Portage d'un code sous MPI (Message Passing Interface)

Atelier T8.A10

Thierry GARCIA

Portage d'un code sous 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
- http://calcul.math.cnrs.fr/Documents/Ecoles/LEM2I/Mod3/paral.pdf

Portage d'un code sous MPI (Message Passing Interface)

PRINCIPAL OBJECTIF

- Comprendre comment porter un code séquentiel résolvant un problème simple dans un environnement parallèle en utilisant les primitives de communication MPI.

Portage d'un code sous MPI (Message Passing Interface)

DESCRIPTION :

- porter un code séquentiel résolvant un problème simple de calcul scientifique, dans un environnement parallèle, en utilisant les primitives de communication MPI ;
- importance de la communication et de la synchronisation des machines dans le cas d'une résolution synchrone ou asynchrone.

Portage d'un code sous MPI (Message Passing Interface)

Code (pour ceux qui ont des machines) :

télécharger T8_A10.zip sur (donner l'adresse)

décompresser le fichier

Portage d'un code sous MPI (Message Passing Interface)

```
! Déclaration des variables
!============================================================
real*8,dimension (141,141,141):: u
real*8,dimension (141,141,141):: f
real*8,dimension (141):: x,y,z
integer :: Nbtemp,m,p,n,nn,i,j,k,t,tt,num,iter1,iter2
integer :: iterG1,iterG2
real*8 :: Temps,lamda1,Rij,Uconst,cmv1,ct2
real*8 :: lamda2,cmv2,longx
real*8 :: epsi,h,kk,A1,codiag1, diag1,ct11,ct31prim
real*8 :: A2,codiag2, diag2
real*8 :: start,normmax1,normmax2
```

Que va-t-on ou peut-on paralléliser ?

Portage d'un code sous MPI (Message Passing Interface)

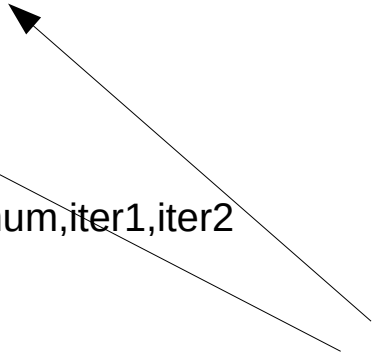
```
! Déclaration des variables
!============================================================
real*8,dimension (141,141,141):: u
real*8,dimension (141,141,141):: f
real*8,dimension (141):: x,y,z
integer :: Nbtemp,m,p,n,nn,i,j,k,t,tt,num,iter1,iter2
integer :: iterG1,iterG2
real*8 :: Temps,lamda1,Rij,Uconst,cmv1,ct2
real*8 :: lamda2,cmv2,longx
real*8 :: epsi,h,kk,A1,codiag1, diag1,ct11,ct31prim
real*8 :: A2,codiag2, diag2
real*8 :: start,normmax1,normmax2
```

Que va-t-on ou peut-on paralléliser ?

Dans la partie données du programme fortran, il y a des tableaux ...

Portage d'un code sous MPI (Message Passing Interface)

```
! Déclaration des variables
!=============================================================
real*8,dimension (141,141,141):: u
real*8,dimension (141,141,141):: f
real*8,dimension (141):: x,y,z
integer :: Nbtemp,m,p,n,nn,i,j,k,t,tt,num,iter1,iter2
integer :: iterG1,iterG2
real*8 :: Temps,lamda1,Rij,Uconst,cmv1,ct2
real*8 :: lamda2,cmv2,longx
real*8 :: epsi,h,kk,A1,codiag1, diag1,ct11,ct31prim
real*8 :: A2,codiag2, diag2
real*8 :: start,normmax1,normmax2
```

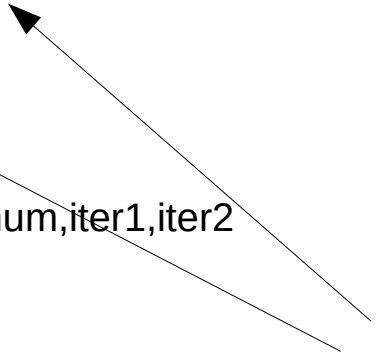


Que va-t-on ou peut-on paralléliser ?

Dans la partie données du programme fortran, il y a des tableaux et un vecteur ...

Portage d'un code sous MPI (Message Passing Interface)

```
! Déclaration des variables
!============================================================
real*8,dimension (141,141,141):: u
real*8,dimension (141,141,141):: f
real*8,dimension (141):: x,y,z
integer :: Nbtemp,m,p,n,nn,i,j,k,t,tt,num,iter1,iter2
integer :: iterG1,iterG2
real*8 :: Temps,lamda1,Rij,Uconst,cmv1,ct2
real*8 :: lamda2,cmv2,longx
real*8 :: epsi,h,kk,A1,codiag1, diag1,ct11,ct31prim
real*8 :: A2,codiag2, diag2
real*8 :: start,normmax1,normmax2
```



Le but va être de simuler avec des valeurs supérieures à 141 !!!

Sur une seule machine, les ressources risquent d'être insuffisantes et mais surtout le temps de calcul risque d'être très long.

Portage d'un code sous MPI (Message Passing Interface)

! Initialisation

! Définition constantes physiques

! Pas de discrétisation h

! Initialisation second membre

! Coefficients physiques concernant la zone Omega 1

! Coefficients physiques concernant la zone Omega 2

! C.L DIRICHLET PARTOUT SAUF A L'INTERFACE

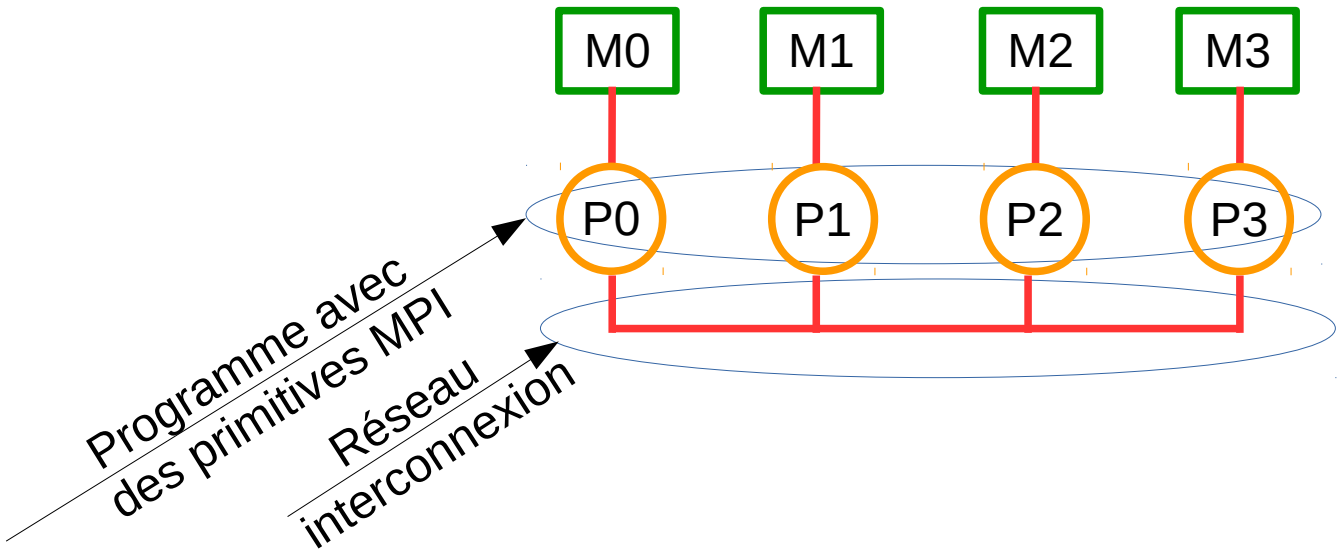
! Conditions initiales

Les initialisations, la définition des constantes et des coefficients physiques, le pas de discrétisation et les conditions initiales ne semblent pas être important pour notre parallélisation à venir

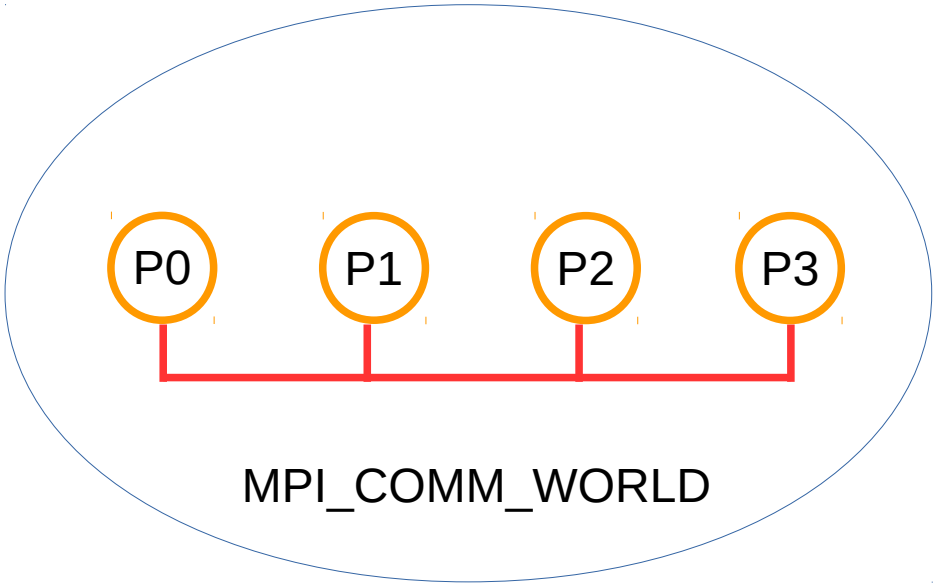
Portage d'un code sous MPI (Message Passing Interface)

- 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

Portage d'un code sous MPI (Message Passing Interface)



Portage d'un code sous MPI (Message Passing Interface)



MPI utilise des communicateurs.

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

Portage d'un code sous MPI (Message Passing Interface)

En FORTRAN :

```
INCLUDE "mpif.h"          ! entête mpi fortran (obligatoire)
INTEGER :: nbproc, ierr, rang
CALL MPI_Init(ierr)
CALL MPI_Comm_size(MPI_COMM_WORLD,nbproc,ierr)
CALL MPI_Comm_rank(MPI_COMM_WORLD,rang,ierr)
……
CALL MPI_Finalize(ierr)
STOP
END PROGRAM
```

Portage d'un code sous MPI (Message Passing Interface)

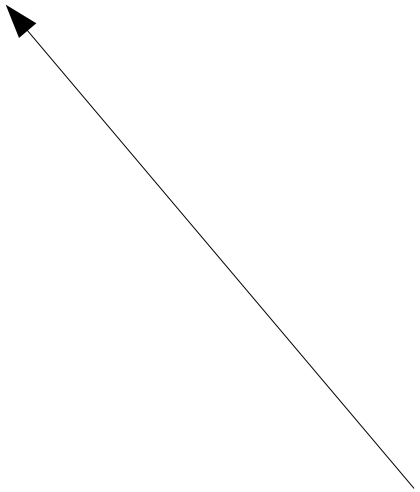
```
include "mpif.h"          ! entête mpi fortran (obligatoire)
!=====================================================================
```

On rajoute (voir fichier ExerciceMPI_1.f90) :

- La librairie

Portage d'un code sous MPI (Message Passing Interface)

```
      include "mpif.h"          ! entête mpi fortran (obligatoire)
!=====================================================================
! MPI variables
!=====================================================================
      INTEGER :: nbproc, ierr, comm, numproc
```

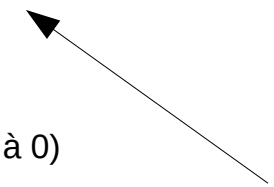


On rajoute (voir fichier ExerciceMPI_1.f90) :

- La librairie
- Une partie contenant les variables utiles à l'utilisation de MPI et de la parallélisation

Portage d'un code sous MPI (Message Passing Interface)

```
      include "mpif.h"          ! entête mpi fortran (obligatoire)
!=====================================================================
! MPI variables
!=====================================================================
      INTEGER :: nbproc, ierr, comm, numproc
!=====================================================================
! MPI initialisation
!=====================================================================
      CALL MPI_Init(ierr)    ! recuperation des informations mpi
      comm = MPI_COMM_WORLD
      CALL MPI_Comm_size(comm,nbproc,ierr)
!     recup nb de processeurs : nbproc
      CALL MPI_Comm_rank(comm,rang,ierr)
!     num du processeurs courant : rang (démarre à 0)
```



On rajoute (voir fichier ExerciceMPI_1.f90) :

- La librairie
- Une partie contenant les variables utiles à l'utilisation de MPI et de la parallélisation
- L'initialisation MPI

Portage d'un code sous MPI (Message Passing Interface)

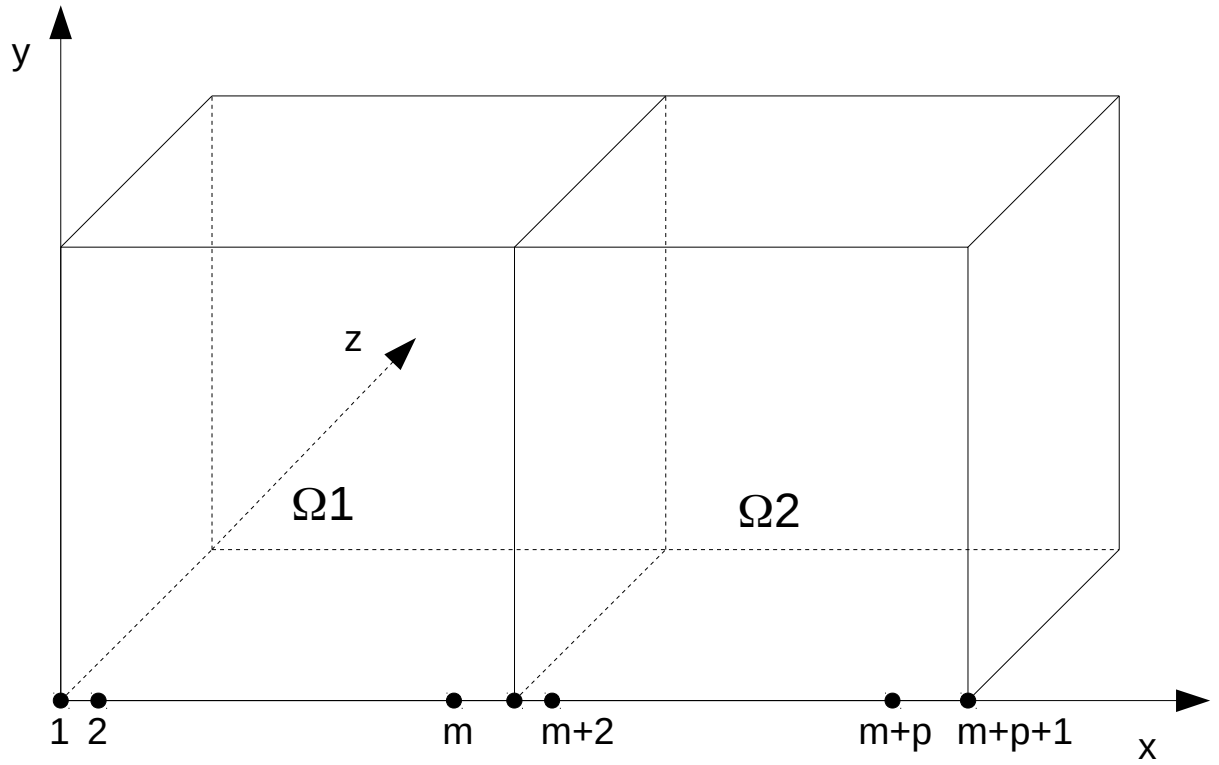
```
……...
  start=stop_time-start_time
  write (*,*) 'temps=', start
  write (*,*) 'normmax1=',normmax1
  write (*,*) 'normmax2=',normmax2
  write (*,*) 'iter1=',iterG1
  write (*,*) 'iter2=',iterG2
!=====================================================================
! MPI Finalisation
  CALL MPI_Finalize(ierr)
!=====================================================================
stop
end
```

On n'oublie pas :

- De finaliser l'utilisation de MPI (libérer les processeur, tout clôturer) à la fin du programme.

Portage d'un code sous MPI (Message Passing Interface)

Représentation du domaine global :



Portage d'un code sous MPI (Message Passing Interface)

```
SUBROUTINE PTSINT1(f,u,diag1,codiag1,nn,n,m,normmax1,iter1,epsi)
…….
do while( normmax1.gt.epsi)
      normmax1=0.
      do k=2,nn+1
            do j=2,n+1
                  do i=2,m
                     SS=….
                     RR=dabs(SS-u(i,j,k))
                     if (RR .gt. normmax1)then
                              normmax1=RR
                      endif
                      u(i,j,k)=SS
                   enddo
            enddo
      enddo
…….
```

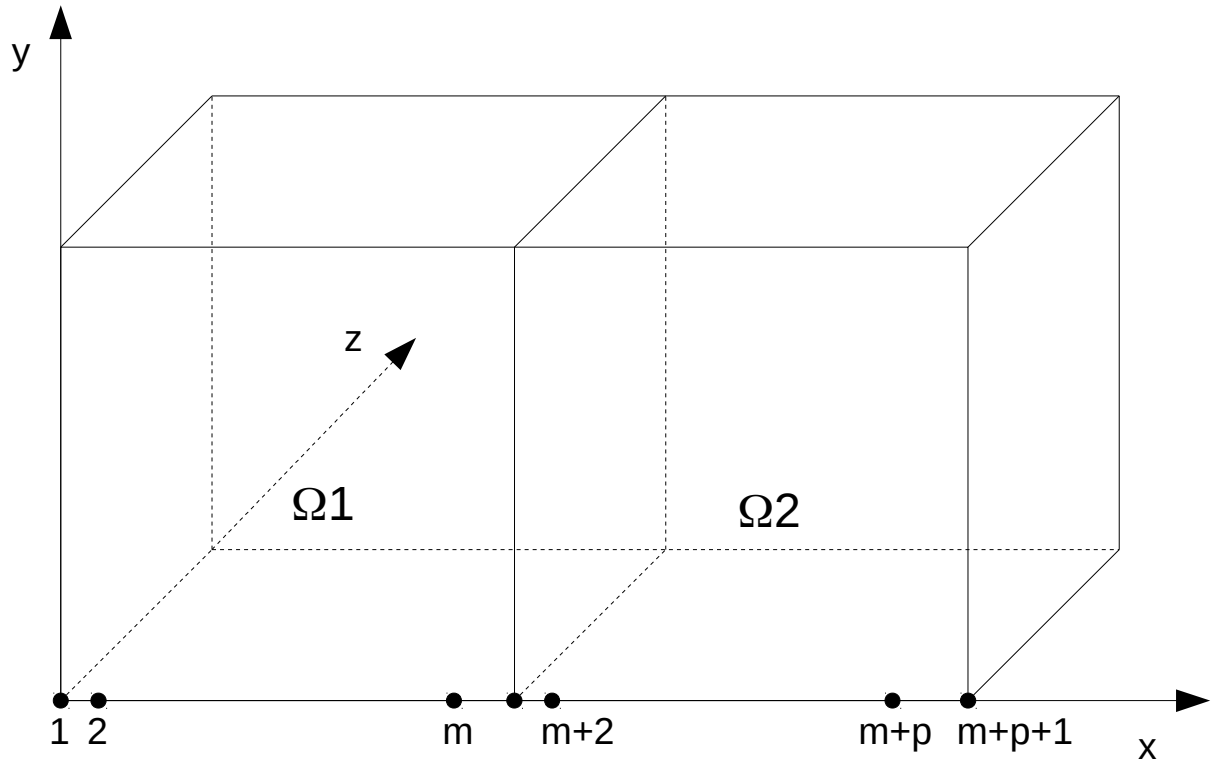
Ici on calcule les points intérieurs.

a) On remarque qu'on résout en i, j et k (3 boucles imbriquées)

b) Qu'on réitère ces calculs tant que la normmax1 est > à epsilon

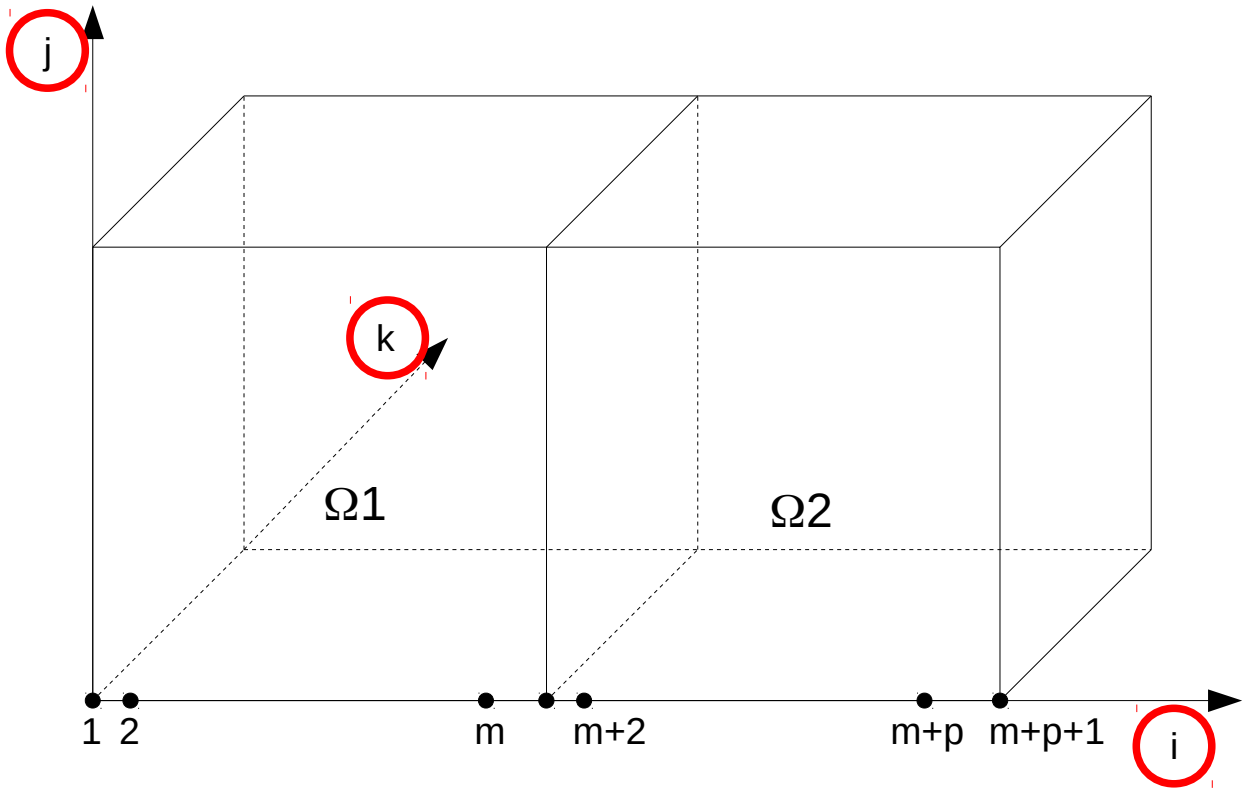
Portage d'un code sous MPI (Message Passing Interface)

a) On remarque qu'on résout en i, j et k (3 boucles imbriquées)



