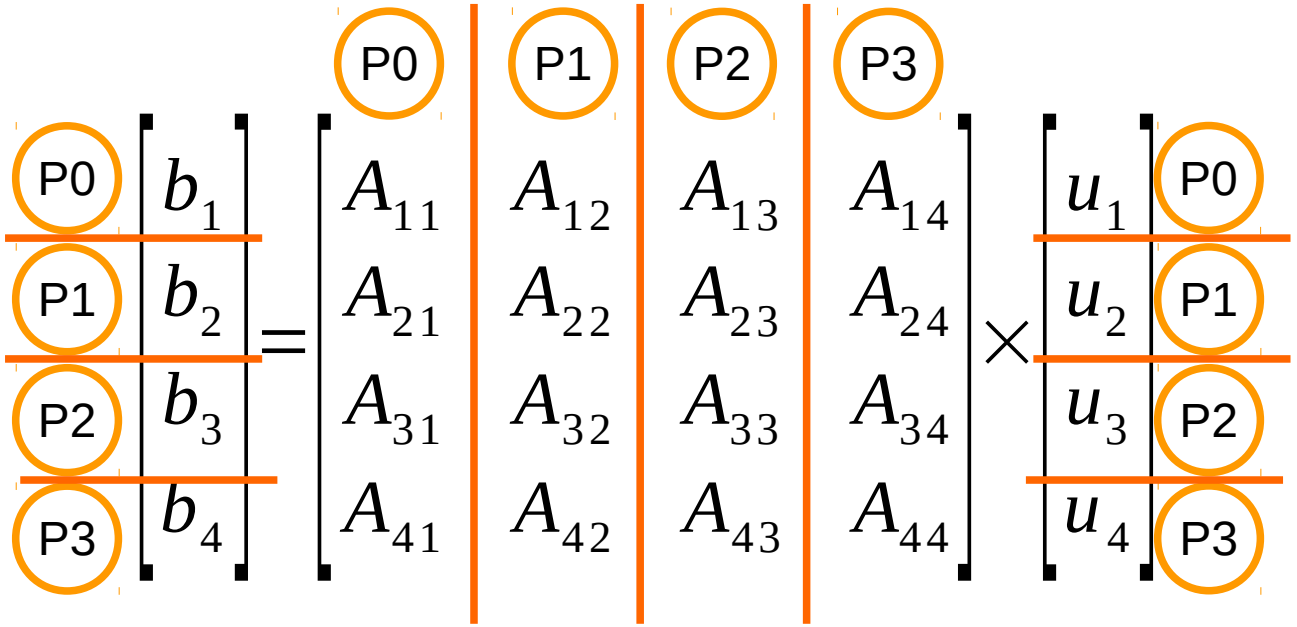


MPI : Exercice Dirigé



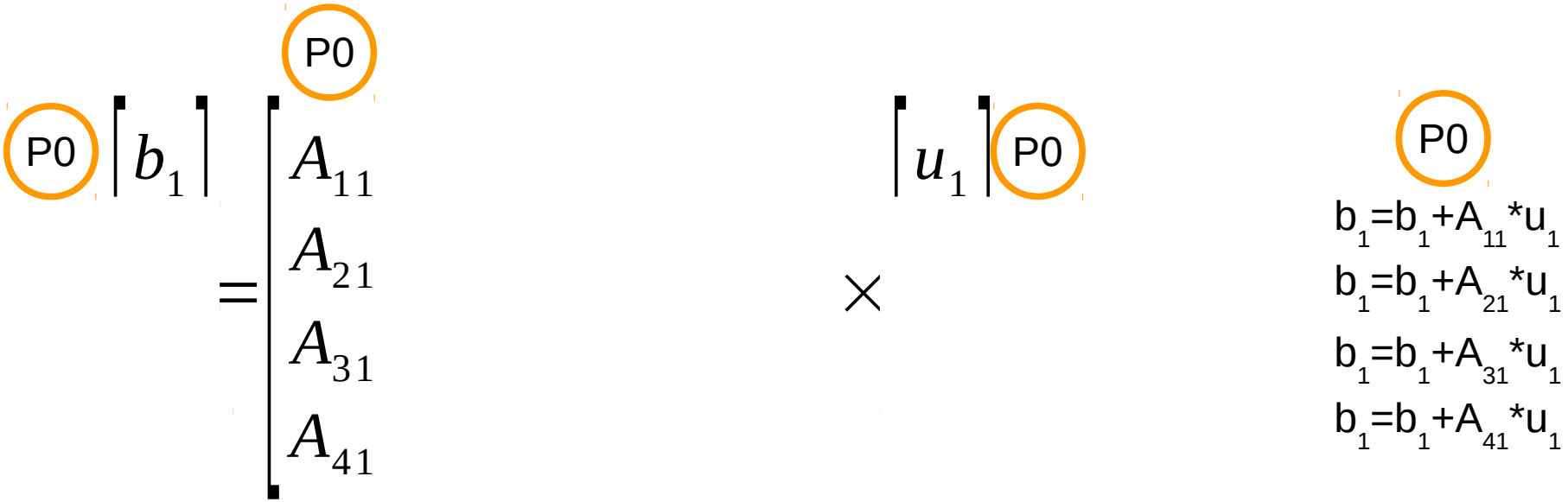
MPI : Exercice Dirigé

Communications ?



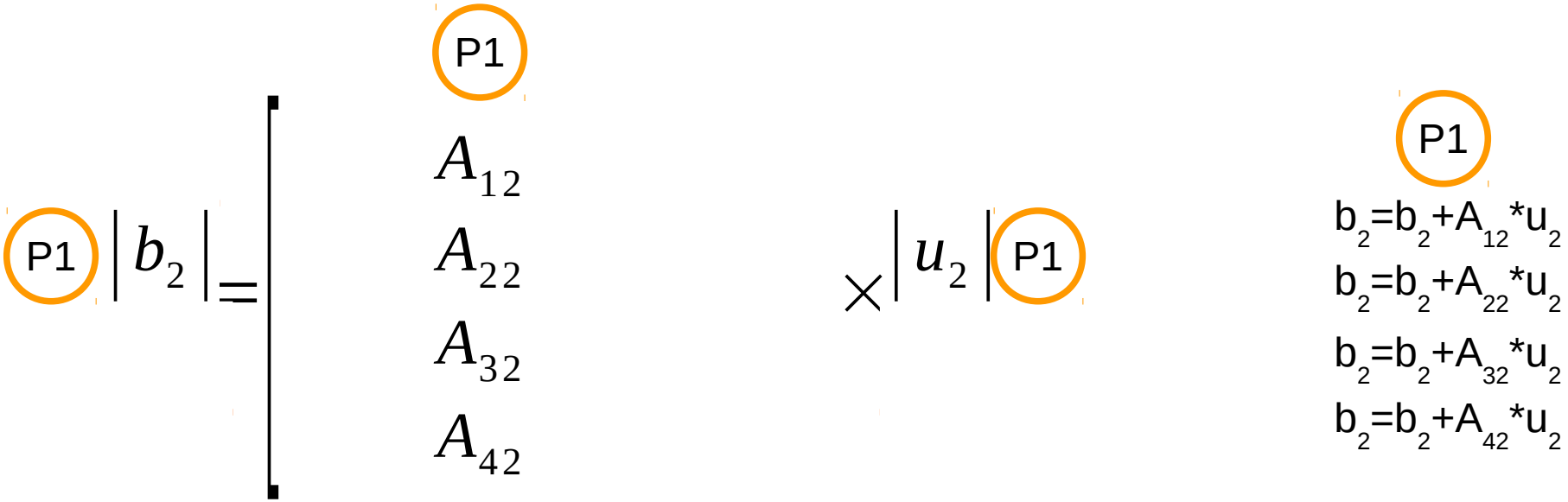
MPI : Exercice Dirigé

Communications ?



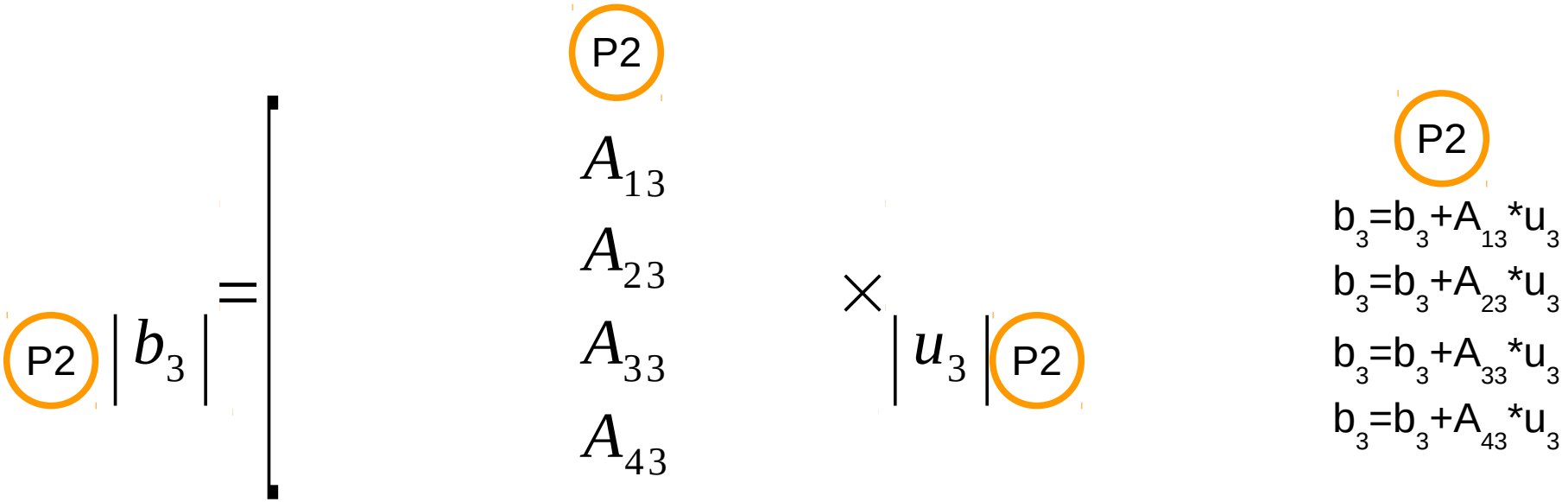
MPI : Exercice Dirigé

Communications ?