Portage d'un code sous MPI (Message Passing Interface)

a) On remarque qu'on résout en i, j et k (3 boucles imbriquées)

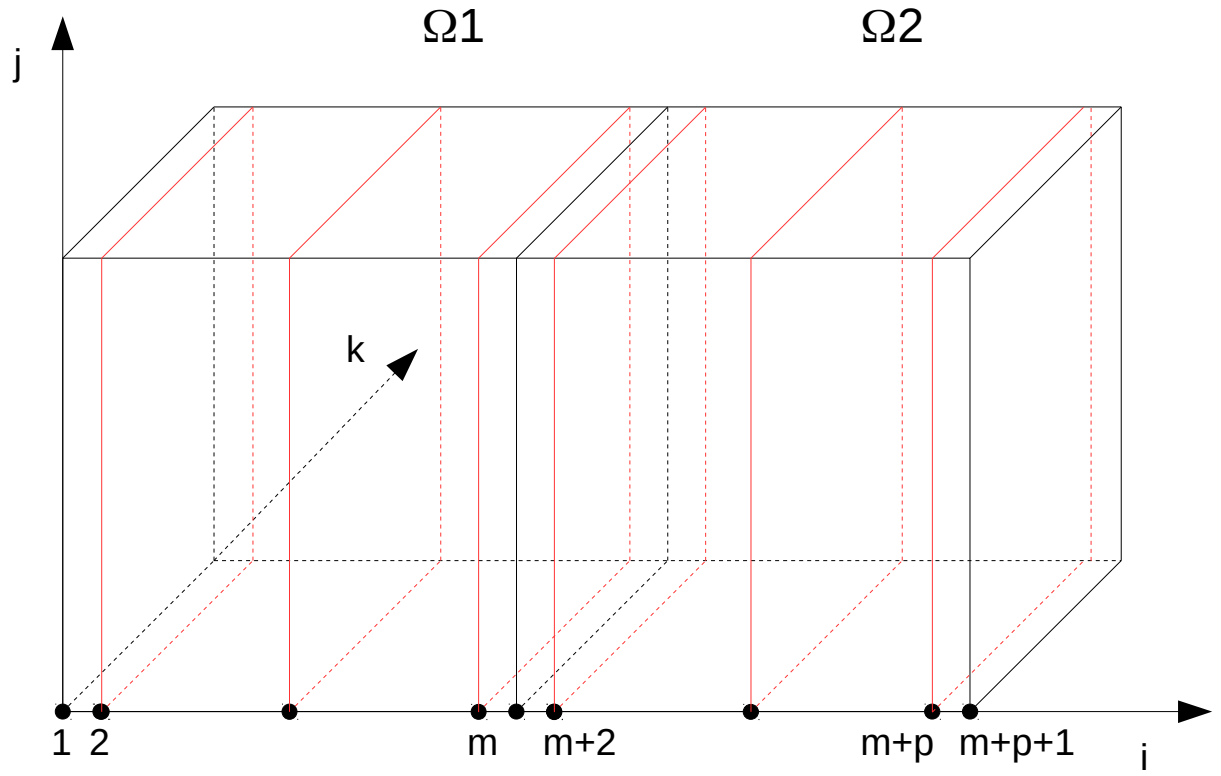


Portage d'un code sous MPI (Message Passing Interface)

Comment découper notre problème en sous-problèmes plus petit ?

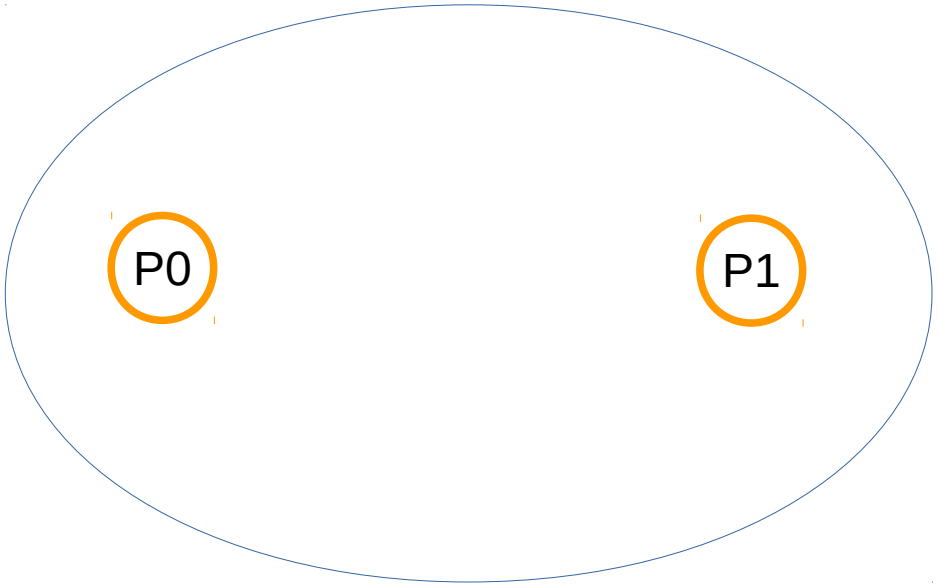
Portage d'un code sous MPI (Message Passing Interface)

On peut partager notre domaine en tranche en chaque i ...



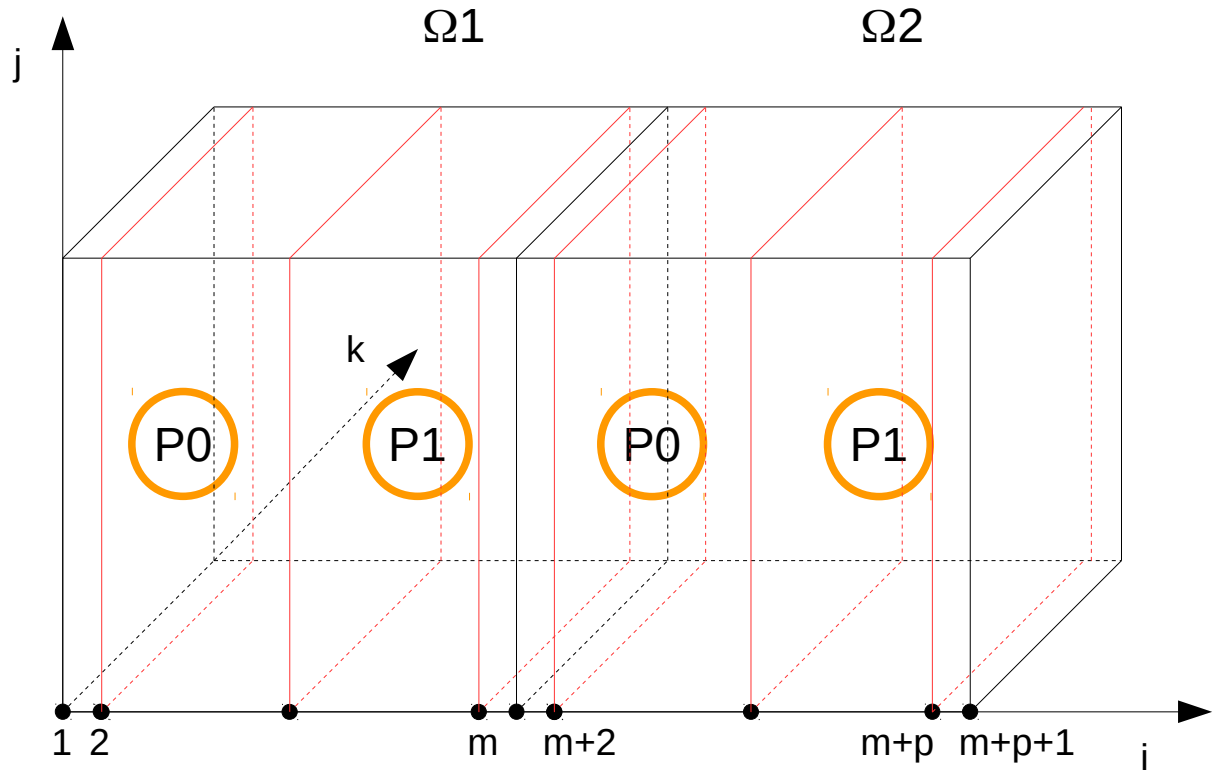
Portage d'un code sous MPI (Message Passing Interface)

Que se passe-t-il si on deux processeurs ?



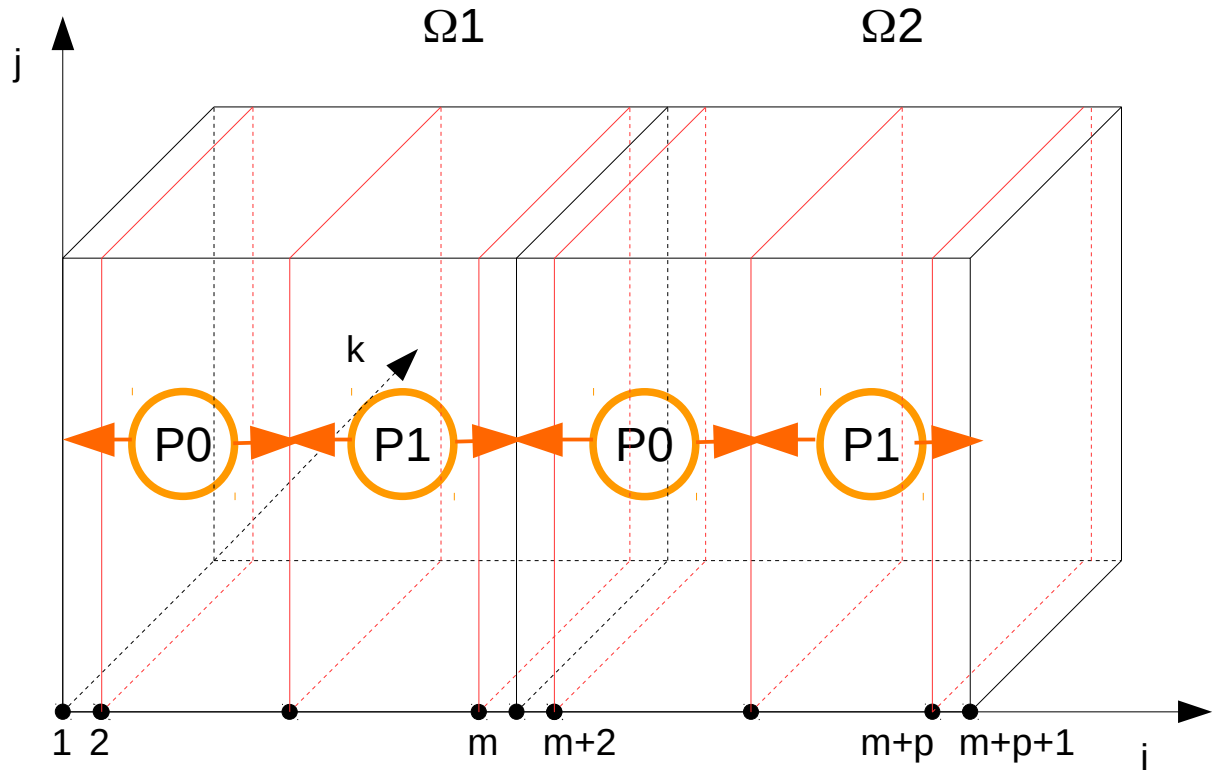
Portage d'un code sous MPI (Message Passing Interface)

On peut partager notre domaine en tranche en chaque i ...



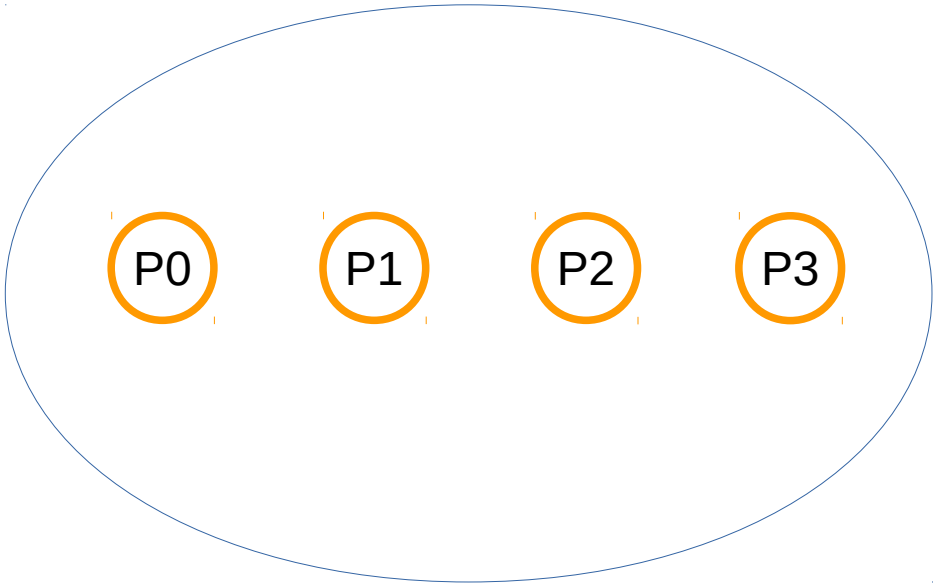
Portage d'un code sous MPI (Message Passing Interface)

On peut partager notre domaine en tranche en chaque i ...

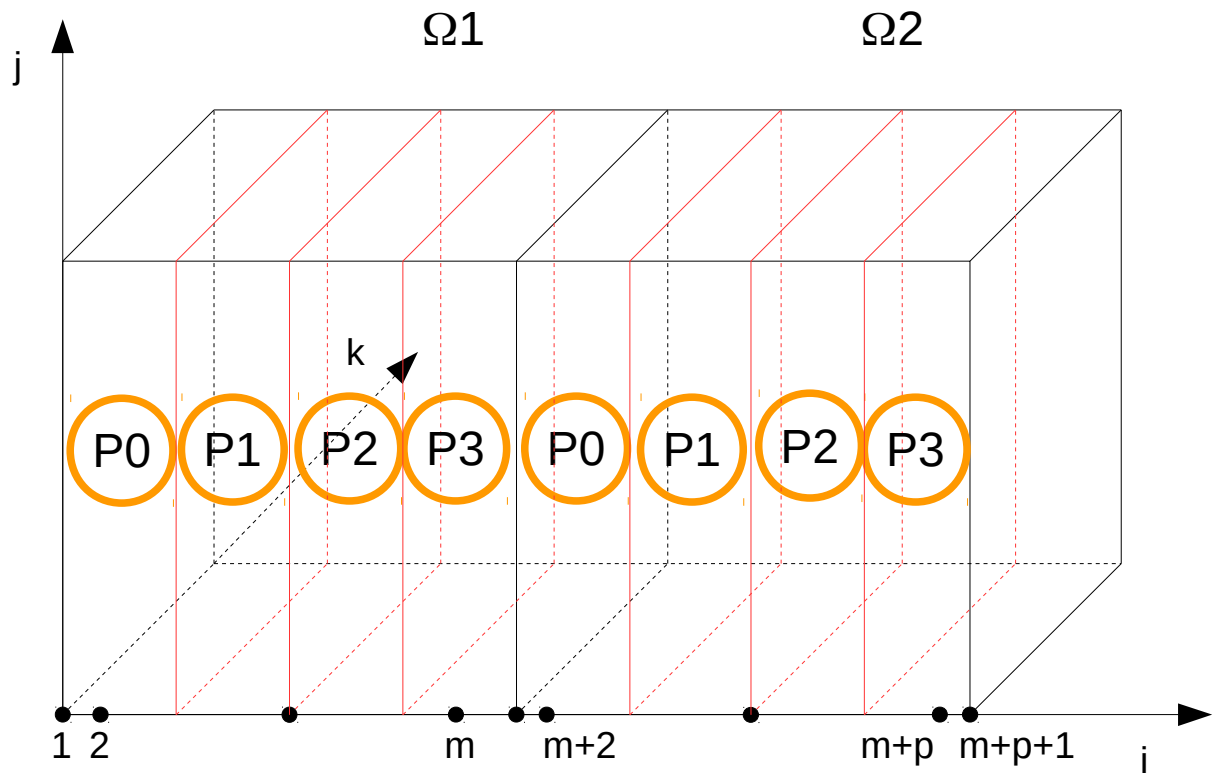


Portage d'un code sous MPI (Message Passing Interface)

Exercice : nous avons 4 processeurs – comment faire évoluer notre découpage ?

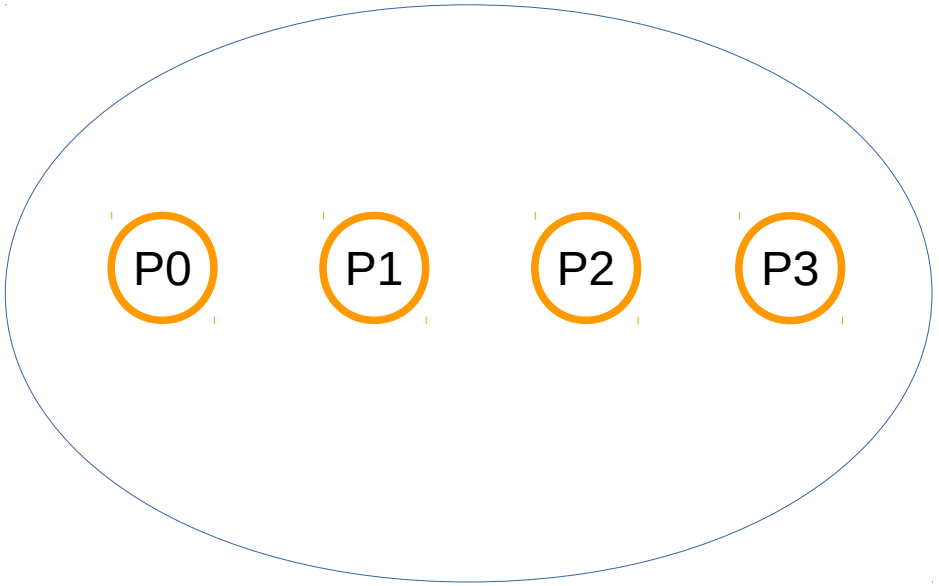


Portage d'un code sous MPI (Message Passing Interface)

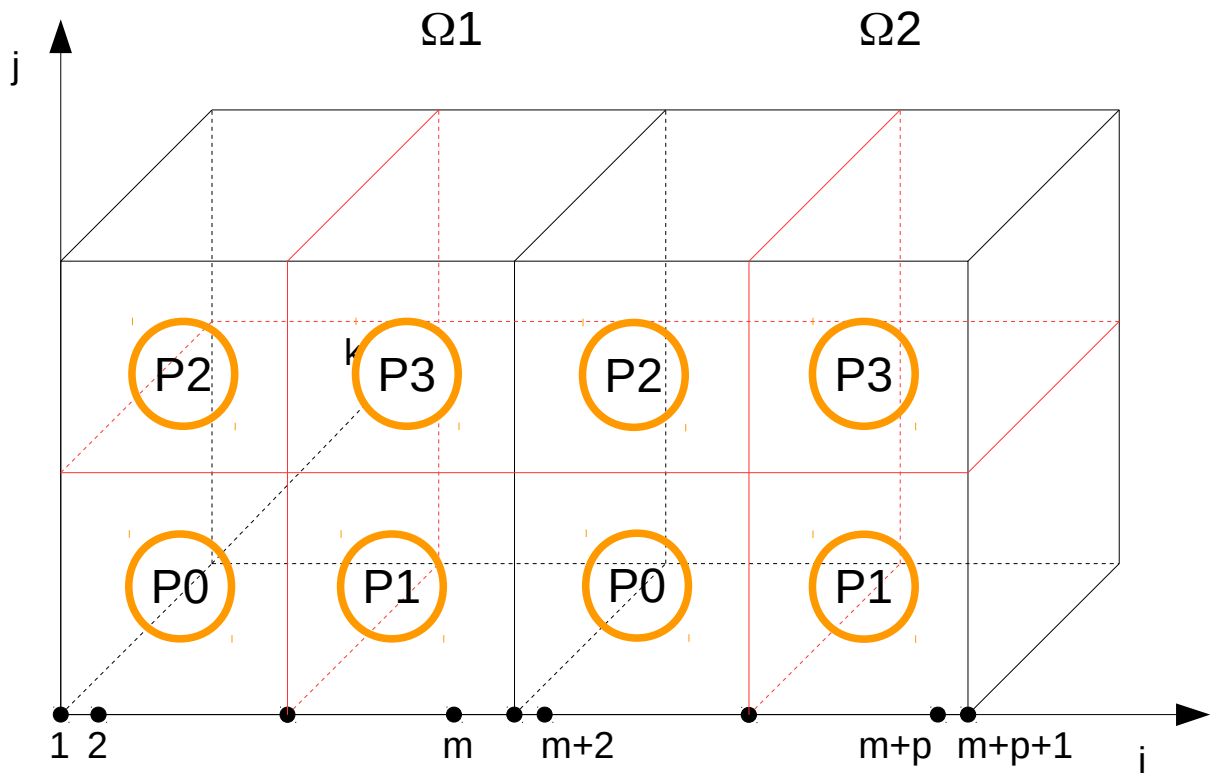


Portage d'un code sous MPI (Message Passing Interface)

Exercice : existe-t-il d'autres façon de découper notre domaine ?