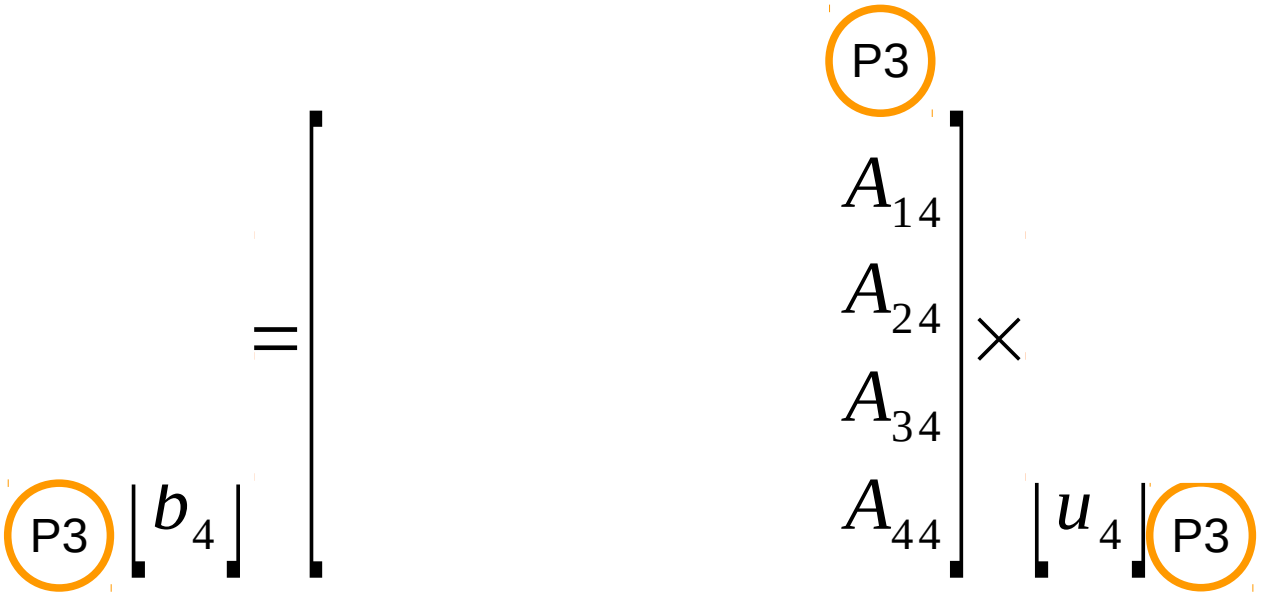
MPI : Exercice Dirigé

Communications ?



MPI : Exercice Dirigé

Communications ?



P3

$b_4 = b_4 + A_{14} * u_4$

$b_4 = b_4 + A_{24} * u_4$

$b_4 = b_4 + A_{34} * u_4$

$b_4 = b_4 + A_{44} * u_4$

MPI : Exercice Dirigé

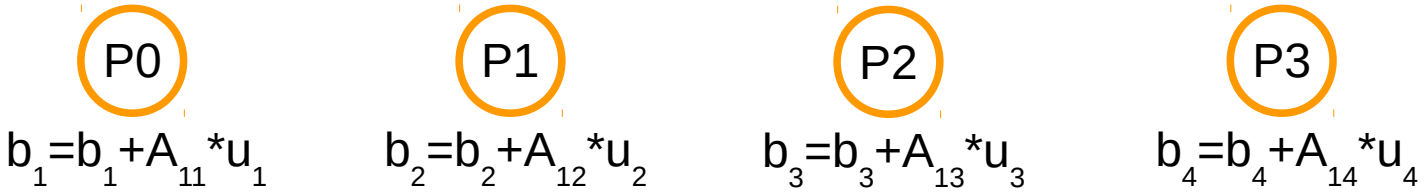
Besoin de communications ?

On remarque qu'il y a 4 opérations sur chaque processeur …

Détaillons la première opération ...

MPI : Exercice Dirigé

- Etape 1



MPI : Exercice Dirigé

- Etape 1

P0 : $b_1 = b_1 + A_{11} * u_1$

P1 : $b_2 = b_2 + A_{12} * u_2$

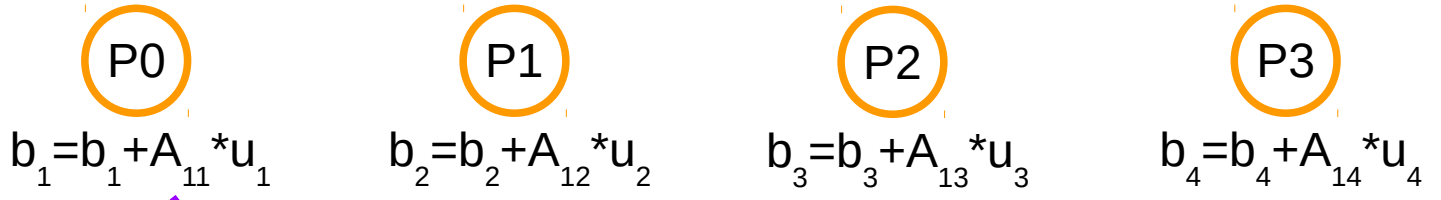
P2 : $b_3 = b_3 + A_{13} * u_3$

P3 : $b_4 = b_4 + A_{14} * u_4$

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} A_{11}u_1 + A_{12}u_2 + A_{13}u_3 + A_{14}u_4 \\ A_{21}u_1 + A_{22}u_2 + A_{23}u_3 + A_{24}u_4 \\ A_{31}u_1 + A_{32}u_2 + A_{33}u_3 + A_{34}u_4 \\ A_{41}u_1 + A_{42}u_2 + A_{43}u_3 + A_{44}u_4 \end{bmatrix}$$

MPI : Exercice Dirigé

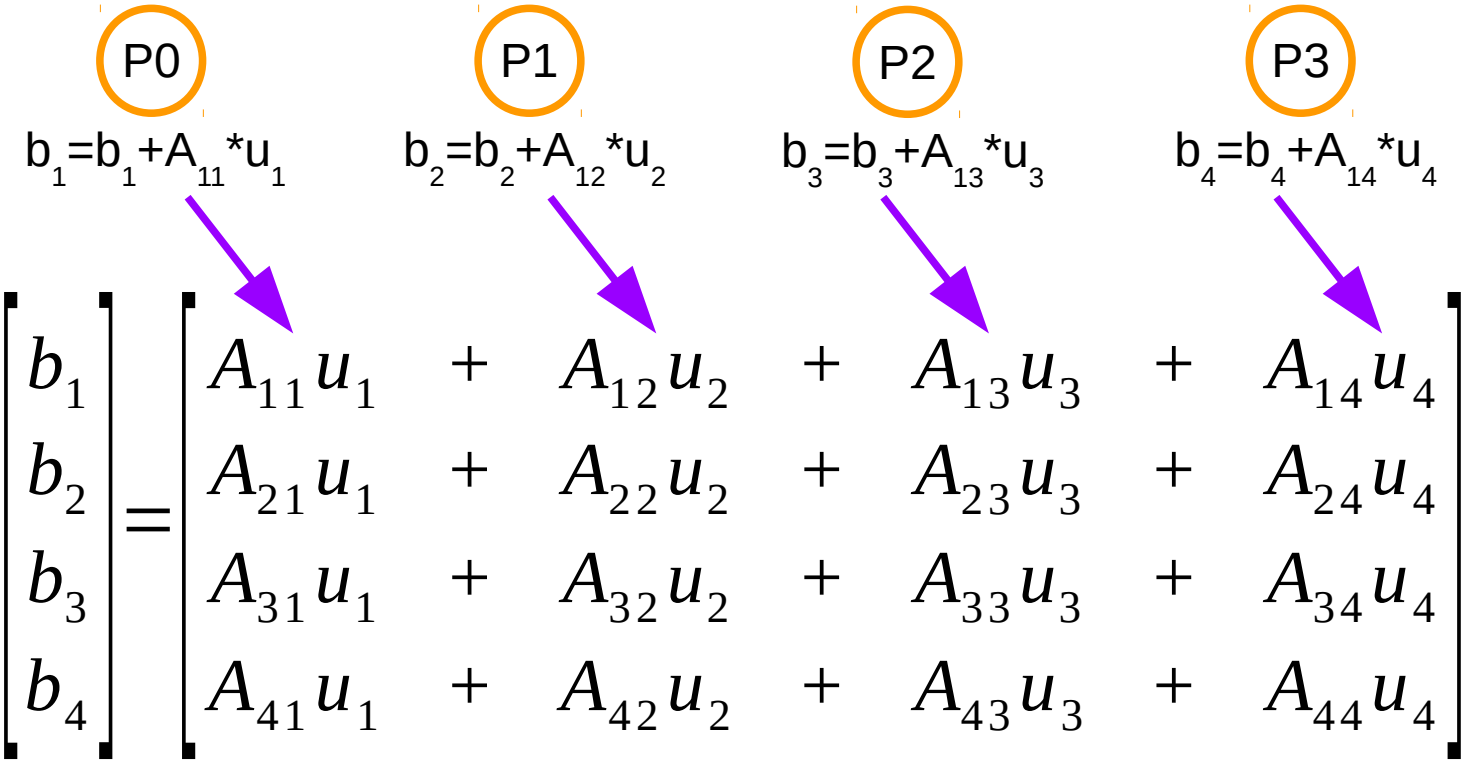
- Etape 1



$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} A_{11}u_1 + A_{12}u_2 + A_{13}u_3 + A_{14}u_4 \\ A_{21}u_1 + A_{22}u_2 + A_{23}u_3 + A_{24}u_4 \\ A_{31}u_1 + A_{32}u_2 + A_{33}u_3 + A_{34}u_4 \\ A_{41}u_1 + A_{42}u_2 + A_{43}u_3 + A_{44}u_4 \end{bmatrix}$$