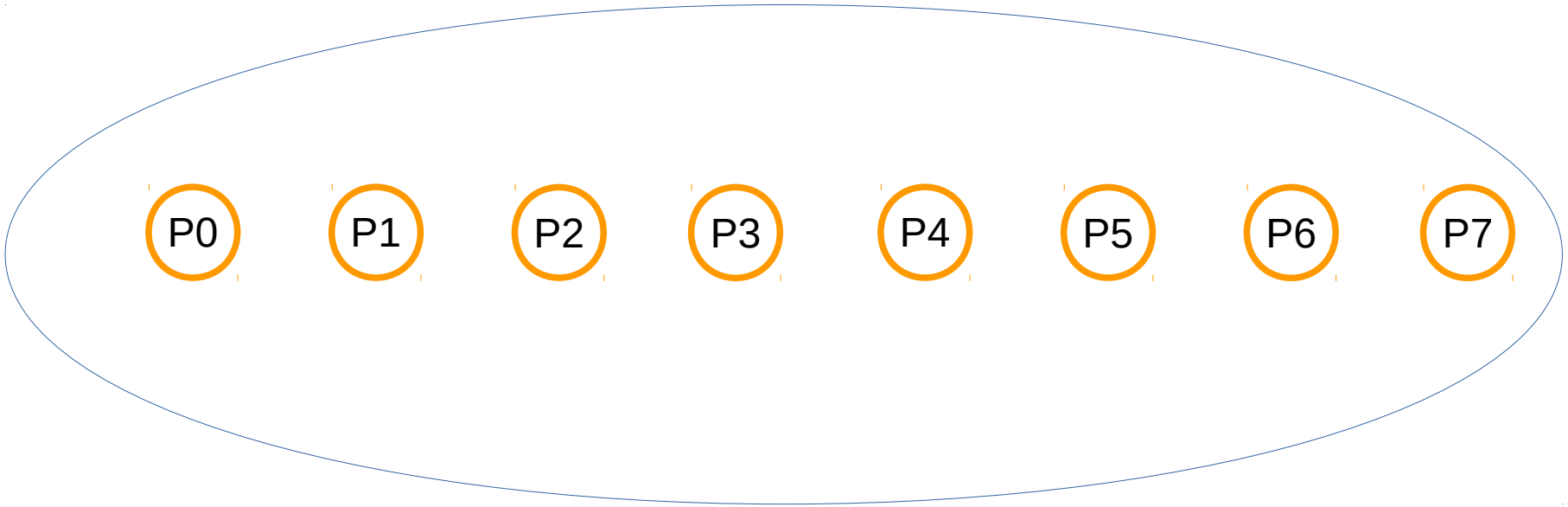


Portage d'un code sous MPI (Message Passing Interface)

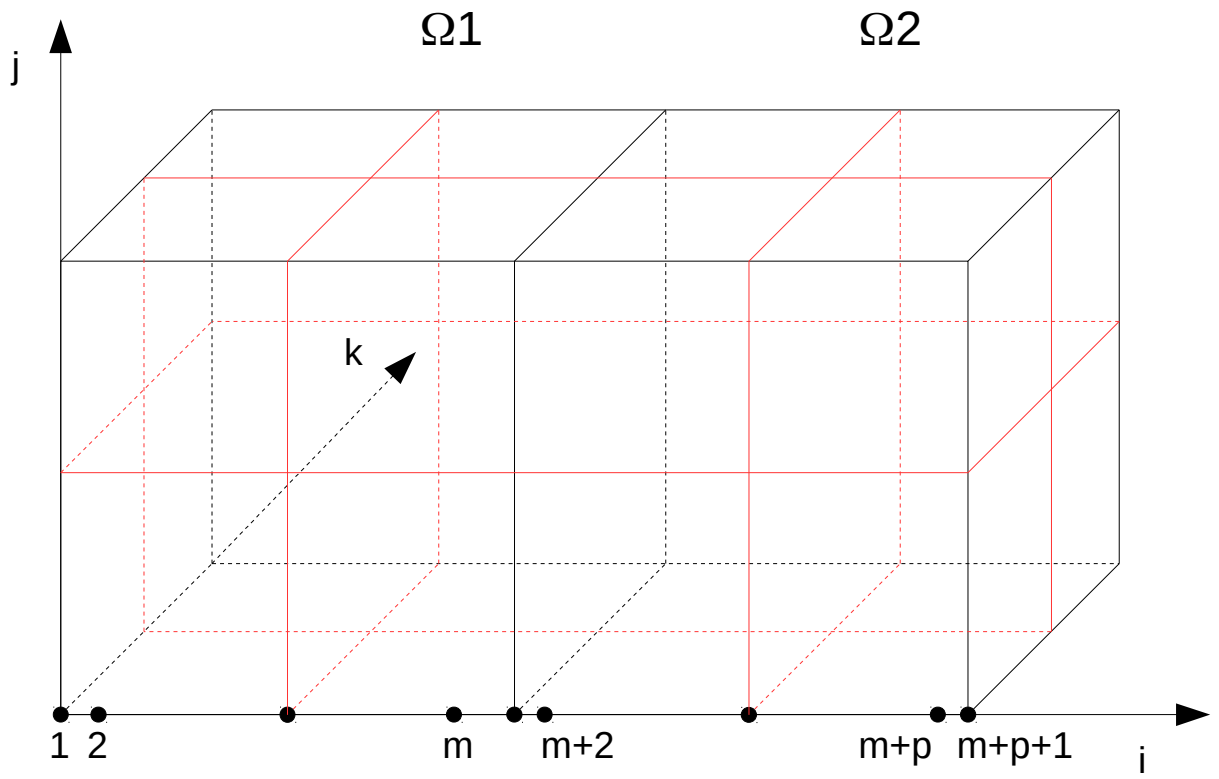


Portage d'un code sous MPI (Message Passing Interface)

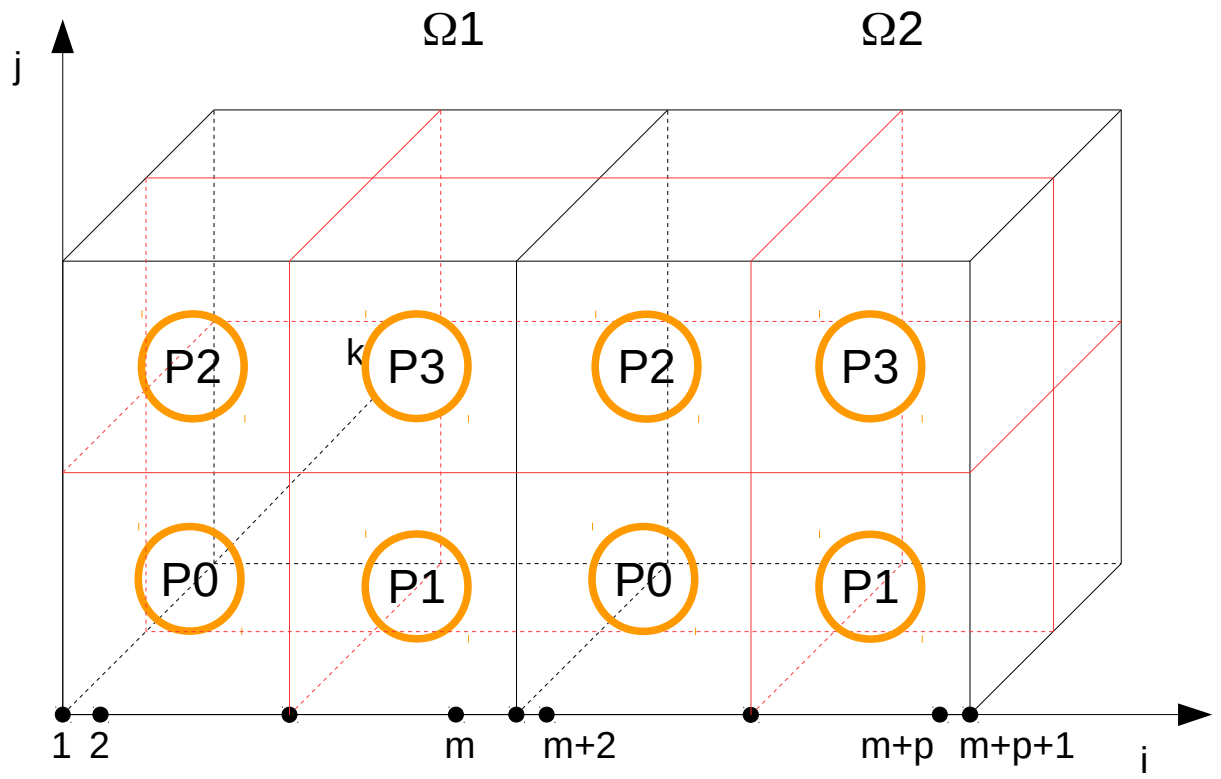
Exercice : 8 processeurs ?



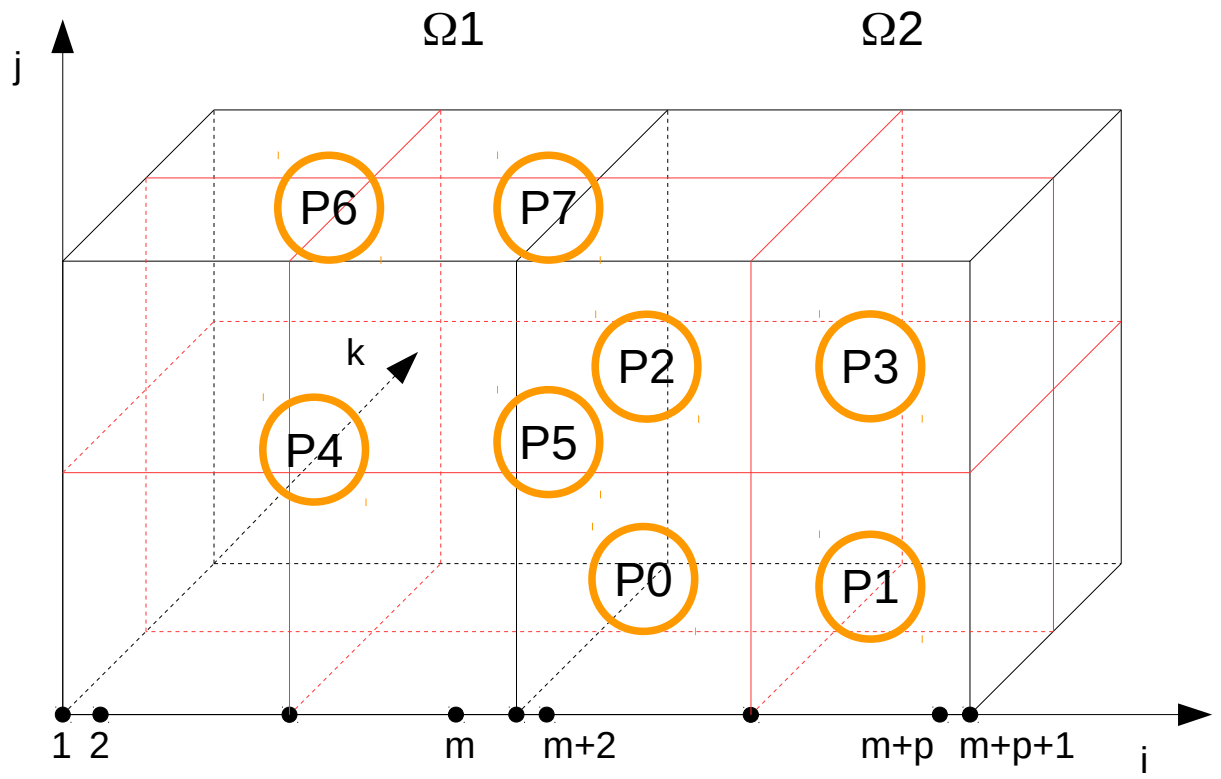
Portage d'un code sous MPI (Message Passing Interface)



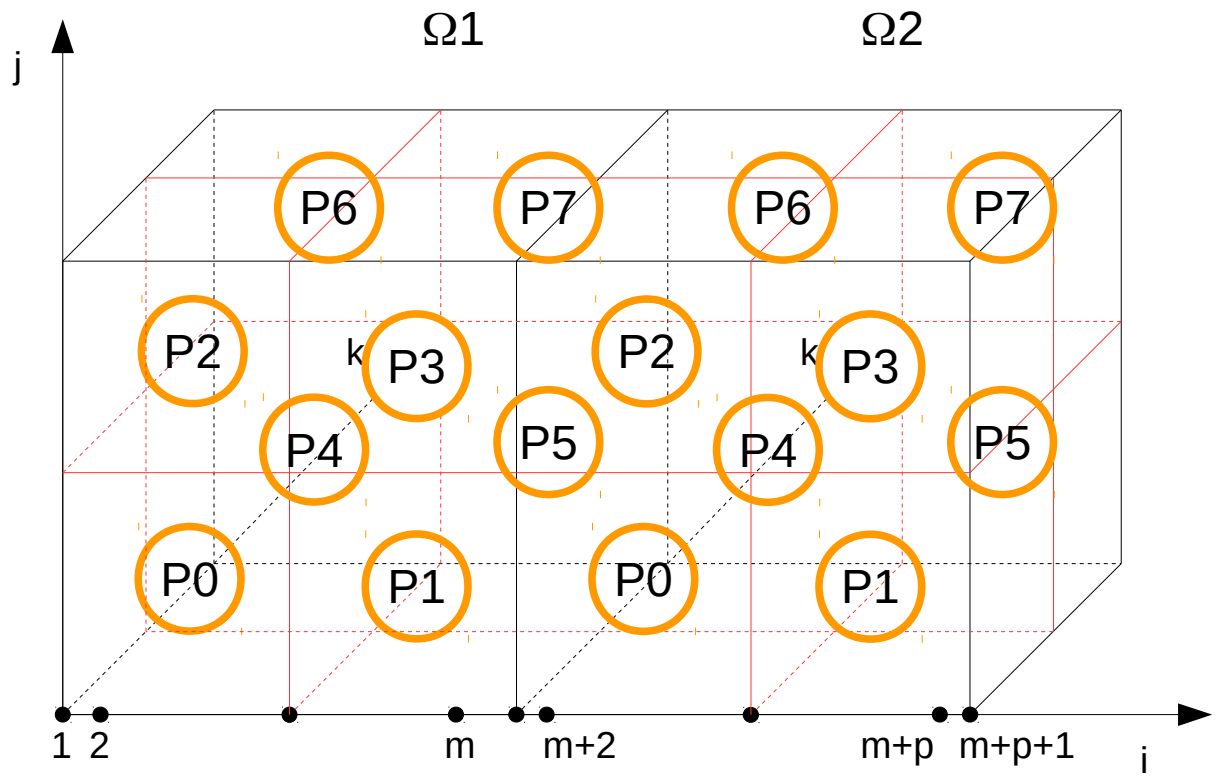
Portage d'un code sous MPI (Message Passing Interface)



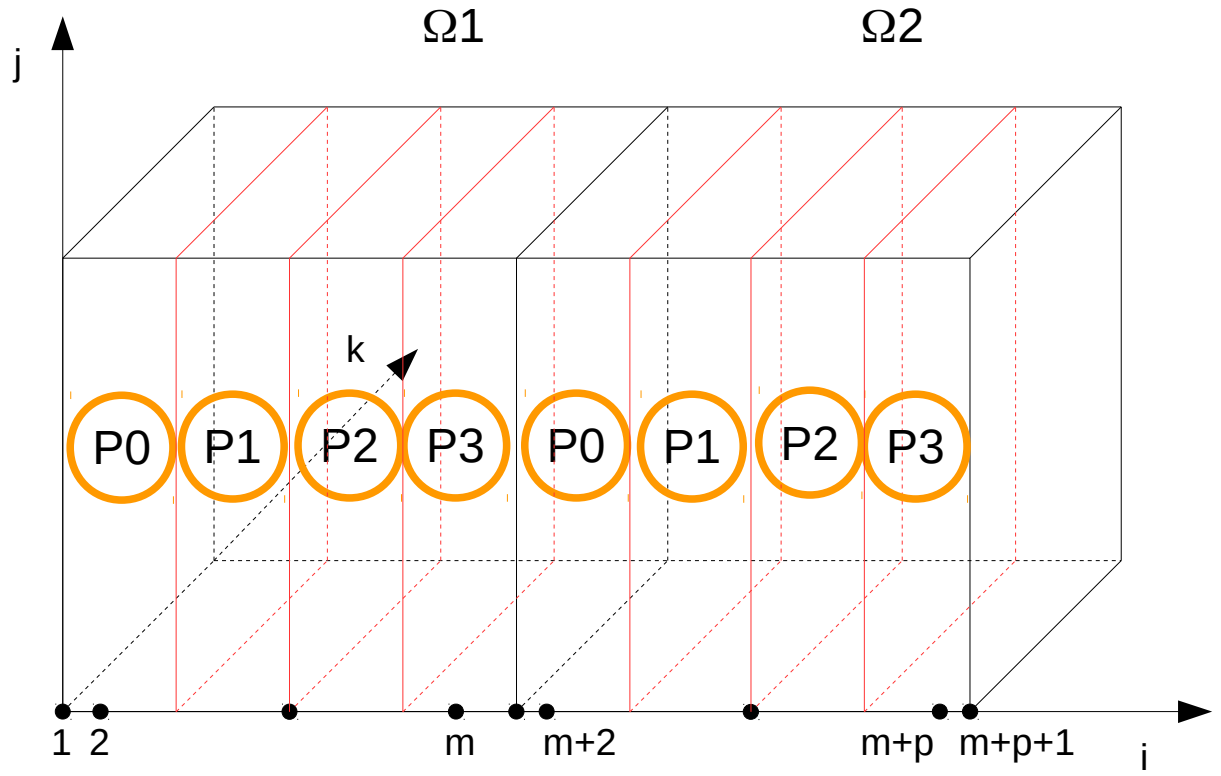
Portage d'un code sous MPI (Message Passing Interface)



Portage d'un code sous MPI (Message Passing Interface)



Portage d'un code sous MPI (Message Passing Interface)

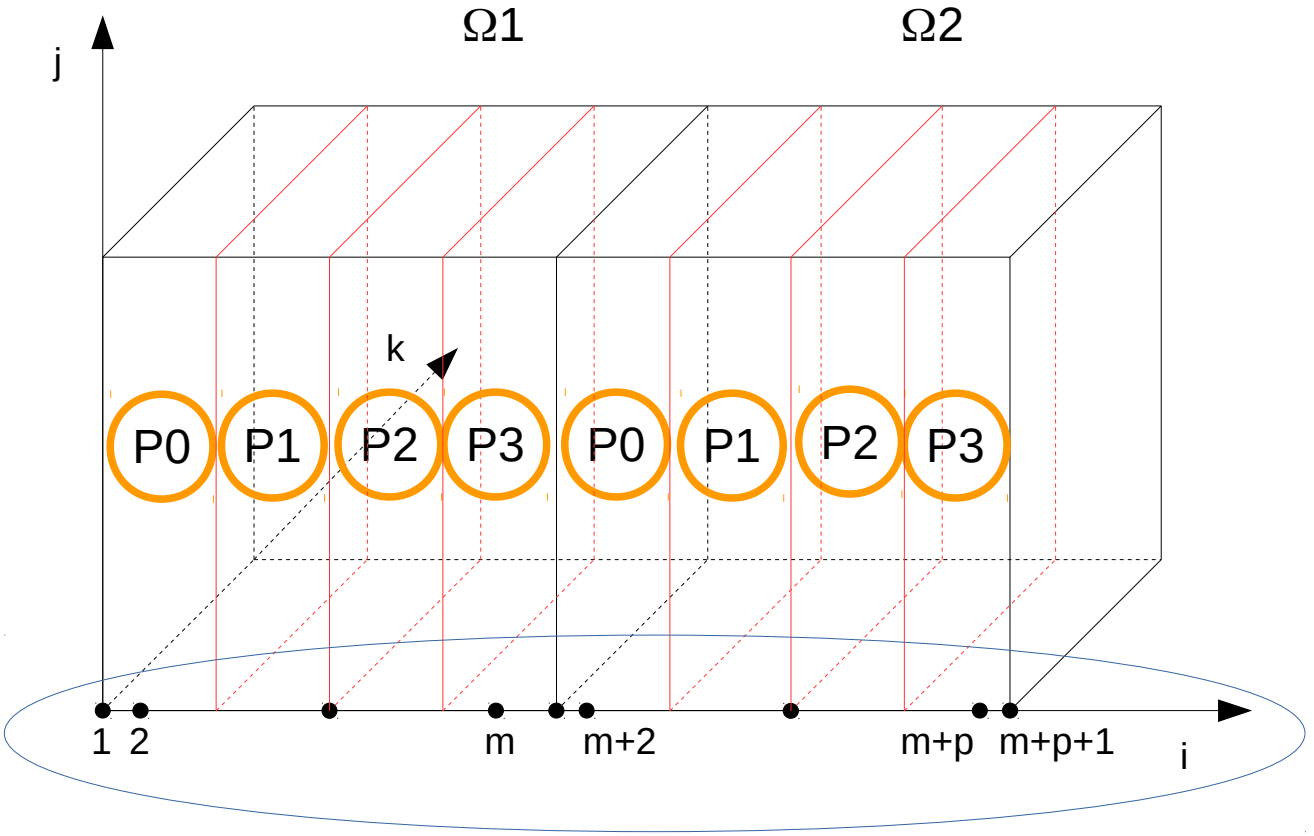


Portage d'un code sous MPI (Message Passing Interface)

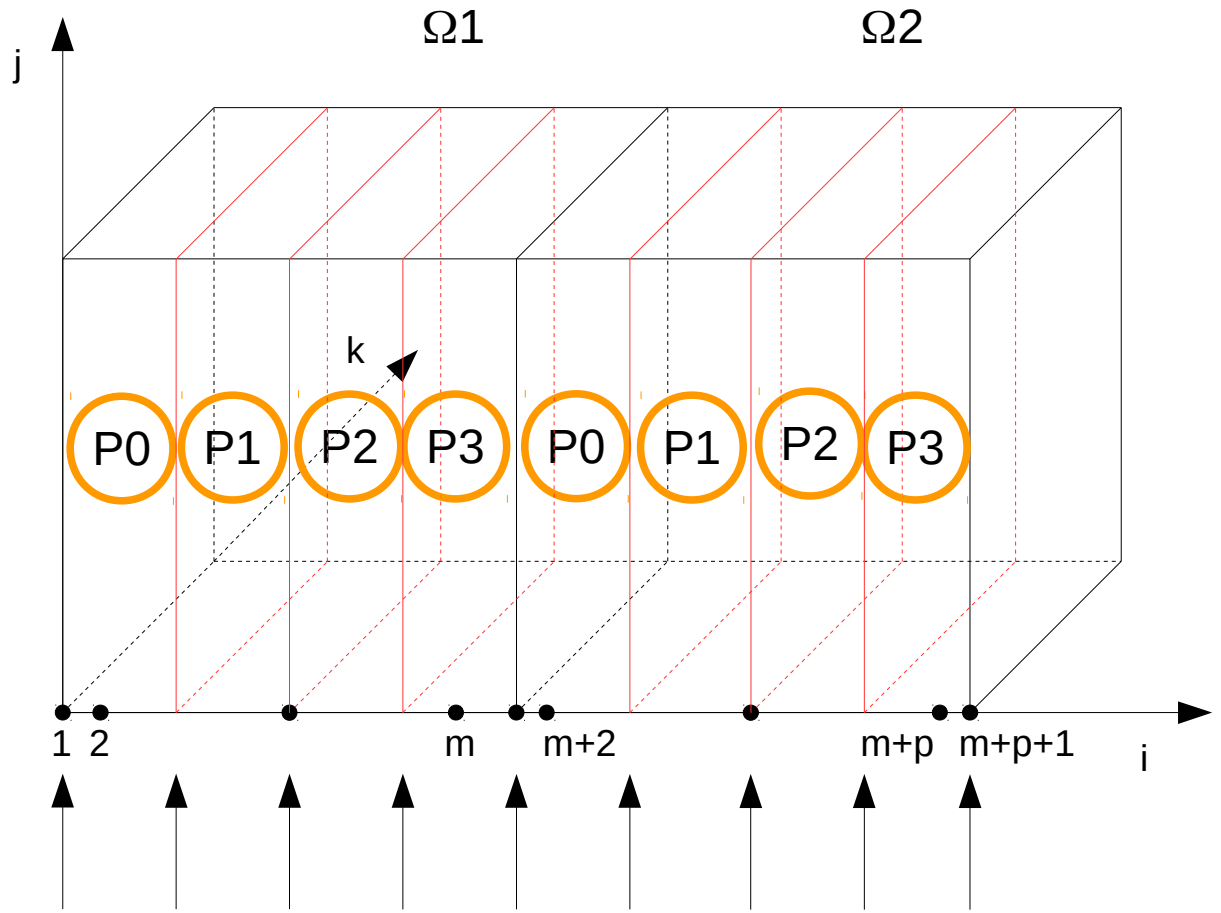
- Comment gérer les sous-domaines (blocs) ?
- Quels sont les indices qui vont être à gérer ?

Portage d'un code sous MPI (Message Passing Interface)

- Quels sont les indices qui vont être à gérer : i ?

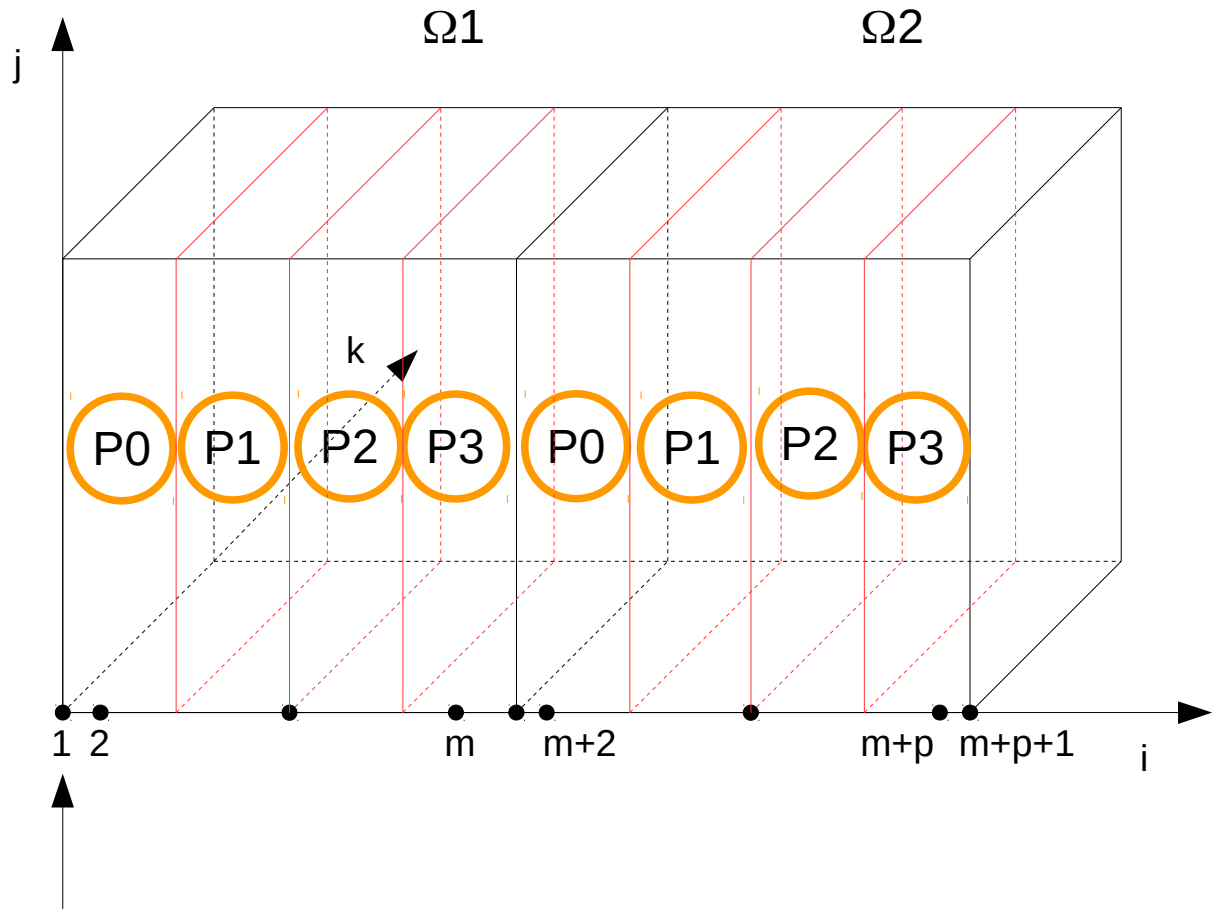


Portage d'un code sous MPI (Message Passing Interface)

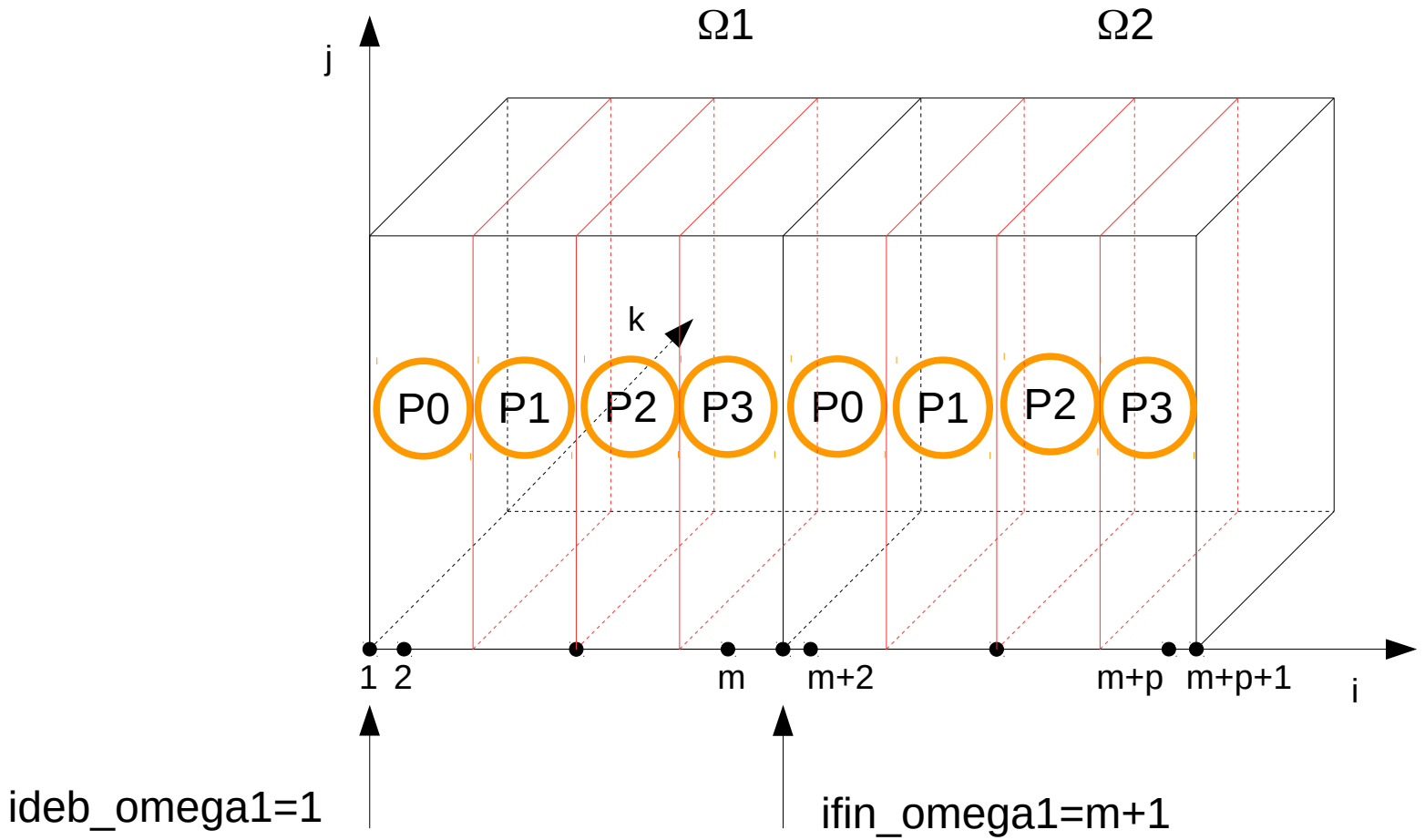


On va devoir trouver les indices de début et de fin de chaque sous-domaine

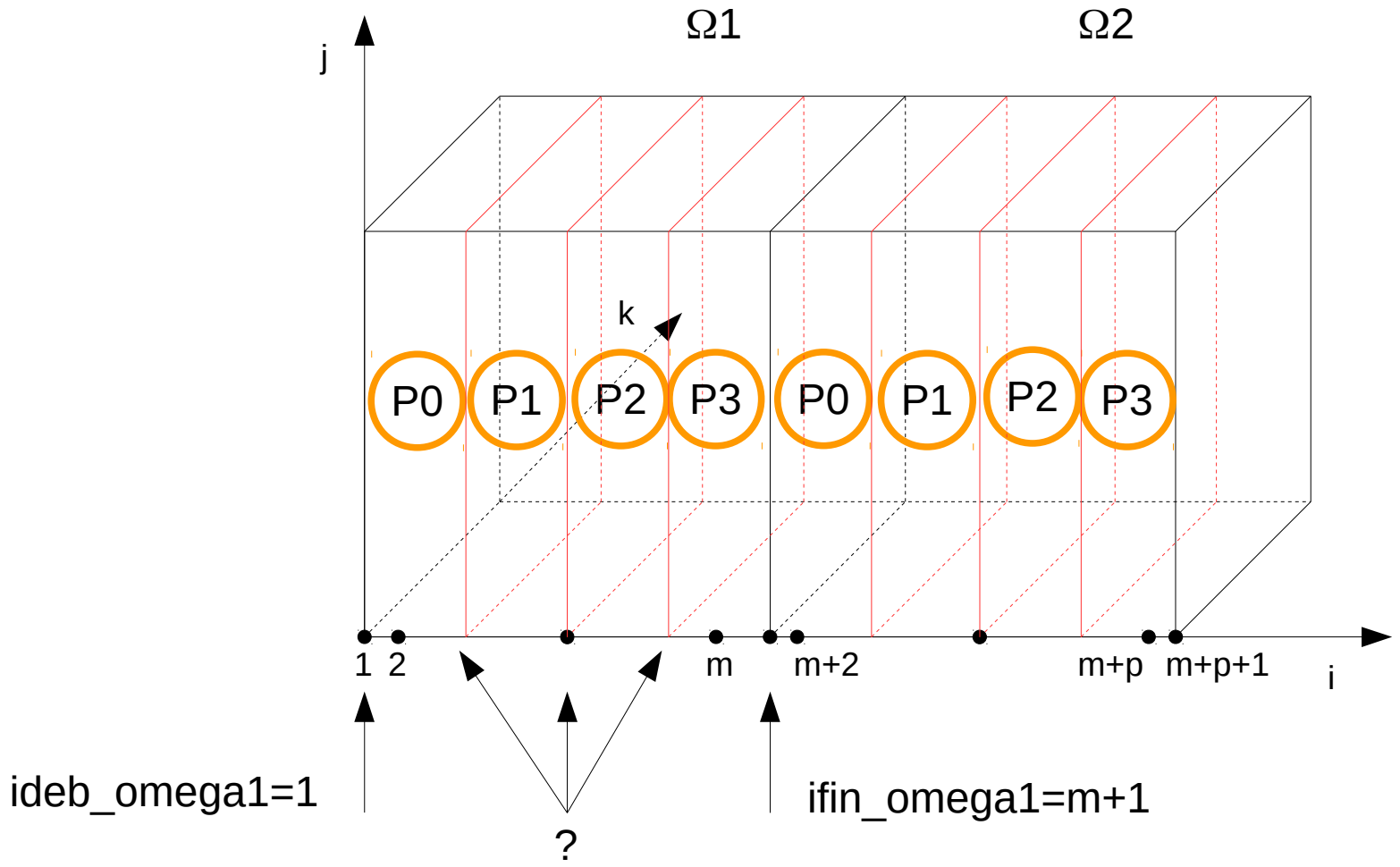
Portage d'un code sous MPI (Message Passing Interface)



Portage d'un code sous MPI (Message Passing Interface)



Portage d'un code sous MPI (Message Passing Interface)



Portage d'un code sous MPI (Message Passing Interface)

On remarque que chaque sous-domaine dans Omega 1 a comme indice

- Plan 0 : 1 ……(m+1)/nbproc
- Plan 1 : (m+1)/nbproc+1 ……..2 * (m+1)/nbproc
- Plan 2 : 2 * (m+1)/nbproc+1 ……3 * (m+1)/nbproc
- Plan 3 : 3 * (m+1)/nbproc+1 ……4 * (m+1)/nbproc

||

m+1

Portage d'un code sous MPI (Message Passing Interface)

On remarque que chaque sous-domaine dans Omega 1 a comme indice

1

||

- Plan 0 : 0 * (m+1)/nbproc+1 …….(0+1)*(m+1)/nbproc
- Plan 1 : 1 * (m+1)/nbproc+1 …… (1+1)*(m+1)/nbproc
- Plan 2 : 2 * (m+1)/nbproc+1 …….(2+1)*(m+1)/nbproc
- Plan 3 : 3 * (m+1)/nbproc+1 …….(3+1)*(m+1)/nbproc

||

m+1

Portage d'un code sous MPI (Message Passing Interface)

On remarque que chaque sous-domaine dans Omega 1 a comme indice

1

||

- Plan 0 : rang * (m+1)/nbproc+1 …… (rang+1)*(m+1)/nbproc
- Plan 1 : rang * (m+1)/nbproc+1 …… (rang+1)*(m+1)/nbproc
- Plan 2 : rang * (m+1)/nbproc+1 …….(rang+1)*(m+1)/nbproc
- Plan 3 : rang * (m+1)/nbproc+1 …….(rang+1)*(m+1)/nbproc

||

m+1

Portage d'un code sous MPI (Message Passing Interface)

On a donc les débuts et fins de chaque sous-domaine d'Omega 1:

$i\_debut\_omega_1 = rang * (m+1)/nbproc+1$

$i\_fin\_omega_1 = (rang+1)*(m+1)/nbproc$

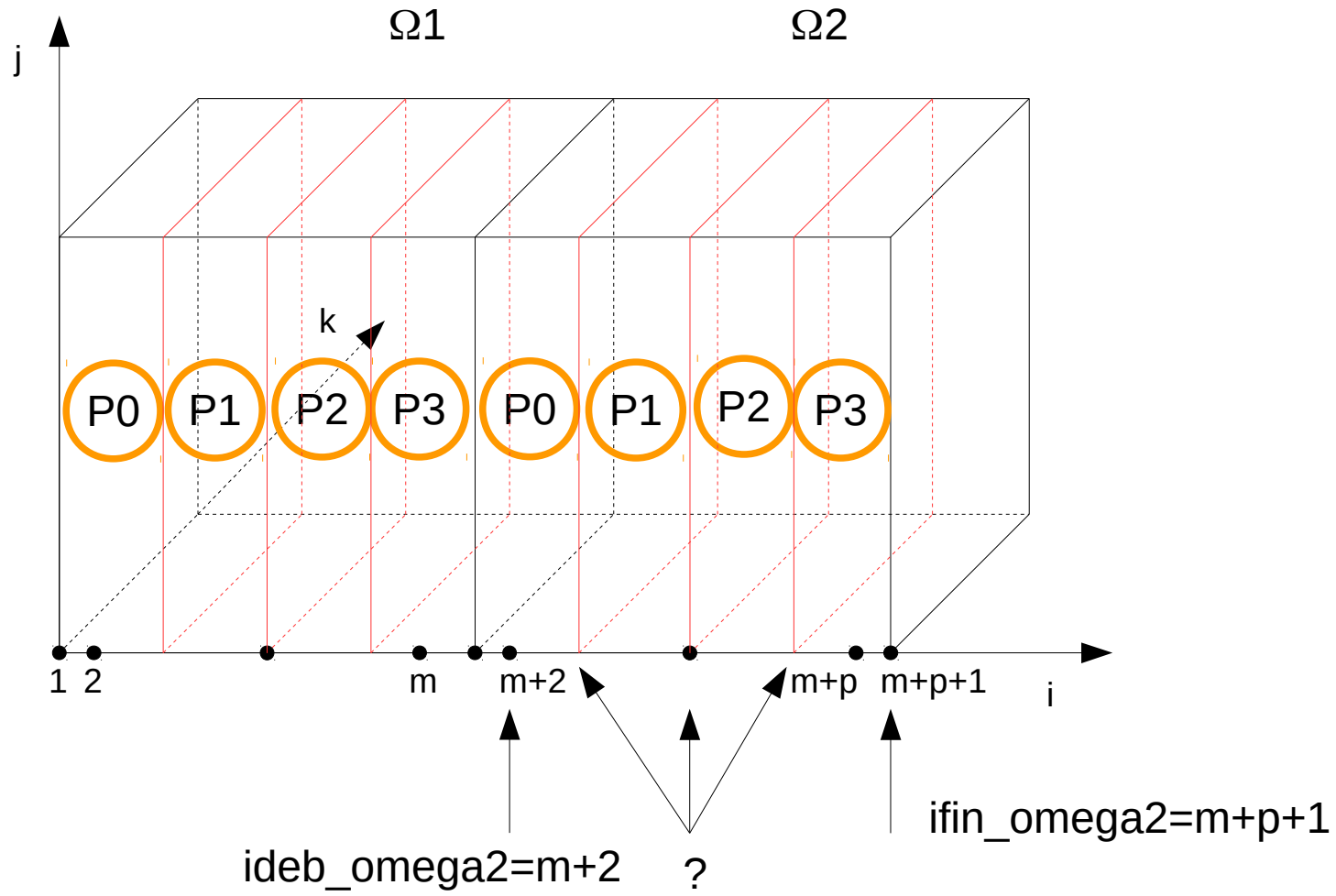
Mais

$i\_fin\_omega_1 = (rang * (m+1)/nbproc) + (m+1)/nbproc$

$i\_fin\_omega_1 = (rang * (m+1)/nbproc+1-1) + (m+1)/nbproc$

$i\_fin\_omega_1 = i\_debut\_omega_1 - 1 + (m+1)/nbproc$

Portage d'un code sous MPI (Message Passing Interface)



Portage d'un code sous MPI (Message Passing Interface)

On a donc les débuts et fins de chaque sous-domaine d'Omega 2:

$i\_debut\_omega_2 = rang * (p+1)/nbproc+(m+2)$

$i\_fin\_omega_2 = i\_debut\_omega_2 - 1 + (p+1)/nbproc$

Portage d'un code sous MPI (Message Passing Interface)

(voir fichier ExerciceMPI_2.f90)

```
C=============================================================
C     Definition des indices des plans pour Omega1 et Omega2
C=============================================================
      ideb_omega1=1+((m+1)/nbproc)*rang
      ifin_omega1=(m+1)/nbproc+ideb_omega1-1

      ideb_omega2=(m+2)+((p+1)/nbproc)*rang
      ifin_omega2=(p+1)/nbproc+ideb_omega2-1
```

Portage d'un code sous MPI (Message Passing Interface)

Comme le programme va s'exécuter en parallèle, on va devoir considérer l'algorithme s'exécutant sur chaque sous-domaine et transmettre les indices de chaque sous-domaine dans les sous-programmes.