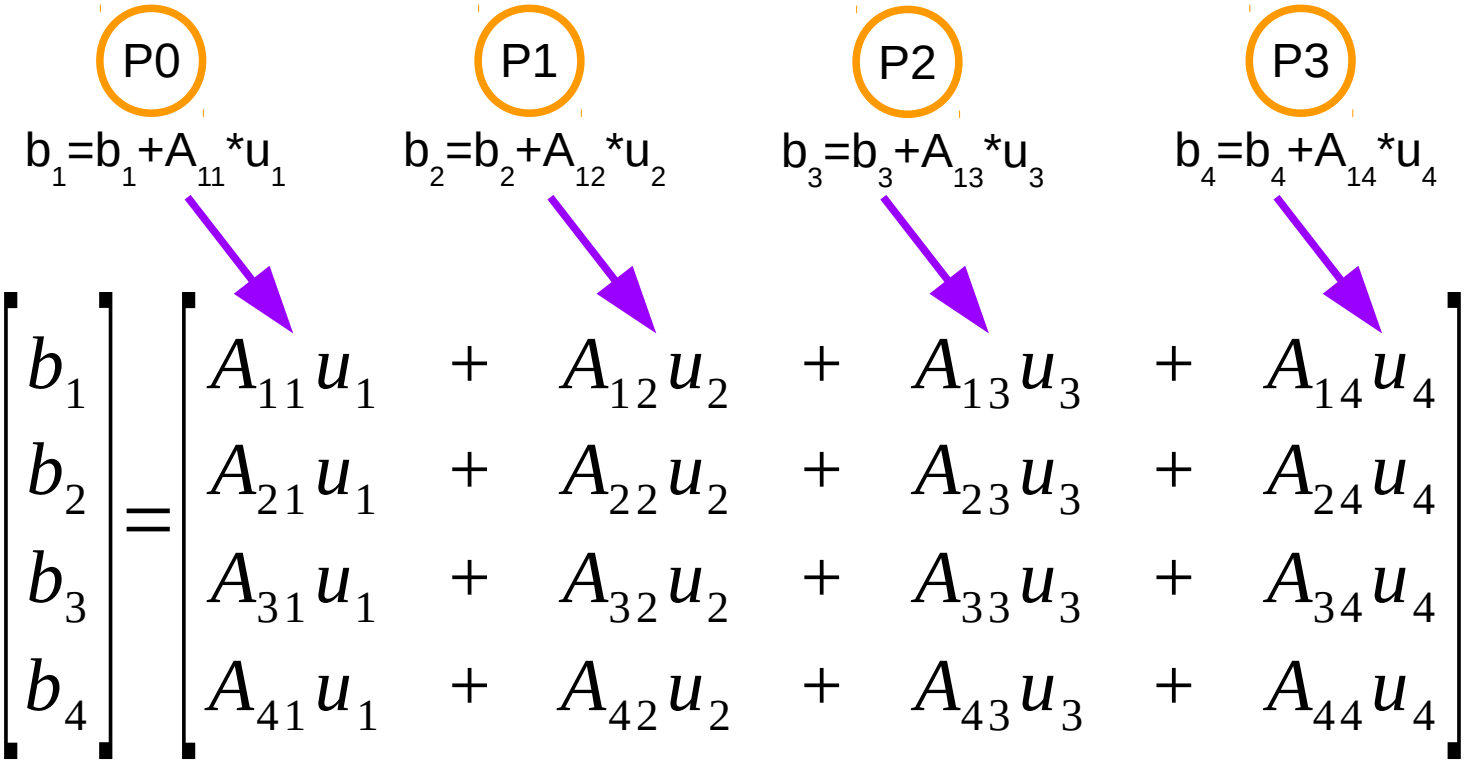
MPI : Exercice Dirigé

- Etape 1



MPI : Exercice Dirigé

- Etape 1



MPI : Exercice Dirigé

- Etape 1

P0 : $b_1 = b_1 + A_{11} * u_1$

P1 : $b_2 = b_2 + A_{12} * u_2$

P2 : $b_3 = b_3 + A_{13} * u_3$

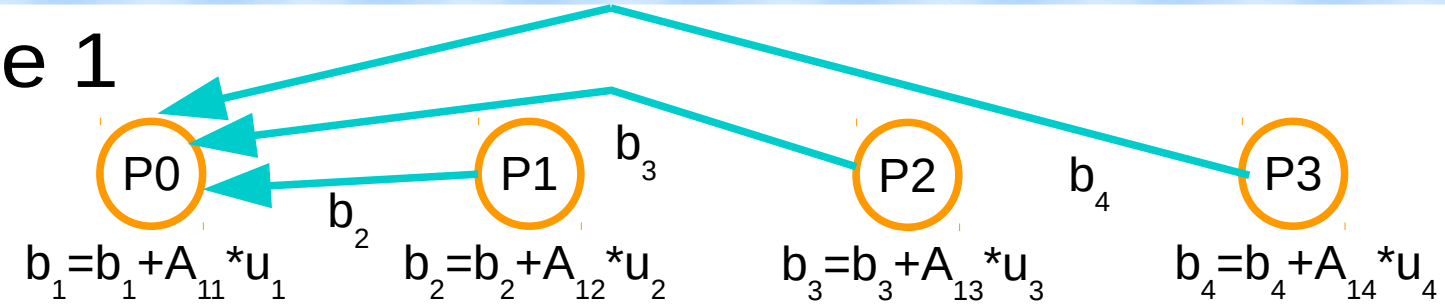
P3 : $b_4 = b_4 + A_{14} * u_4$

$b_1 = b_1 + b_2 + b_3 + b_4$

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} A_{11}u_1 + A_{12}u_2 + A_{13}u_3 + A_{14}u_4 \\ A_{21}u_1 + A_{22}u_2 + A_{23}u_3 + A_{24}u_4 \\ A_{31}u_1 + A_{32}u_2 + A_{33}u_3 + A_{34}u_4 \\ A_{41}u_1 + A_{42}u_2 + A_{43}u_3 + A_{44}u_4 \end{bmatrix}$$

MPI : Exercice Dirigé

- Etape 1



$b_1 = b_1 + b_2 + b_3 + b_4$

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} A_{11}u_1 + A_{12}u_2 + A_{13}u_3 + A_{14}u_4 \\ A_{21}u_1 + A_{22}u_2 + A_{23}u_3 + A_{24}u_4 \\ A_{31}u_1 + A_{32}u_2 + A_{33}u_3 + A_{34}u_4 \\ A_{41}u_1 + A_{42}u_2 + A_{43}u_3 + A_{44}u_4 \end{bmatrix}$$

MPI : Exercice Dirigé

Soit P processeurs (ici 4)

- Chaque processeur calcule la première ligne
- Chaque processeur envoie son premier calcul au processeur 0
- Le processeur 0 reçoit les valeurs des autres processeurs et les additionne à son propre calcul
- On obtient b_1

MPI : Exercice Dirigé

Besoin de communications ?

On remarque qu'il y a 4 opérations sur chaque processeur …

Détaillons la deuxième opération ...