Ainsi (voir fichier ExerciceMPI_3.f90),

```
! Dans omega 1 - Calcul du second membre dans OMEGA 1

CALL SCDMB1(f,u,A1,nn,n,m,ideb_omega1,ifin_omega1)

! Boucle de calcul dans OMEGA 1 (points intérieurs)

CALL PTSINT1(f,u,diag1,codiag1,nn,n,m,normmax1,iter1,epsi,ideb_omega1,ifin_omega1)


! Dans omega 2 - Calcul du second membre dans OMEGA 2

CALL SCDMB2(f,u,A2,nn,n,m,p,ideb_omega2,ifin_omega2)

! Boucle de calcul dans OMEGA 2 (points intérieurs)

CALL PTSINT2(f,u,diag2,codiag2,nn,n,m,p,normmax2,iter2,epsi,ideb_omega2,ifin_omega2)
```

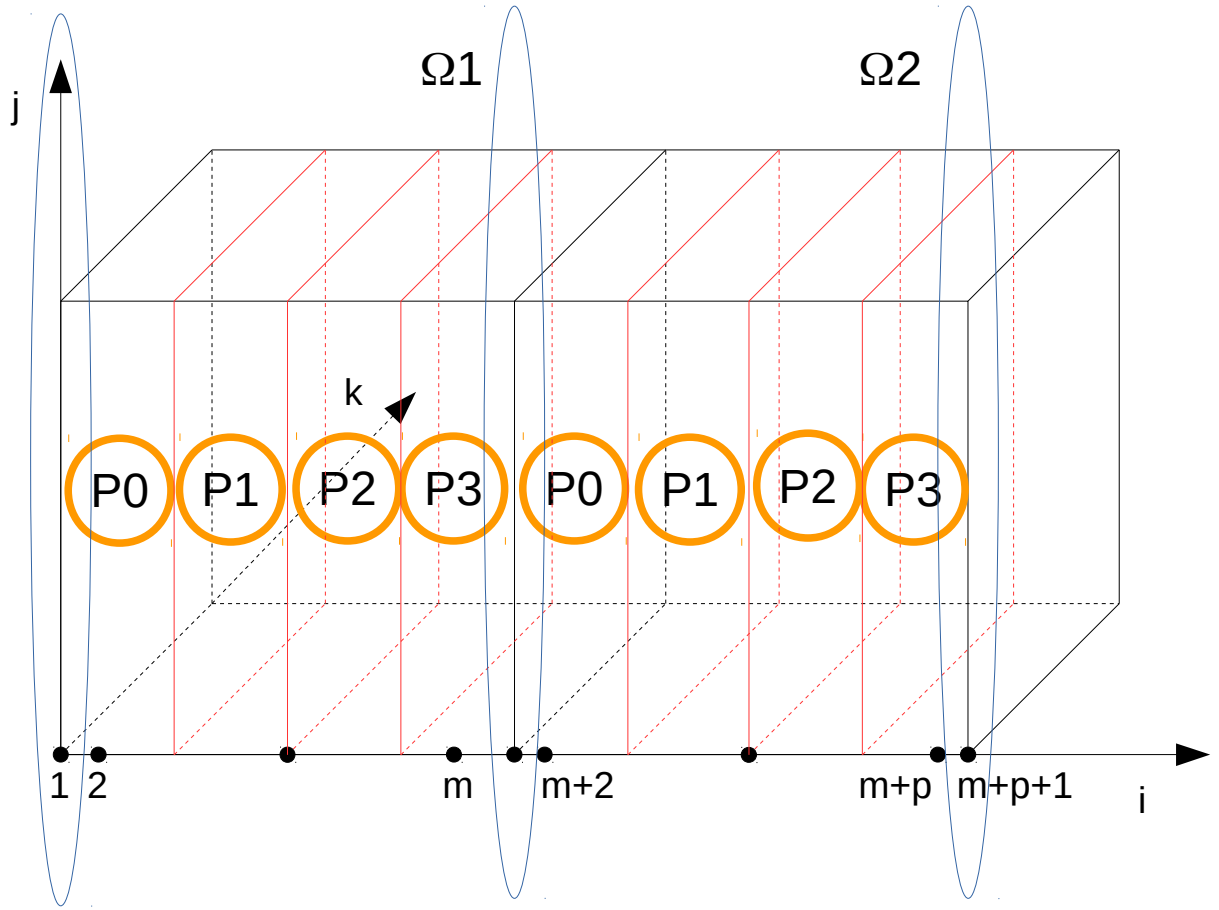
Portage d'un code sous MPI (Message Passing Interface)

Modifions aussi :

```
SUBROUTINE SCDMB1(f,u,A1,nn,n,m,ideb,ifin)
integer nn,n,m
integer ideb,ifin
…….
do k=1,nn+2
     do j=1,n+2
          !do i=2,m
          do i=ideb,ifin
               f(i,j,k)=A1*u(i,j,k)
          enddo
     enddo
enddo
return
end
```

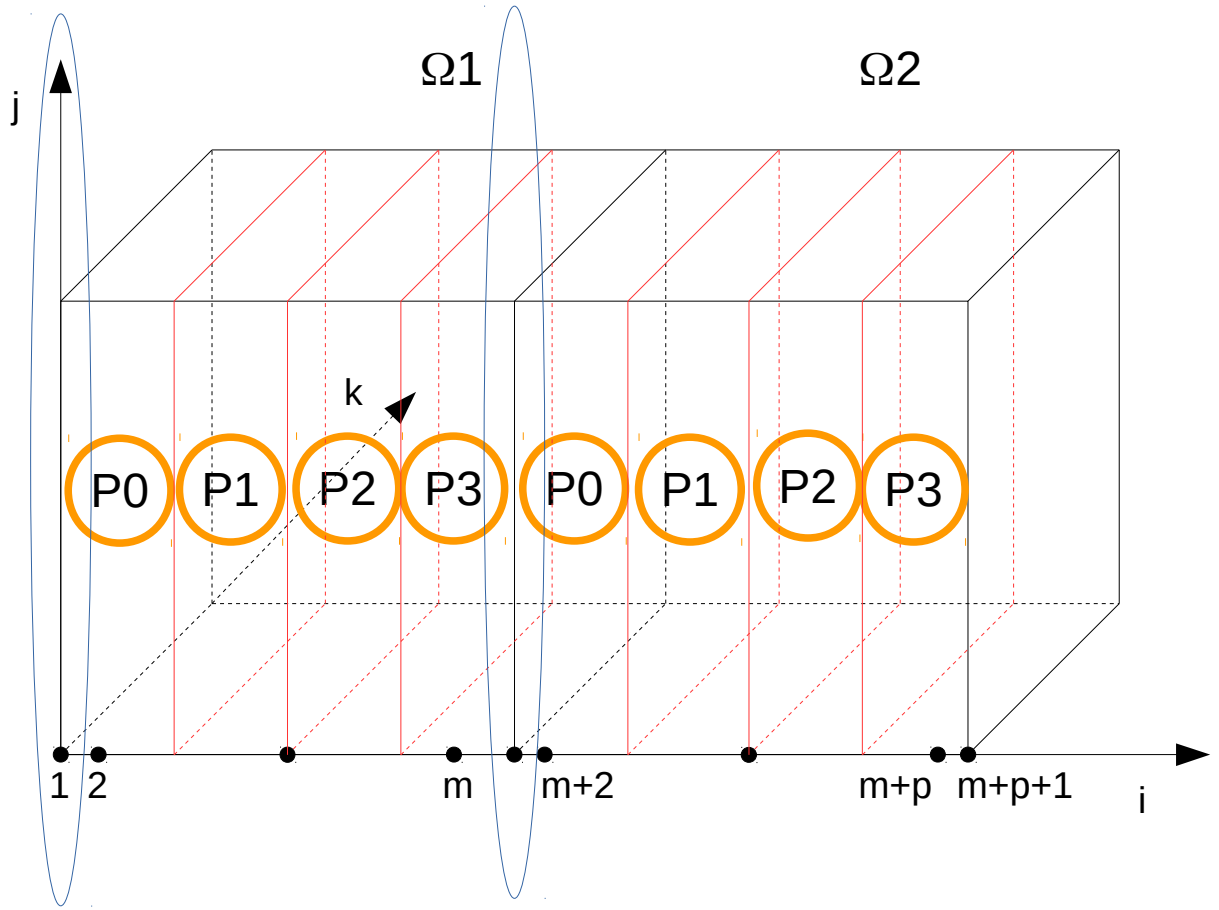
Portage d'un code sous MPI (Message Passing Interface)

Attention au cas 1, cas m+1 et cas m+p+1



Portage d'un code sous MPI (Message Passing Interface)

Attention au cas 1, cas m+1 pour Omega 1



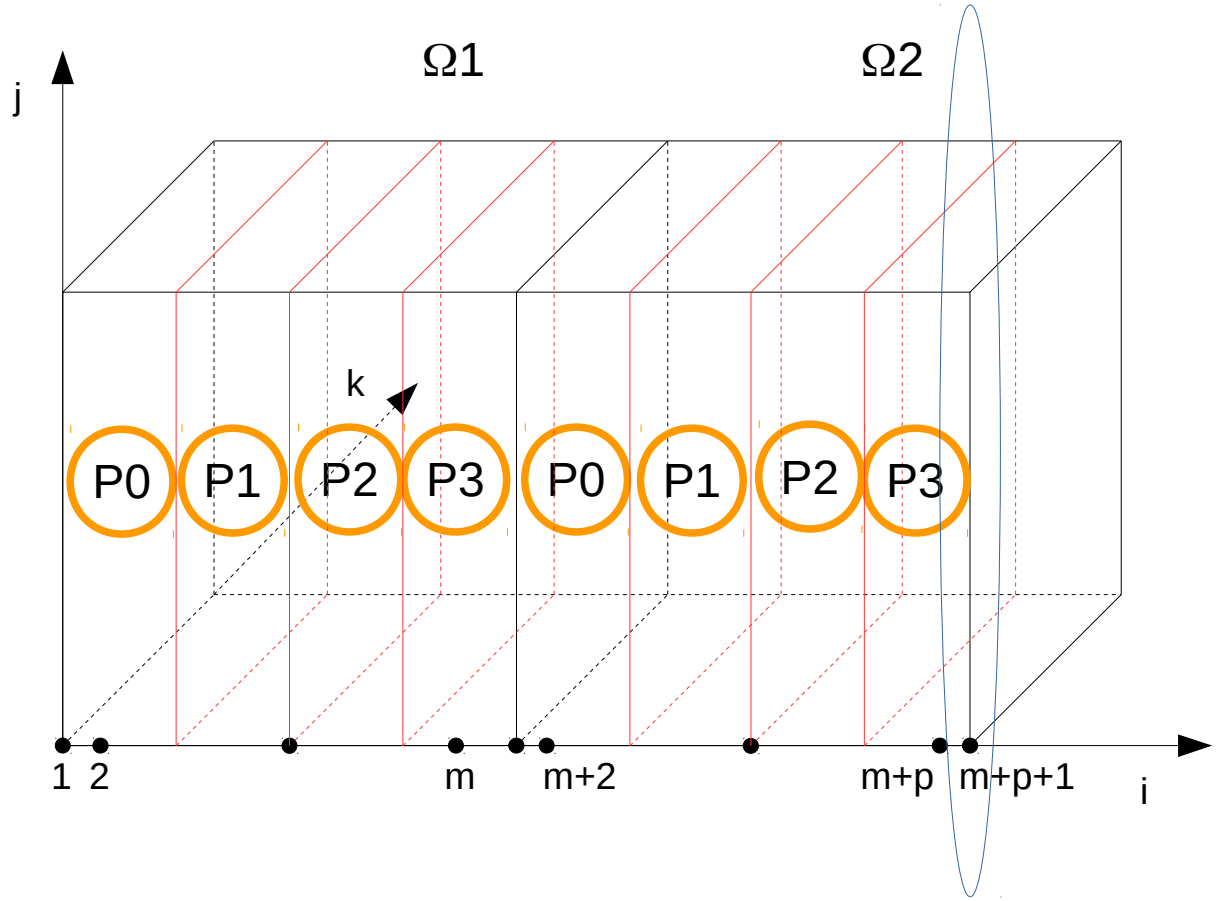
Portage d'un code sous MPI (Message Passing Interface)

Ce qui donne (voir fichier ExerciceMPI_4.f90) :

```
SUBROUTINE SCDMB1(f,u,A1,nn,n,m,ideb,ifin)
…….
do k=1,nn+2
     do j=1,n+2
        !do i=2,m
        do i=ideb,ifin
! on parcourt les i sauf dans le cas i=1 sur le proc 1 et on parcourt les i sauf dans le cas i=m+1 sur le dernier proc
              IF ((.NOT. ((rang == 0) .AND. (i == 1))).AND.(.NOT. ((rang == nbproc-1) .AND. (i == m+1)))) THEN
                  f(i,j,k)=A1*u(i,j,k)
                ENDIF
        enddo
    enddo
enddo
```

Portage d'un code sous MPI (Message Passing Interface)

Attention au cas m+p+1 pour Omega 2



Portage d'un code sous MPI (Message Passing Interface)

Ce qui donne (voir fichier ExerciceMPI_4.f90) :

```
SUBROUTINE SCDMB2(f,u,A2,nn,n,m,p,ideb,ifin)
…….
do k=1,nn+2
   do j=1,n+2
      !do i=m+2,m+p
       do i=ideb,ifin
       ! les i sauf dans le cas i=m+p+1 sur le dernier proc
              IF (.NOT. ((rang == nbproc-1) .AND. (i == m+p+1))) THEN
                f(i,j,k)=A2*u(i,j,k)
             ENDIF
      enddo
   enddo
enddo
```

Portage d'un code sous MPI (Message Passing Interface)

On applique le même processus pour les points intérieurs (voir fichier ExerciceMPI_5.f90)

```
SUBROUTINE PTSINT1(f,u,diag1,codiag1,nn,n,m,normmax1,iter1,epsi,ideb,ifin)
    do k=2,nn+1
        do j=2,n+1
!do i=2,m
            DO i=ideb,ifin
! on parcourt les i sauf dans le cas i=1 sur le proc et on parcourt les i sauf dans le cas i=m+1 sur le dernier proc
                IF ((.NOT. ((rang == 0) .AND. (i == 1))).AND.(.NOT. ((rang == nbproc-1) .AND. (i == m+1))))THEN
                    SS= ….
                    !%norme
                    RR=dabs(SS-u(i,j,k))
                    if (RR .gt. normmax1)then
                        normmax1=RR
                    endif
                    u(i,j,k)=SS
                ENDIF
```

Portage d'un code sous MPI (Message Passing Interface)

On applique le même processus pour les points intérieurs (voir fichier ExerciceMPI_5.f90)

```
SUBROUTINE PTSINT2(f,u,diag2,codiag2,nn,n,m,p,normmax2,iter2,epsi,ideb,ifin)
    do k=2,nn+1
      do j=2,n+1
       !do i=m+2,m+p
            DO i=ideb,ifin
              ! les i sauf dans le cas i=m+p+1 sur le dernier proc
               IF (.NOT. ((rang==nbproc-1).AND.(i==m+p+1))) THEN
                  SS=…..
                  !%norme
                  RR=dabs(SS-u(i,j,k))
                  if (RR .gt. normmax2) then
                     normmax2=RR
                  endif
                  u(i,j,k)=SS
               ENDIF
```

Portage d'un code sous MPI (Message Passing Interface)

On applique le même processus pour l'interface entre Omega 1 et Omega 2 (voir fichier ExerciceMPI_5.f90)

```
! INTERFACE OMEGA 1 OMEGA 2
SUBROUTINE LINTERFACE(u,epsi,nn,n,m,iter1,normax1,ct11,ct2,ct31prim)
integer :: nn,n,m, iter1
real*8 ,dimension (141,141,141):: u
real*8 epsi, normax1,ct11,ct2,ct31prim, SS, RR
! VERTICAL INTERFACE omega1 omega2     I=m+1
 IF (numproc == nbproc) THEN ! le dernier proc a le i=m+1
        do k=1,nn+2
           do j=1,n+2
              SS=(ct11*u(m,j,k)+ct2)/ct31prim
              ….
         ENDIF
 return
 end
```

Portage d'un code sous MPI (Message Passing Interface)

Mesure du temps d'exécution

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 donne le nombre de secondes qui ont passé entre ces deux endroits.

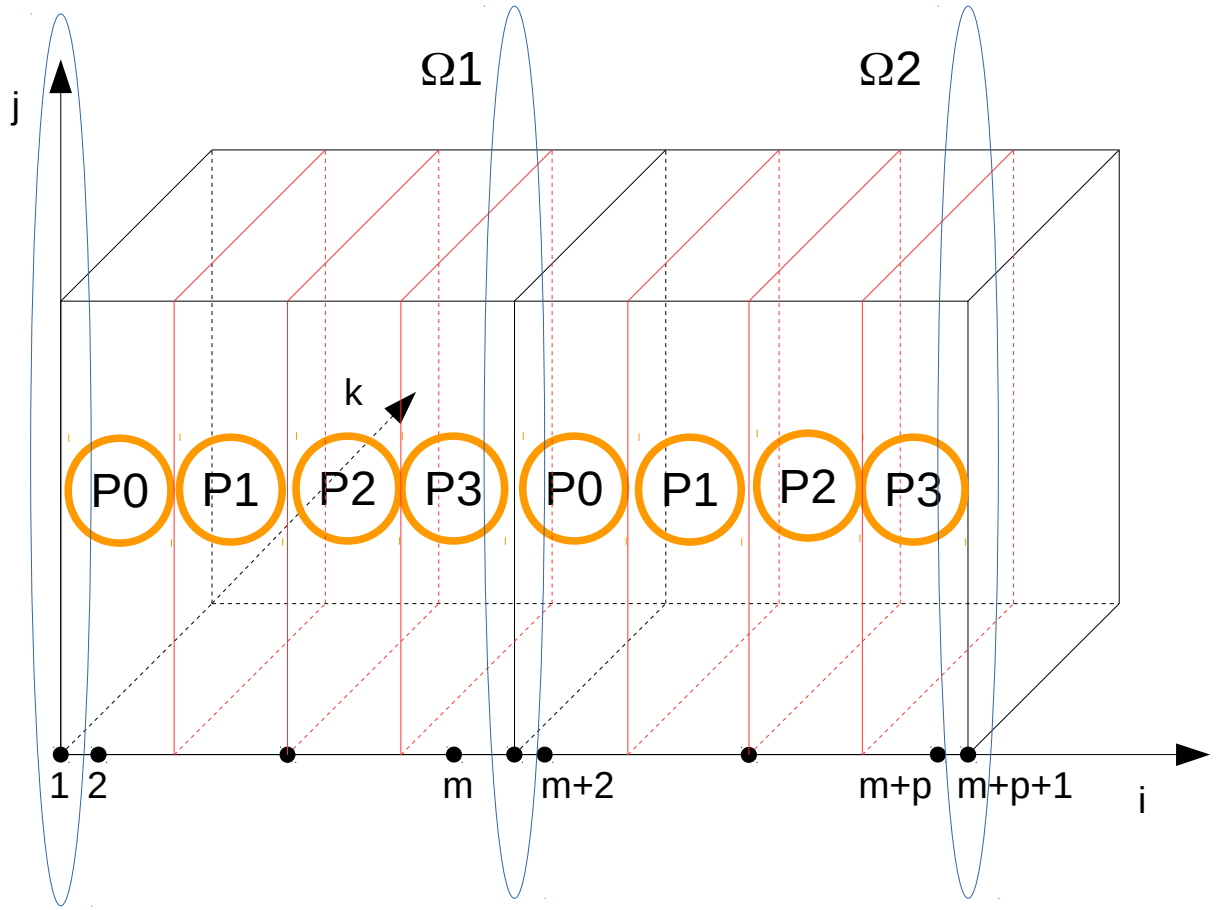
Portage d'un code sous MPI (Message Passing Interface)

On modifie (voir fichier ExerciceMPI_6.f90) :

```
!=====================================
! Appel horloge (mesure du temps d'exécution)
!=====================================
!call cpu_time(start_time)
start_time=MPI_WTIME()
….
!call cpu_time(stop_time)
stop_time=MPI_WTIME()
start=stop_time-start_time
```

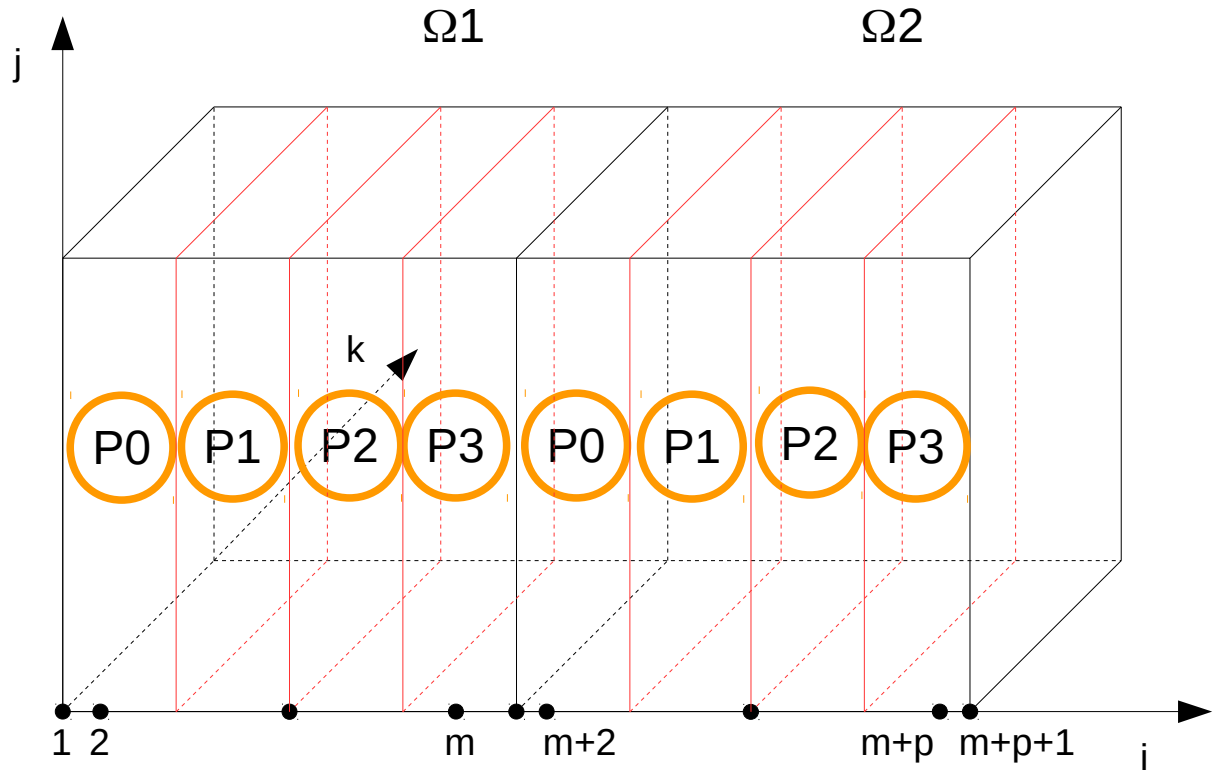
Portage d'un code sous MPI (Message Passing Interface)

A-t-on les bons résultats et est-ce terminé?



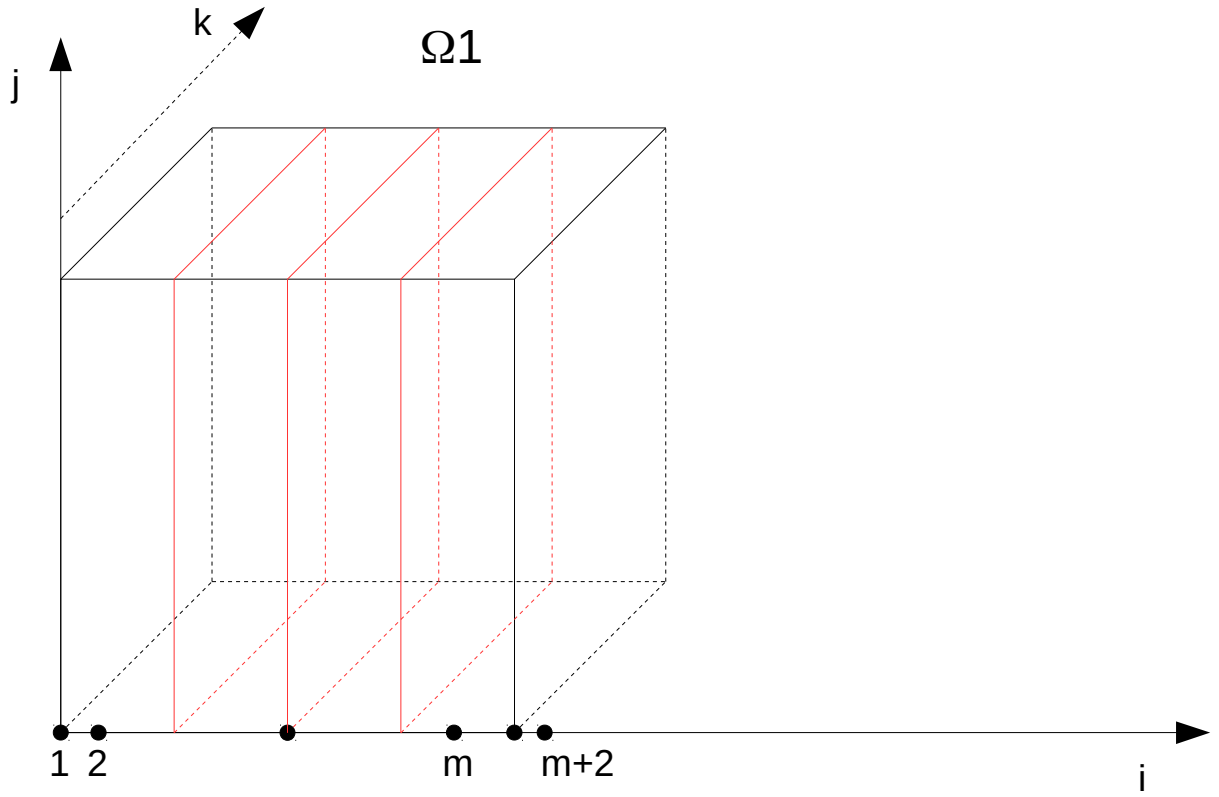
Portage d'un code sous MPI (Message Passing Interface)

A-t-on les bons résultats et est-ce terminé ?



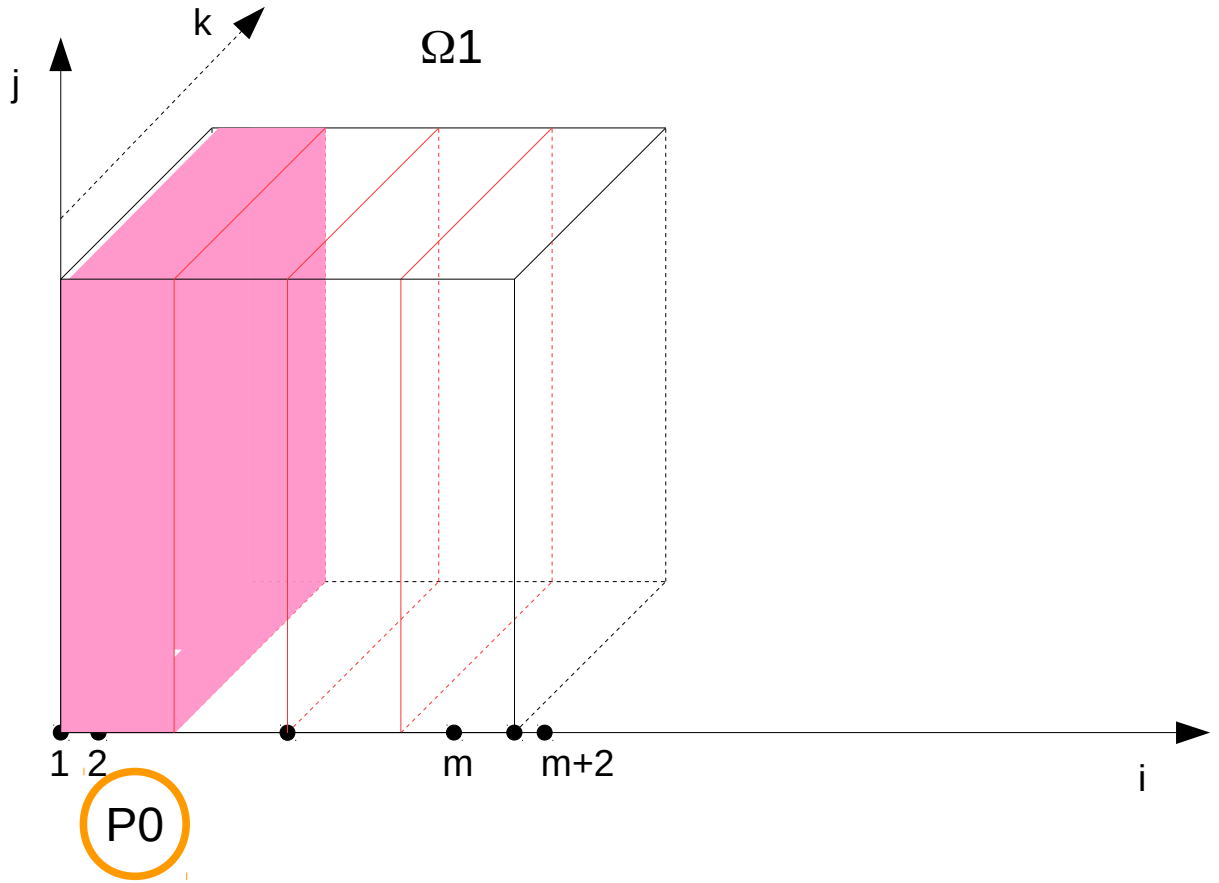
Portage d'un code sous MPI (Message Passing Interface)

A-t-on les bons résultats et est-ce terminé ?



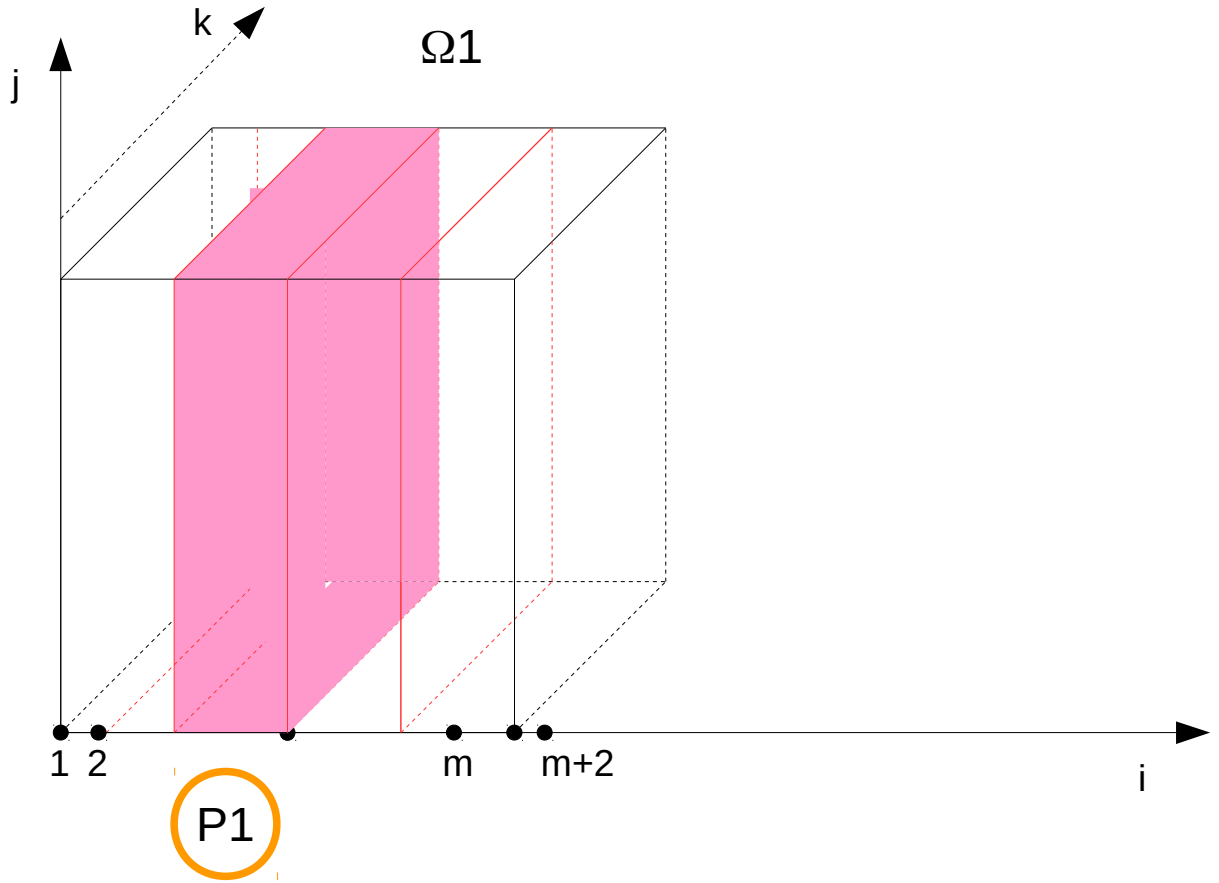
Portage d'un code sous MPI (Message Passing Interface)

A-t-on les bons résultats et est-ce terminé ?



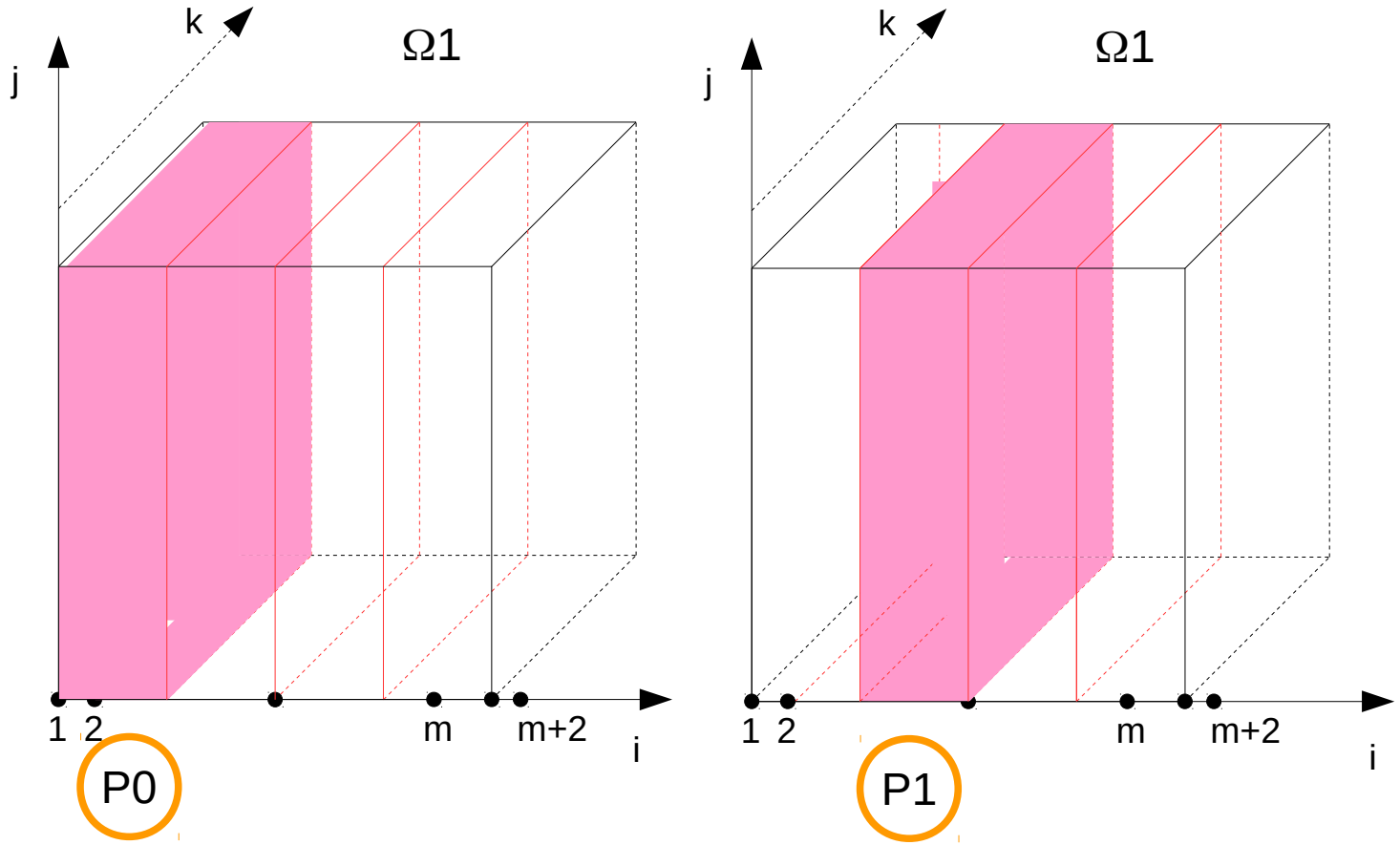
Portage d'un code sous MPI (Message Passing Interface)

A-t-on les bons résultats et est-ce terminé ?



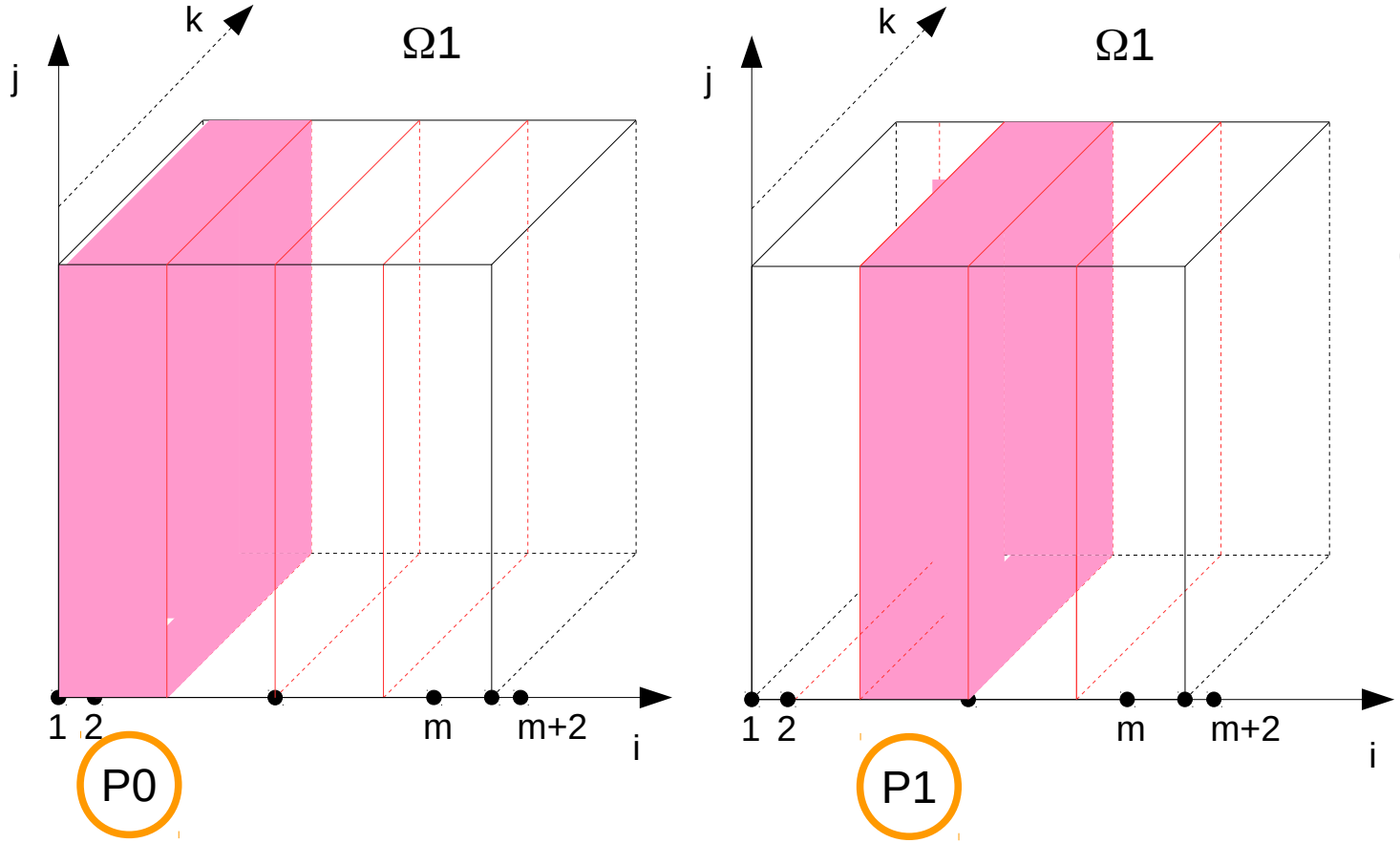
Portage d'un code sous MPI (Message Passing Interface)

b) **Qu'on réitère ces calculs** tant que la normmax1 est > à epsilon



Portage d'un code sous MPI (Message Passing Interface)