MPI : Exercice Dirigé

- Etape 2

P0 : $b_1 = b_1 + A_{21} * u_1$

P1 : $b_2 = b_2 + A_{22} * u_2$

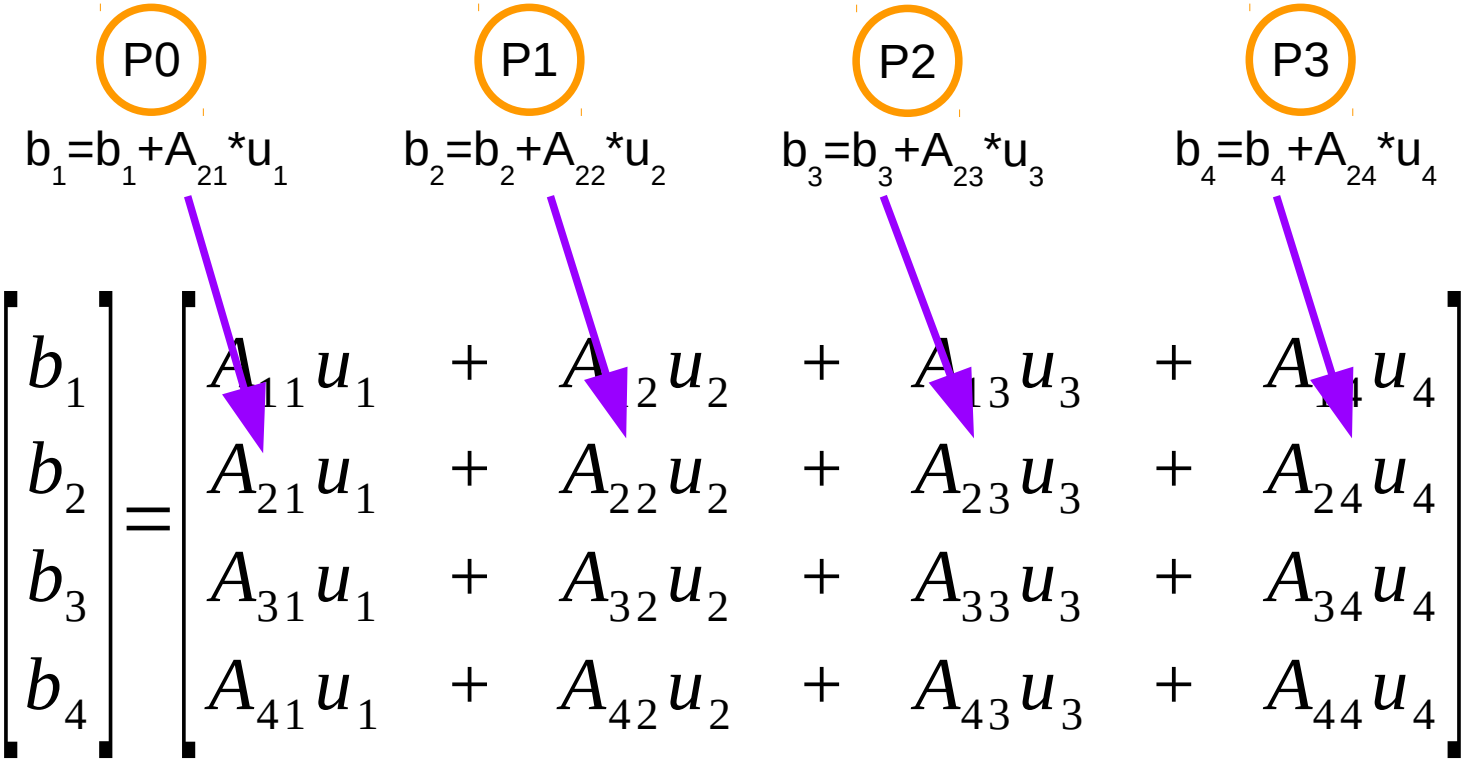
P2 : $b_3 = b_3 + A_{23} * u_3$

P3 : $b_4 = b_4 + A_{24} * u_4$

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} A_{11}u_1 + A_{12}u_2 + A_{13}u_3 + A_{14}u_4 \\ A_{21}u_1 + A_{22}u_2 + A_{23}u_3 + A_{24}u_4 \\ A_{31}u_1 + A_{32}u_2 + A_{33}u_3 + A_{34}u_4 \\ A_{41}u_1 + A_{42}u_2 + A_{43}u_3 + A_{44}u_4 \end{bmatrix}$$

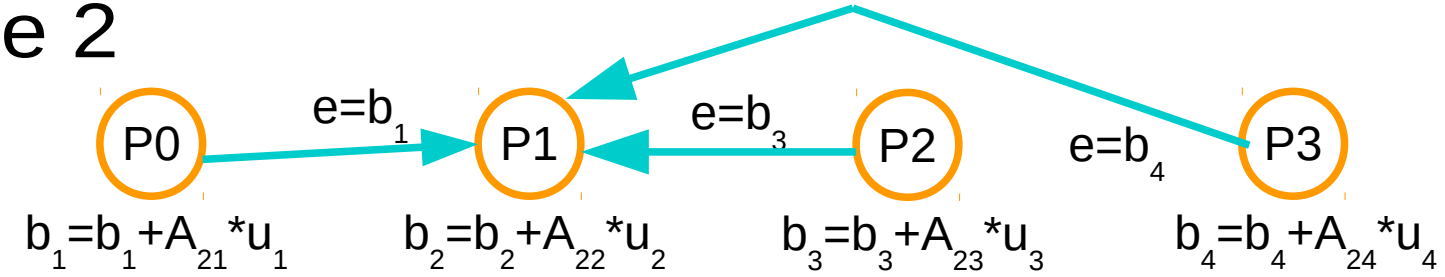
MPI : Exercice Dirigé

- Etape 2



MPI : Exercice Dirigé

- Etape 2



$b_2 = b_2 + b_1 + b_3 + b_4$

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} A_{11}u_1 + A_{12}u_2 + A_{13}u_3 + A_{14}u_4 \\ A_{21}u_1 + A_{22}u_2 + A_{23}u_3 + A_{24}u_4 \\ A_{31}u_1 + A_{32}u_2 + A_{33}u_3 + A_{34}u_4 \\ A_{41}u_1 + A_{42}u_2 + A_{43}u_3 + A_{44}u_4 \end{bmatrix}$$

MPI : Exercice Dirigé

Soit P processeurs (ici 4)

- Chaque processeur calcule la deuxième ligne
- Chaque processeur envoie son deuxième calcul au processeur 1
- Le processeur 1 reçoit les valeurs des autres processeurs et les additionne à son propre calcul
- On obtient b_2

MPI : Exercice Dirigé

Pour chaque étape e comprise entre 0 et P-1

- chaque processeur calcule localement b

- chaque processeur envoie b au processeur e

- le processeur e reçoit les valeurs des autres processeurs et les additionne à son propre calcul

Fin Pour

MPI : Exercice Dirigé

```
#include <mpi.h>
#include <stdio.h>
#define n 4  //nb lignes
#define m 4 //nb colonnes

int main(int argc,char *argv[]) {
    int error,nbproc,rang;
    MPI_Status status;

    error = MPI_Init(&argc,&argv) ;
    MPI_Comm_size(MPI_COMM_WORLD, &nbproc) ;
    MPI_Comm_rank(MPI_COMM_WORLD, &rang) ;



    MPI_Finalize();
    return(0);
}
```

MPI : Exercice Dirigé

```
#include <mpi.h>
#include <stdio.h>
#define n 4  //nb lignes
#define m 4 //nb colonnes

int main(int argc,char *argv[]) {
    int error,nbproc,rang;
    MPI_Status status;

    error = MPI_Init(&argc,&argv) ;
    MPI_Comm_size(MPI_COMM_WORLD, &nbproc) ;
    MPI_Comm_rank(MPI_COMM_WORLD, &rang) ;



    MPI_Finalize();
    return(0);
}
```

MPI : Exercice Dirigé

```
#include <mpi.h>
#include <stdio.h>
#define n 4  //nb lignes
#define m 4 //nb colonnes

int main(int argc,char *argv[]) {
    int error,nbproc,rang;
    MPI_Status status;

    error = MPI_Init(&argc,&argv) ;
    MPI_Comm_size(MPI_COMM_WORLD, &nbproc) ;
    MPI_Comm_rank(MPI_COMM_WORLD, &rang) ;



    MPI_Finalize();
    return(0);
}
```

MPI : Exercice Dirigé

Les données A et u ?

- On peut récupérer des données (par exemple à partir d'un fichier)
- On peut créer des données pour la simulation

```
float floatrand_inf_sup(float inf, float sup){
    return (rand()/(float)RAND_MAX)*(sup-inf)+inf;
}
```

MPI : Exercice Dirigé

```
int error,nbproc,nproc,rang,i,j,k,p,e;
float b,u,r;  // r sert pour la reception
float A[n+1]; // on a besoin d'une seule colonne de la matrice
srand(time(NULL)+rang); // initialisation de rand

// chaque processeur à un morceau de u et de A
u=floatrand_inf_sup(1.,3.);
for (i=1;i<=n;i++) A[i]=floatrand_inf_sup(1.,10.);

for (e=0;e<nbproc;e++) {   // etapes de calcul
    b=0.0; // initialisation du vecteur b

    b=b+A[e+1]*u; // calcul local sur chaque processeur
```

MPI : Exercice Dirigé

```
int error,nbproc,nproc,rang,i,j,k,p,e;
float b,u,r;  // r sert pour la reception
float A[n+1]; // on a besoin d'une seule colonne de la matrice
srand(time(NULL)+rang); // initialisation de rand

// chaque processeur à un morceau de u et de A
u=floatrand_inf_sup(1.,3.);
for (i=1;i<=n;i++) A[i]=floatrand_inf_sup(1.,10.);

for (e=0;e<nbproc;e++) {   // etapes de calcul
    b=0.0; // initialisation du vecteur b

    b=b+A[e+1]*u; // calcul local sur chaque processeur
```

MPI : Exercice Dirigé

```
int error,nbproc,nproc,rang,i,j,k,p,e;
float b,u,r;  // r sert pour la reception
float A[n+1]; // on a besoin d'une seule colonne de la matrice
srand(time(NULL)+rang); // initialisation de rand

// chaque processeur à un morceau de u et de A
u=floatrand_inf_sup(1.,3.);
for (i=1;i<=n;i++) A[i]=floatrand_inf_sup(1.,10.);

for (e=0;e<nbproc;e++) {   // etapes de calcul
    b=0.0; // initialisation du vecteur b

    b=b+A[e+1]*u; // calcul local sur chaque processeur
```

MPI : Exercice Dirigé

```
// envoi et reception des donnees des autres processeurs
if(rang!=e) MPI_Send(&b,1,MPI_FLOAT,e,102,MPI_COMM_WORLD);
else {
        for (p=0;p<nbproc;p++) {
      if(p!=rang) {
            MPI_Recv(&r,1,MPI_FLOAT,p,102,MPI_COMM_WORLD,&status);
            b=b+r; // mise à jour de b
          }
   }
}
if (rang==e) printf("Sur %d, la valeur de b[%d]=%f\n",rang,rang,b);
} // fin des étapes de calcul
```

MPI : Exercice Dirigé

```
// envoi et reception des donnees des autres processeurs
if(rang!=e) MPI_Send(&b,1,MPI_FLOAT,e,102,MPI_COMM_WORLD);
else {
        for (p=0;p<nbproc;p++) {
      if(p!=rang) {
            MPI_Recv(&r,1,MPI_FLOAT,p,102,MPI_COMM_WORLD,&status);
            b=b+r; // mise à jour de b
          }
   }
}
if (rang==e) printf("Sur %d, la valeur de b[%d]=%f\n",rang,rang,b);
} // fin des étapes de calcul
```

MPI : communications point à point

```
mpirun -np 4 ./Prod_MatVec4Rand

Sur 0, la valeur de b[0]=41.325500
Sur 1, la valeur de b[1]=26.730249
Sur 2, la valeur de b[2]=32.900150
Sur 3, la valeur de b[3]=58.287636
```

MPI : communications point à point

mpirun -np 8 ./Prod_MatVec4Rand

?????