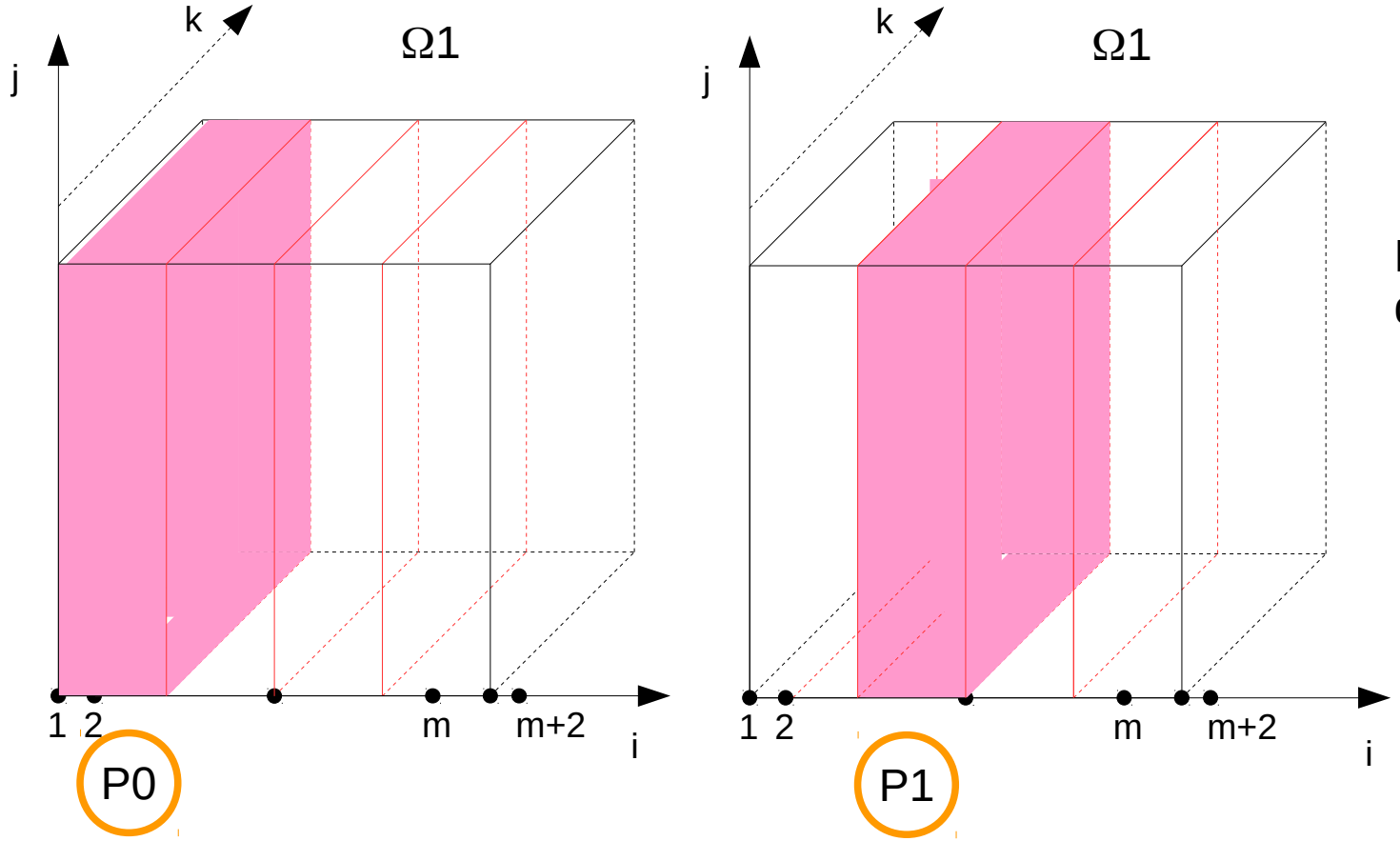
b) **Qu'on réitère ces calculs** tant que la normmax1 est > à epsilon



Calculs locaux

Portage d'un code sous MPI (Message Passing Interface)

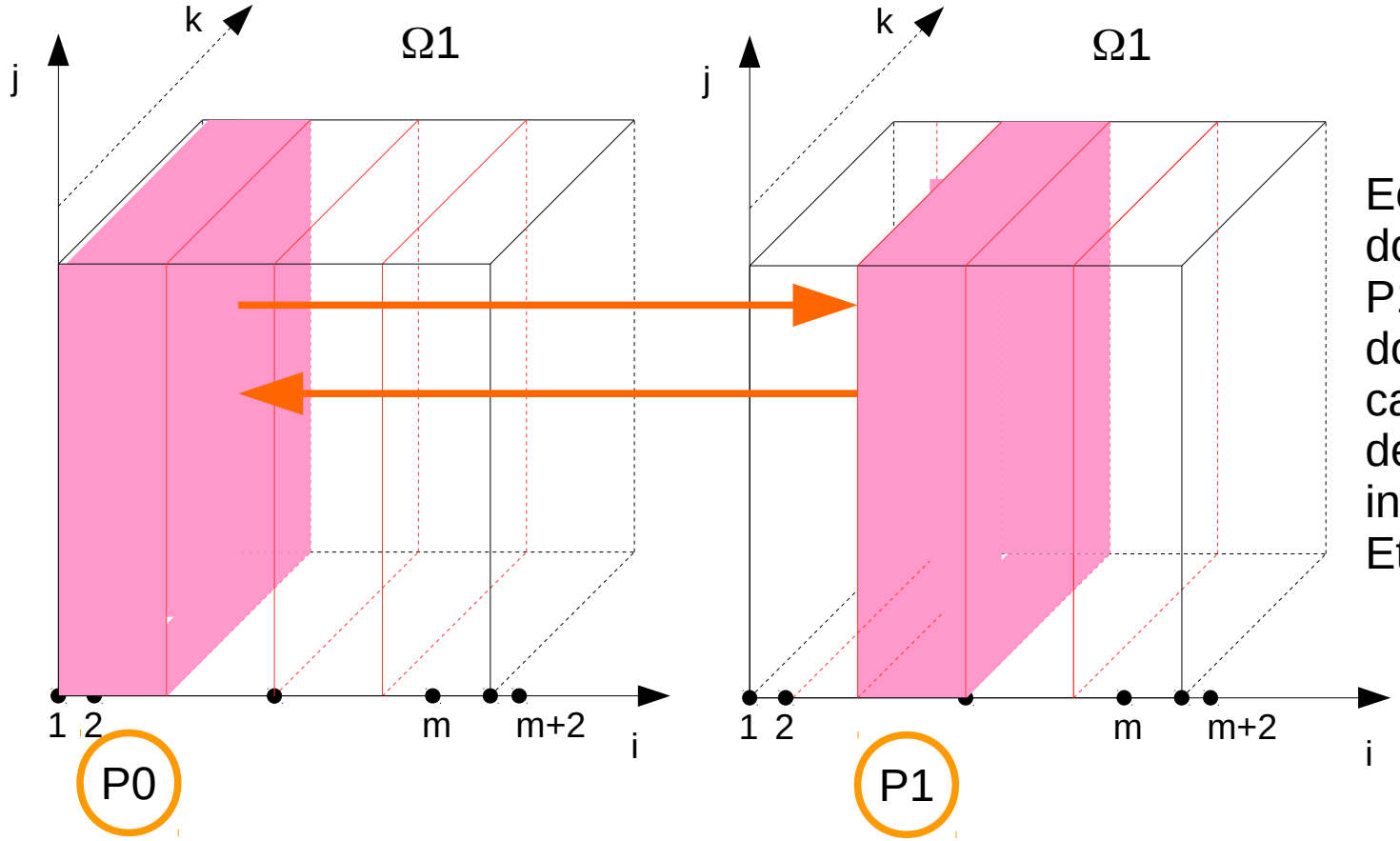
b) **Qu'on réitère ces calculs** tant que la normmax1 est > à epsilon



Echange des données

Portage d'un code sous MPI (Message Passing Interface)

b) **Qu'on réitère ces calculs** tant que la normmax1 est > à epsilon



Echange des données
P1 a besoin des données calculées par P0 de son premier indice.
Et vice-versa ...

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

Portage d'un code sous MPI (Message Passing Interface)

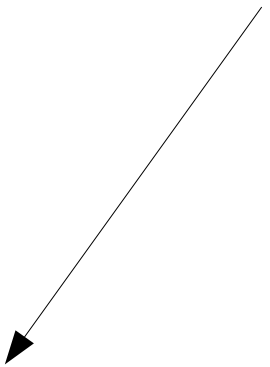
```
!=============================================================
!    Send et Recv buffer gauche et droit
!=============================================================
        IF (.NOT.(rang==nbproc-1)) THEN        ! Envoyer au processeur suivant sauf le dernier
            DO j=1,n+2
              DO k=1,nn+2
                bufferSS(j,k)=u(ifin,j,k)
              ENDDO
            ENDDO
         ENDIF

        IF (.NOT.(rang==nbproc-1)) THEN
           CALL MPI_SEND(bufferSS(1,1),itaille,MPI_REAL8,rang+1,tagsendS,comm,ierr)
        ENDIF
```

On va récupérer le dernier i calculé sur le processeur précédent

Portage d'un code sous MPI (Message Passing Interface)

```
!=============================================================
!    Send et Recv buffer gauche et droit
!=============================================================
        IF (.NOT.(rang==nbproc-1)) THEN        ! Envoyer au processeur suivant sauf le dernier
            DO j=1,n+2
              DO k=1,nn+2
                bufferSS(j,k)=u(ifin,j,k)
              ENDDO
            ENDDO
         ENDIF

        IF (.NOT.(rang==nbproc-1)) THEN
           CALL MPI_SEND(bufferSS(1,1),itaille,MPI_REAL8,rang+1,tagsendS,comm,ierr)
        ENDIF
```

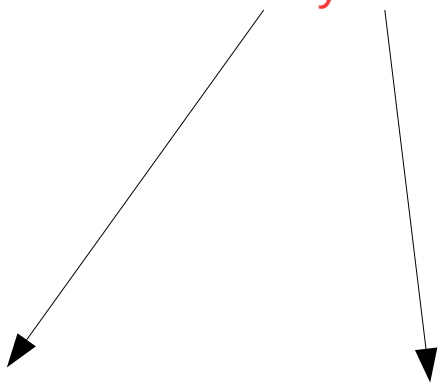
On va récupérer le dernier i calculé sur le processeur précédent



Portage d'un code sous MPI (Message Passing Interface)

```
!=============================================================
!    Send et Recv buffer gauche et droit
!=============================================================
        IF (.NOT.(rang==nbproc-1)) THEN        ! Envoyer au processeur suivant sauf le dernier
            DO j=1,n+2
              DO k=1,nn+2
                bufferSS(j,k)=u(ifin,j,k)
              ENDDO
            ENDDO
         ENDIF


        IF (.NOT.(rang==nbproc-1)) THEN
           CALL MPI_SEND(bufferSS(1,1),itaille,MPI_REAL8,rang+1,tagsendS,comm,ierr)
        ENDIF
```

On va récupérer le dernier i calculé sur le processeur précédent et l'envoyer au processeur suivant

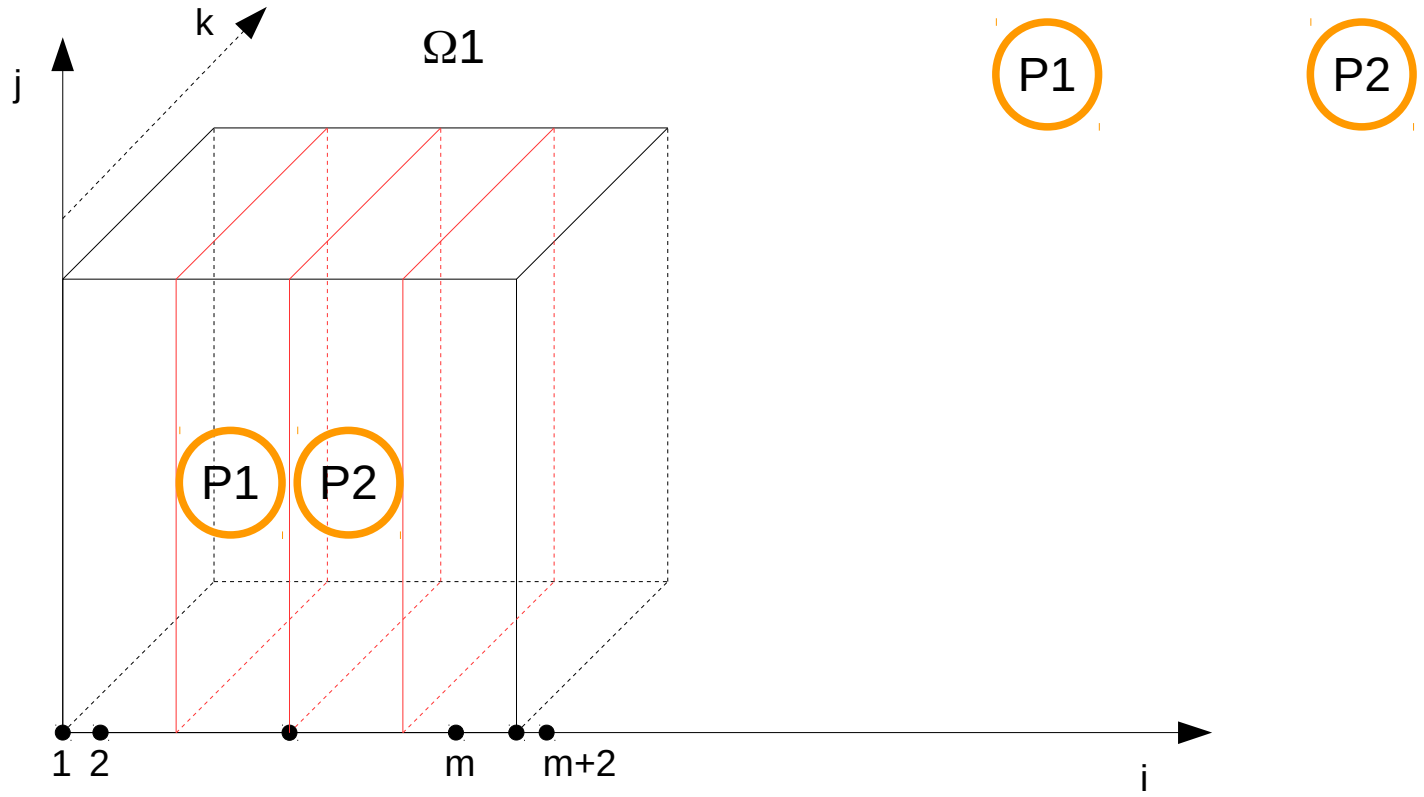


Portage d'un code sous MPI (Message Passing Interface)

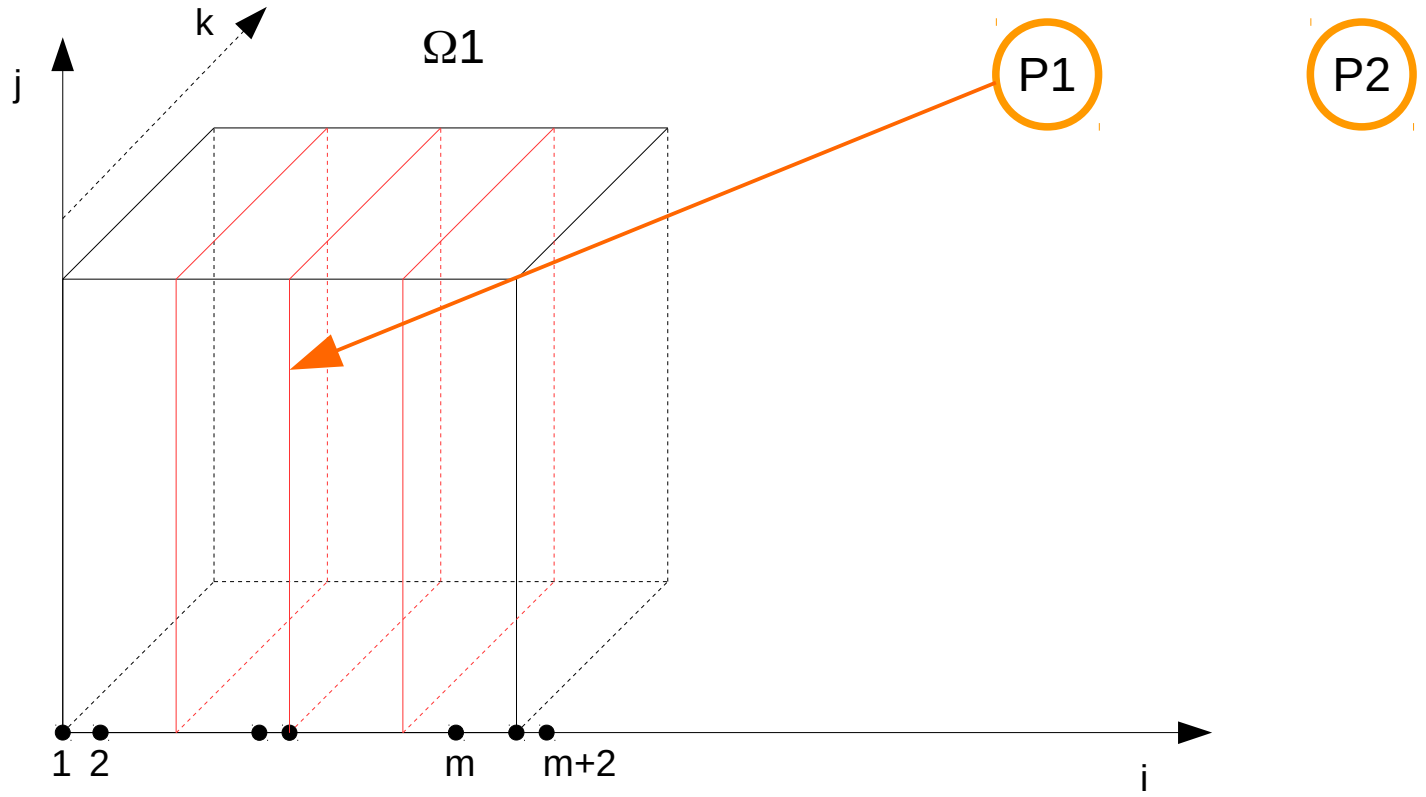
```
!Recevoir du processeur précédent sauf le premier
IF (.NOT.(rang==0)) THEN
   CALL MPI_RECV(bufferRP(1,1),itaille,MPI_REAL8,rang-1,tagrecvP,comm,STATUS,ierr)
ENDIF


IF (.NOT.(rang==0)) THEN
      DO j=1,n+2
        DO k=1,nn+2
          u(ideb-1,j,k)=bufferRP(j,k)
        ENDDO
      ENDDO
ENDIF
```

On va recevoir le i du processeur précédent et mettre à jour l'avant premier plan du processeur courant

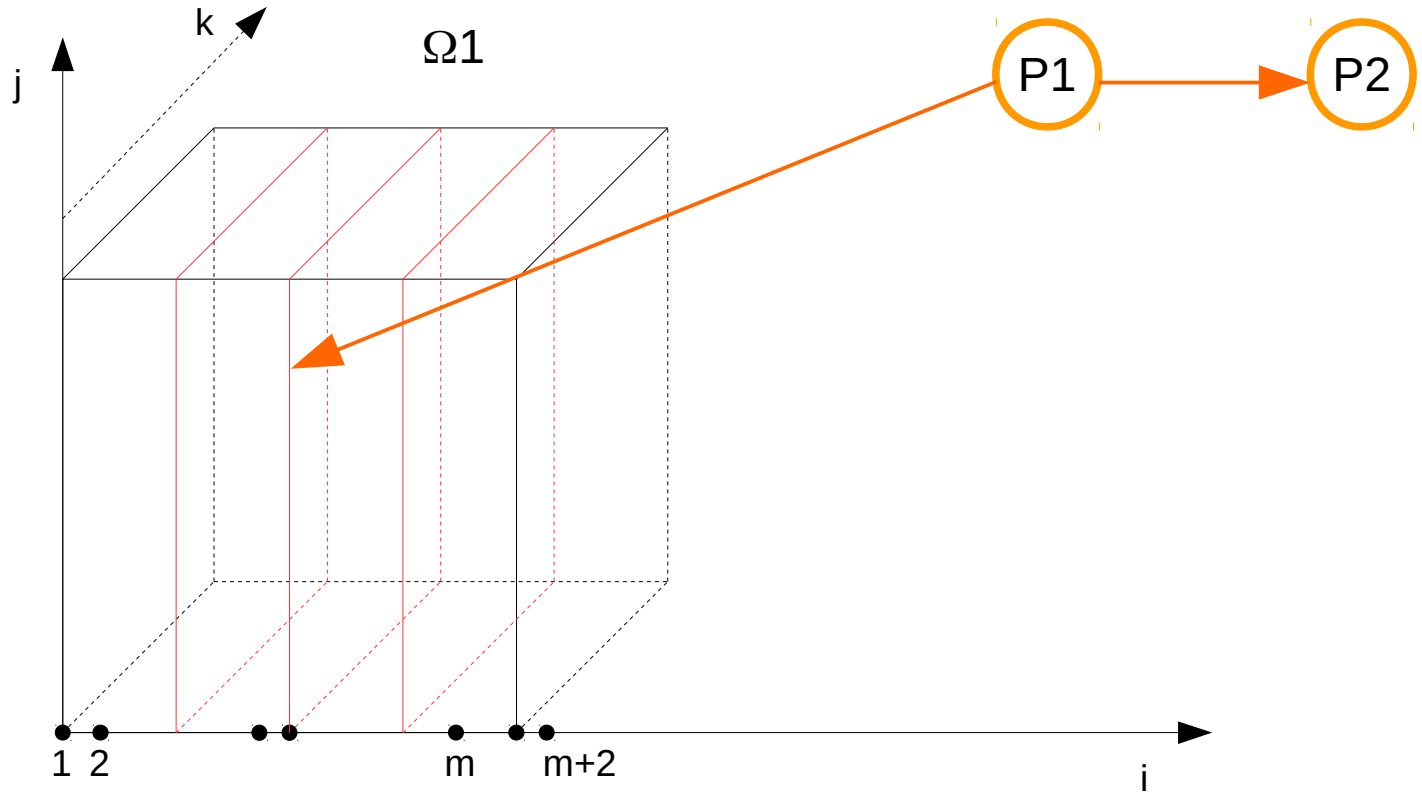
Portage d'un code sous MPI (Message Passing Interface)