MPI : Exercice Dirigé

```
#include <mpi.h>
#include <stdio.h>
#define n 8 //nb lignes
#define m 8 //nb colonnes

int main(int argc,char *argv[]) {
    int error,nbproc,rang;
    MPI_Status status;

    error = MPI_Init(&argc,&argv) ;
    MPI_Comm_size(MPI_COMM_WORLD, &nbproc) ;
    MPI_Comm_rank(MPI_COMM_WORLD, &rang) ;



    MPI_Finalize();
    return(0);
}
```

MPI : Exercice Dirigé

```
int error,nbproc,nproc,rang,i,j,k,p,e;
nproc=n/nbproc; // on suppose n divisible par nbproc
float b[nproc+1],u[nproc+1],r[nproc+1];  // r sert pour la reception
float A[nbproc+1][nbproc+1]; // on a besoin d'un morceau de la matrice
srand(time(NULL)+rang); // initialisation de rand
// chaque processeur à un morceau de u et de A
for (j=1;j<=nproc;j++) {
    u[j]=floatrand_inf_sup(1.,10.);
    for (i=1;i<=nproc;i++) A[i][j]=floatrand_inf_sup(1.,10.); }

for (e=0;e<nbproc;e++) {   // etapes de calcul
    for (j=1;j<=nproc;j++) b[j]=0.0; // initialisation du vecteur b
    for (j=1;j<=nproc;j++)
        for (i=1;i<=nproc;i++) b[i]=b[i]+A[i][j]*u[j];
```

MPI : Exercice Dirigé

```
int error,nbproc,nproc,rang,i,j,k,p,e;
nproc=n/nbproc; // on suppose n divisible par nbproc
float b[nproc+1],u[nproc+1],r[nproc+1];  // r sert pour la reception
float A[nbproc+1][nbproc+1]; // on a besoin d'un morceau de la matrice
srand(time(NULL)+rang); // initialisation de rand
// chaque processeur à un morceau de u et de A
for (j=1;j<=nproc;j++) {
    u[j]=floatrand_inf_sup(1.,10.);
    for (i=1;i<=nproc;i++) A[i][j]=floatrand_inf_sup(1.,10.); }

for (e=0;e<nbproc;e++) {   // etapes de calcul
    for (j=1;j<=nproc;j++) b[j]=0.0; // initialisation du vecteur b
    for (j=1;j<=nproc;j++)
        for (i=1;i<=nproc;i++) b[i]=b[i]+A[i][j]*u[j];
```

MPI : Exercice Dirigé

```
int error,nbproc,nproc,rang,i,j,k,p,e;
nproc=n/nbproc; // on suppose n divisible par nbproc
float b[nproc+1],u[nproc+1],r[nproc+1];  // r sert pour la reception
float A[nbproc+1][nbproc+1]; // on a besoin d'un morceau de la matrice
srand(time(NULL)+rang); // initialisation de rand
// chaque processeur à un morceau de u et de A
for (j=1;j<=nproc;j++) {
    u[j]=floatrand_inf_sup(1.,10.);
    for (i=1;i<=nproc;i++) A[i][j]=floatrand_inf_sup(1.,10.); }

for (e=0;e<nbproc;e++) {   // etapes de calcul
    for (j=1;j<=nproc;j++) b[j]=0.0; // initialisation du vecteur b
    for (j=1;j<=nproc;j++)
        for (i=1;i<=nproc;i++) b[i]=b[i]+A[i][j]*u[j];
```

MPI : Exercice Dirigé

```
// envoi et reception des donnees des autres processeurs
if(rang!=e) MPI_Send(b,nproc+1,MPI_FLOAT,e,102,MPI_COMM_WORLD);
else {
        for (p=0;p<nbproc;p++) {
        if(p!=rang) {
            MPI_Recv(r,nproc+1,MPI_FLOAT,p,102,MPI_COMM_WORLD,&status);
            for (i=1;i<=nproc;i++) b[i]=b[i]+r[i]; // mise à jour de b
          }
    }
}
if (rang==e) for (j=1;j<=nproc;j++) printf("Sur %d, la valeur de b[%d]=%f\n",rang,
(rang*(nbproc-2))+j,b[j]);
} // fin des étapes de calcul
```

MPI : Exercice Dirigé

```
// envoi et reception des donnees des autres processeurs
if(rang!=e) MPI_Send(b,nproc+1,MPI_FLOAT,e,102,MPI_COMM_WORLD);
else {
        for (p=0;p<nbproc;p++) {
        if(p!=rang) {
            MPI_Recv(r,nproc+1,MPI_FLOAT,p,102,MPI_COMM_WORLD,&status);
            for (i=1;i<=nproc;i++) b[i]=b[i]+r[i];   // mise à jour de b
          }
   }
 }
} // fin des étapes de calcul
for (j=1;j<=nproc;j++)
      printf("Sur %d, la valeur de b[%d]=%f\n",rang,j+rang*nproc,b[j]);
```

MPI : communications point à point

mpirun -np 4 ./Prod_MatVec8Rand

Sur 0, la valeur de b[1]=17.629353

Sur 0, la valeur de b[2]=37.377129

Sur 2, la valeur de b[5]=38.817146

Sur 2, la valeur de b[6]=47.973507

Sur 1, la valeur de b[3]=126.732361

Sur 3, la valeur de b[7]=205.900513

Sur 3, la valeur de b[8]=254.324127

Sur 1, la valeur de b[4]=130.649384

MPI : communications point à point

mpirun -np 8 ./Prod_MatVec8Rand

Sur 0, la valeur de b[1]=8.261622

Sur 1, la valeur de b[2]=12.445882

Sur 2, la valeur de b[3]=18.776890

Sur 3, la valeur de b[4]=1.861112

Sur 4, la valeur de b[5]=22.117266

Sur 6, la valeur de b[7]=74.739777

Sur 5, la valeur de b[6]=12.634913

Sur 7, la valeur de b[8]=182.181335

MPI : communications point à point

mpirun -np 2 ./Prod_MatVec8Rand

Sur 0, la valeur de b[1]=8.261622

Sur 1, la valeur de b[2]=12.445882

Sur 2, la valeur de b[3]=18.776890

Sur 3, la valeur de b[4]=1.861112

Sur 4, la valeur de b[5]=22.117266

Sur 6, la valeur de b[7]=74.739777

Sur 5, la valeur de b[6]=12.634913

Sur 7, la valeur de b[8]=182.181335