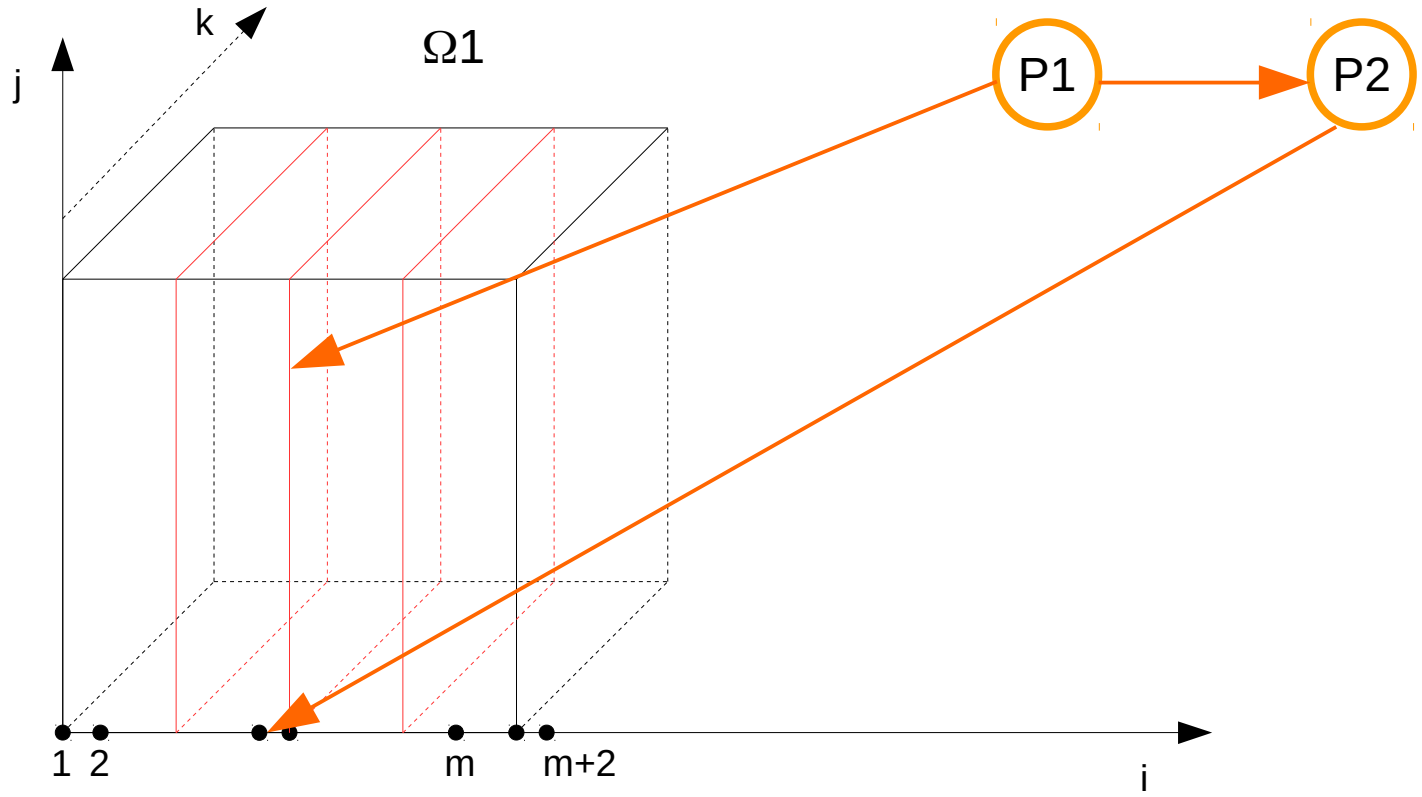
Portage d'un code sous MPI (Message Passing Interface)



Portage d'un code sous MPI (Message Passing Interface)



Portage d'un code sous MPI (Message Passing Interface)



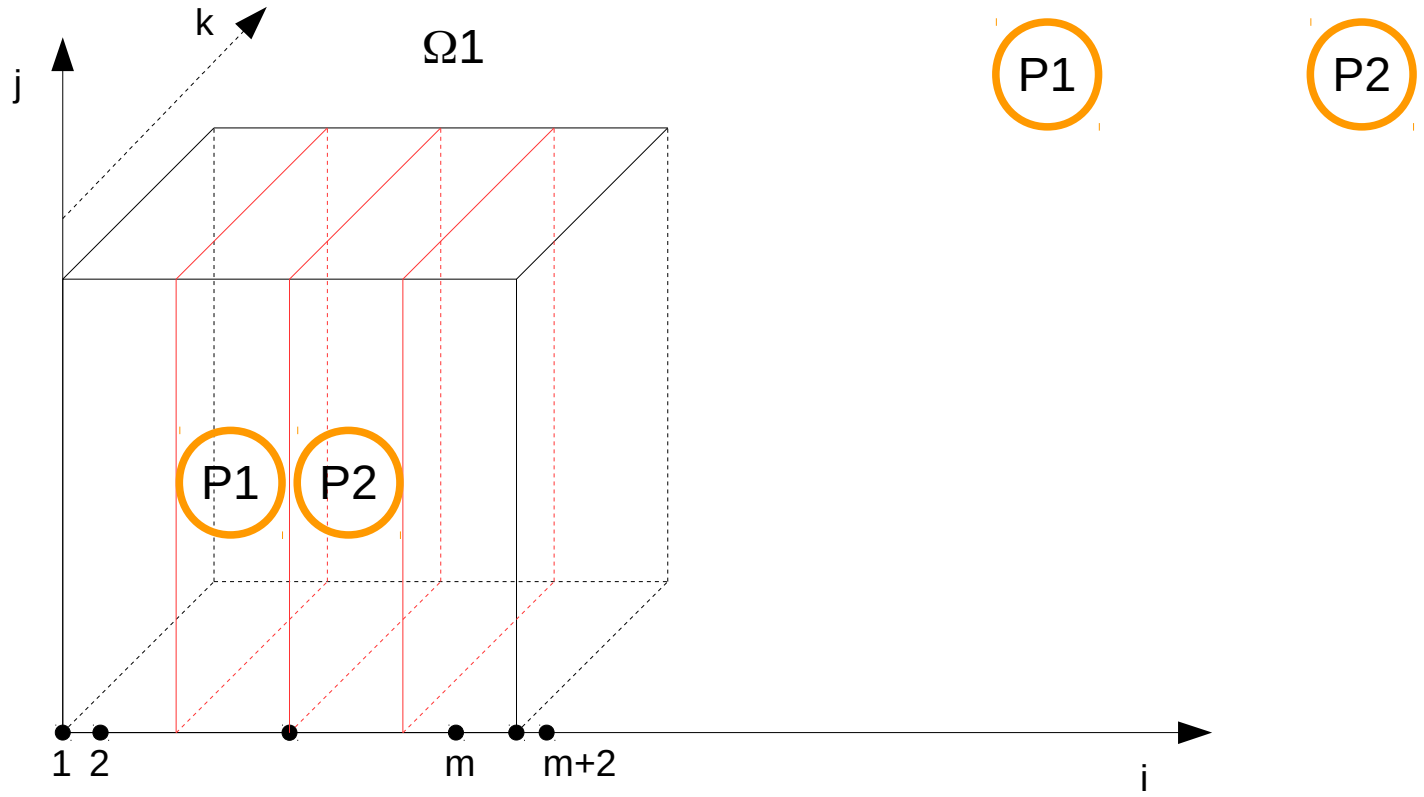
Portage d'un code sous MPI (Message Passing Interface)

```
! Envoyer au processeur precedent sauf le premier
 IF (.NOT.(rang==0)) THEN
        DO j=1,n+2
         DO k=1,nn+2
           bufferSP(j,k)=u(ideb,j,k)
         ENDDO
       ENDDO
 ENDIF

 IF (.NOT.(rang==0)) THEN
      CALL MPI_SEND(bufferSP(1,1),itaille,MPI_REAL8,&
      rang-1,tagsendP,comm,ierr)
 ENDIF
```

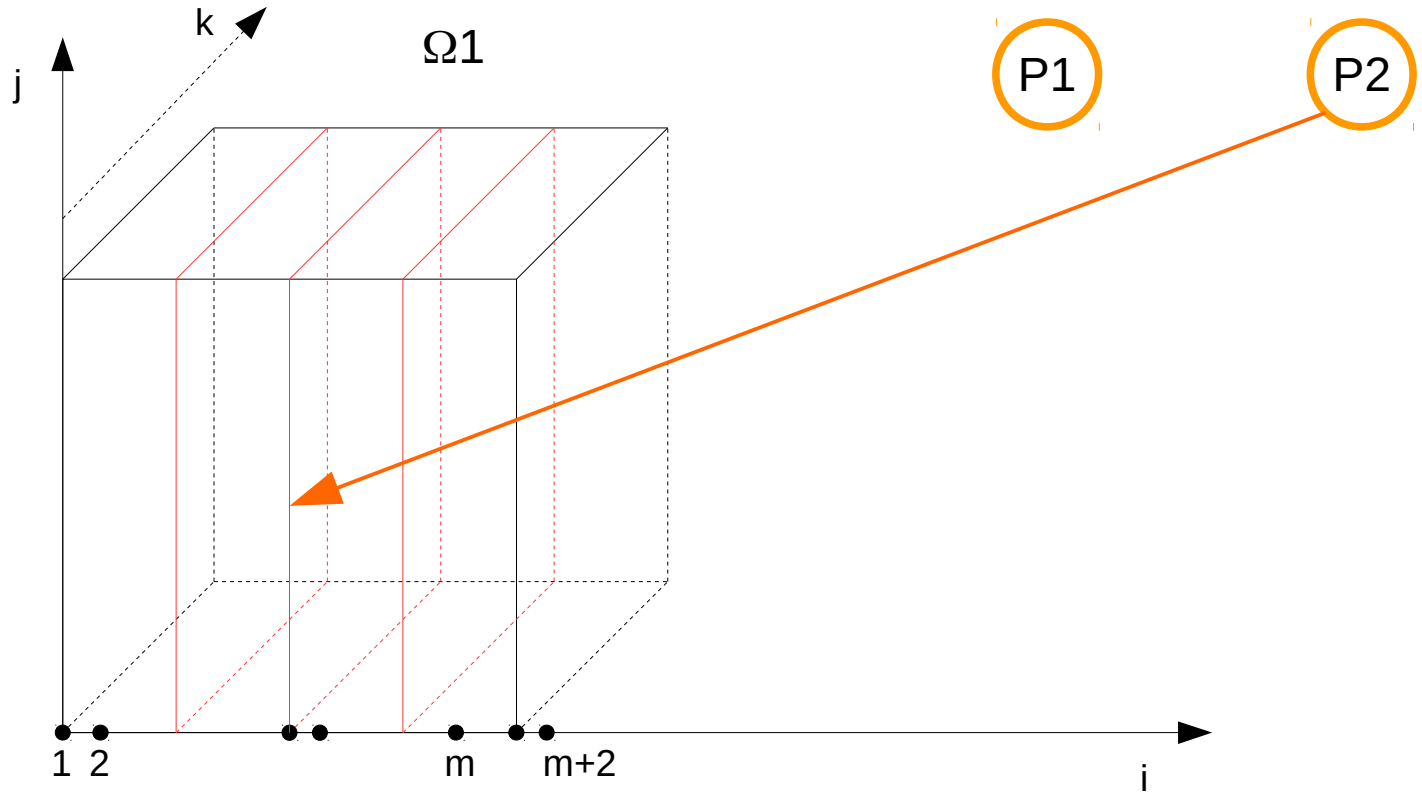
Portage d'un code sous MPI (Message Passing Interface)

```
! Recevoir du processeur suivant sauf le dernier
IF (.NOT.(rang==nbproc-1)) THEN
    CALL
MPI_RECV(bufferRS(1,1),itaille,MPI_REAL8,rang+1,tagrecvS,comm,STATUS,ierr)
ENDIF

IF (.NOT.(rang==nbproc-1)) THEN
  DO j=1,n+2
   DO k=1,nn+2
    u(ifin+1,j,k)=bufferRS(j,k)
   ENDDO
  ENDDO
ENDIF
```

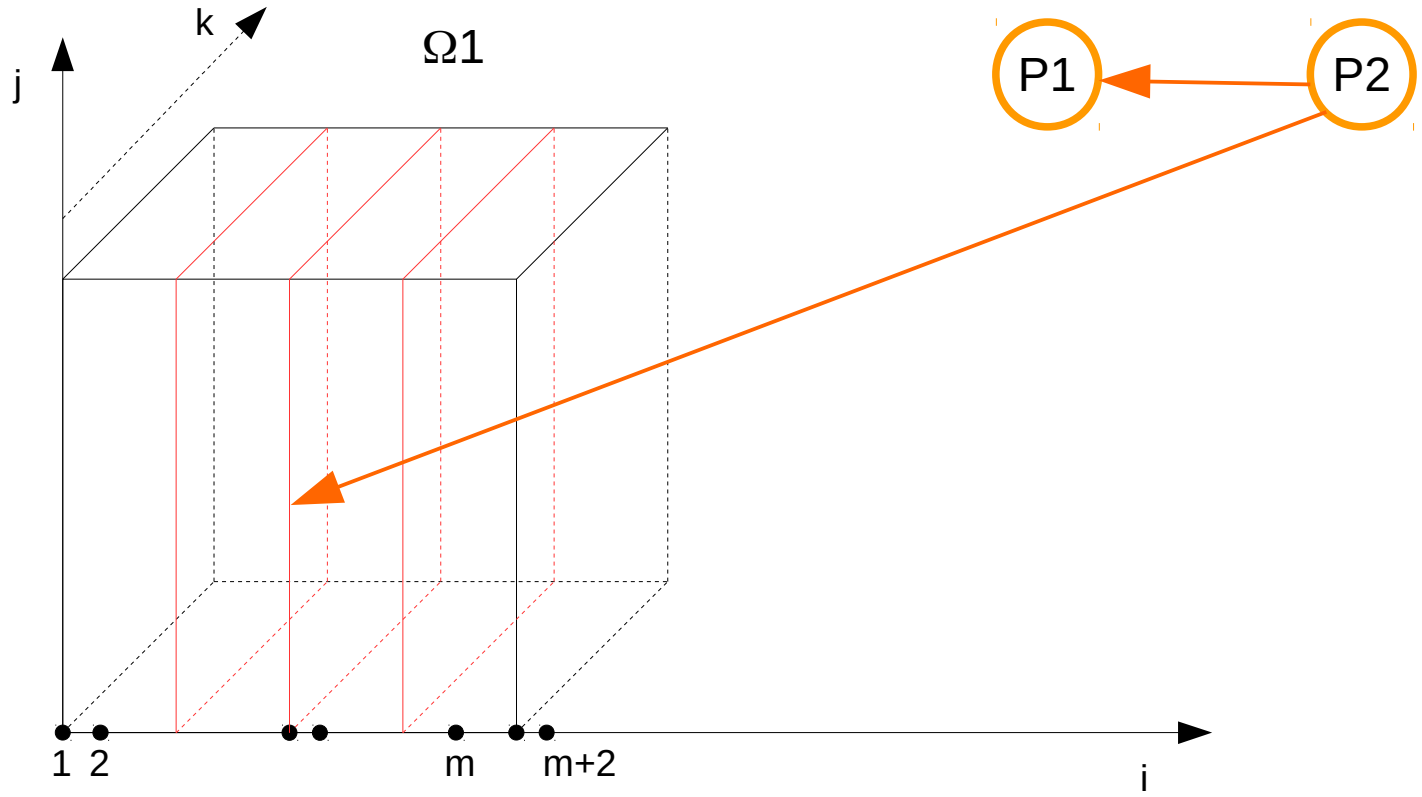
Portage d'un code sous MPI (Message Passing Interface)



Portage d'un code sous MPI (Message Passing Interface)

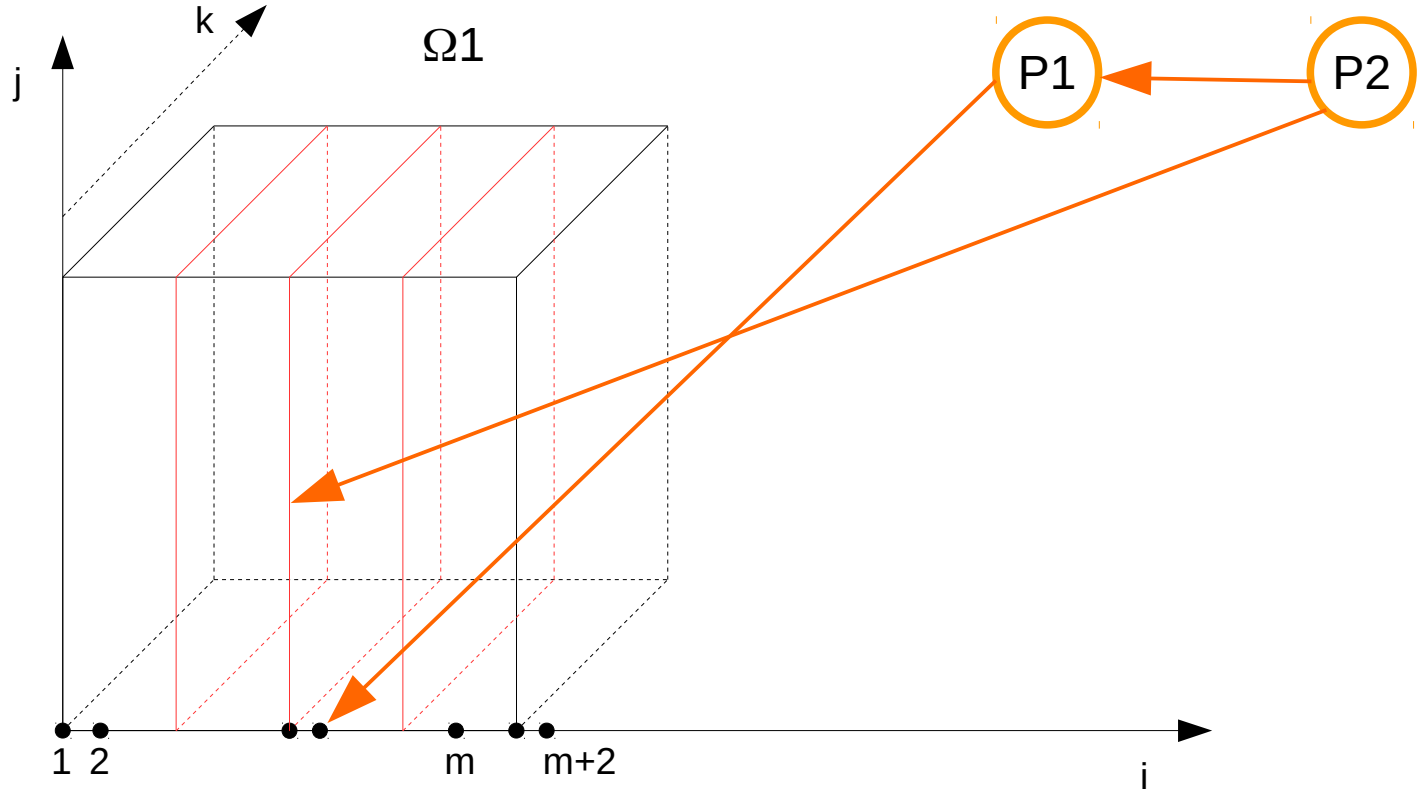


Portage d'un code sous MPI (Message Passing Interface)



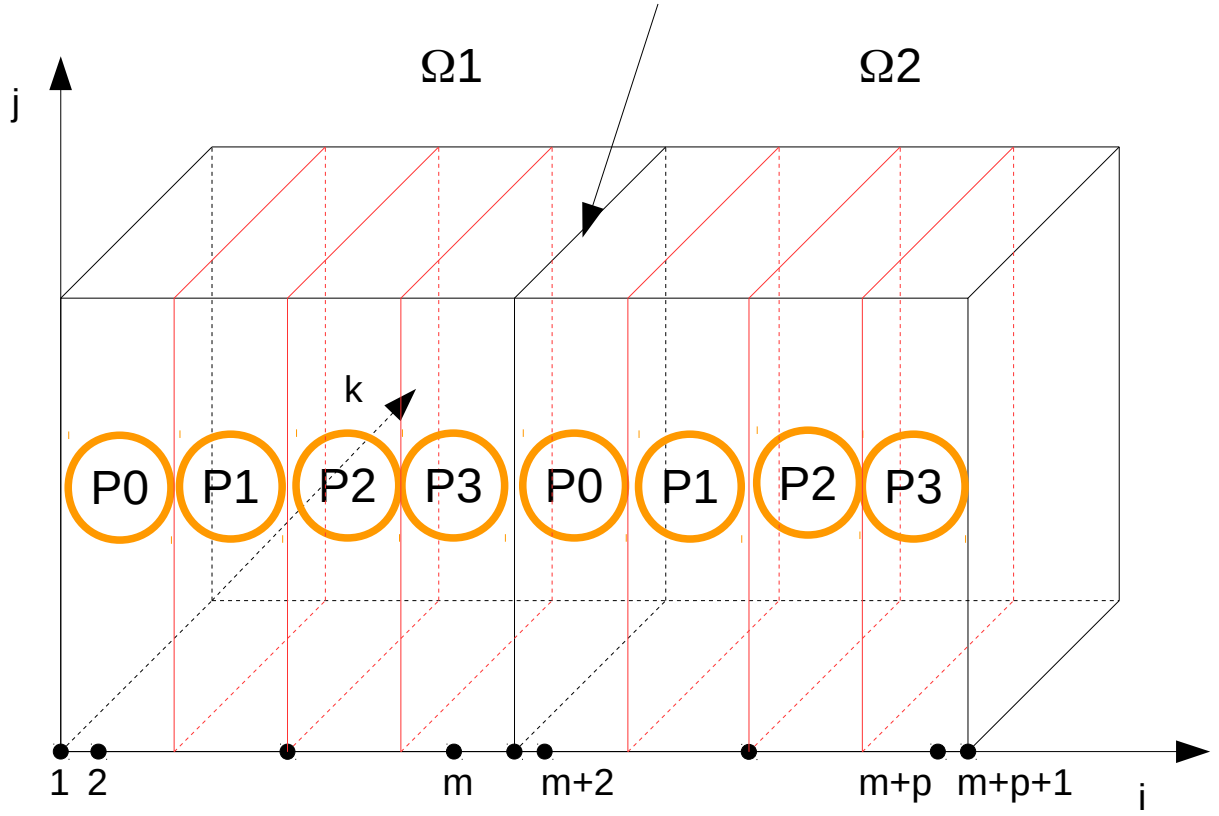
Portage d'un code sous MPI (Message Passing Interface)

(voir fichier ExerciceMPI_7.f90)



Portage d'un code sous MPI (Message Passing Interface)

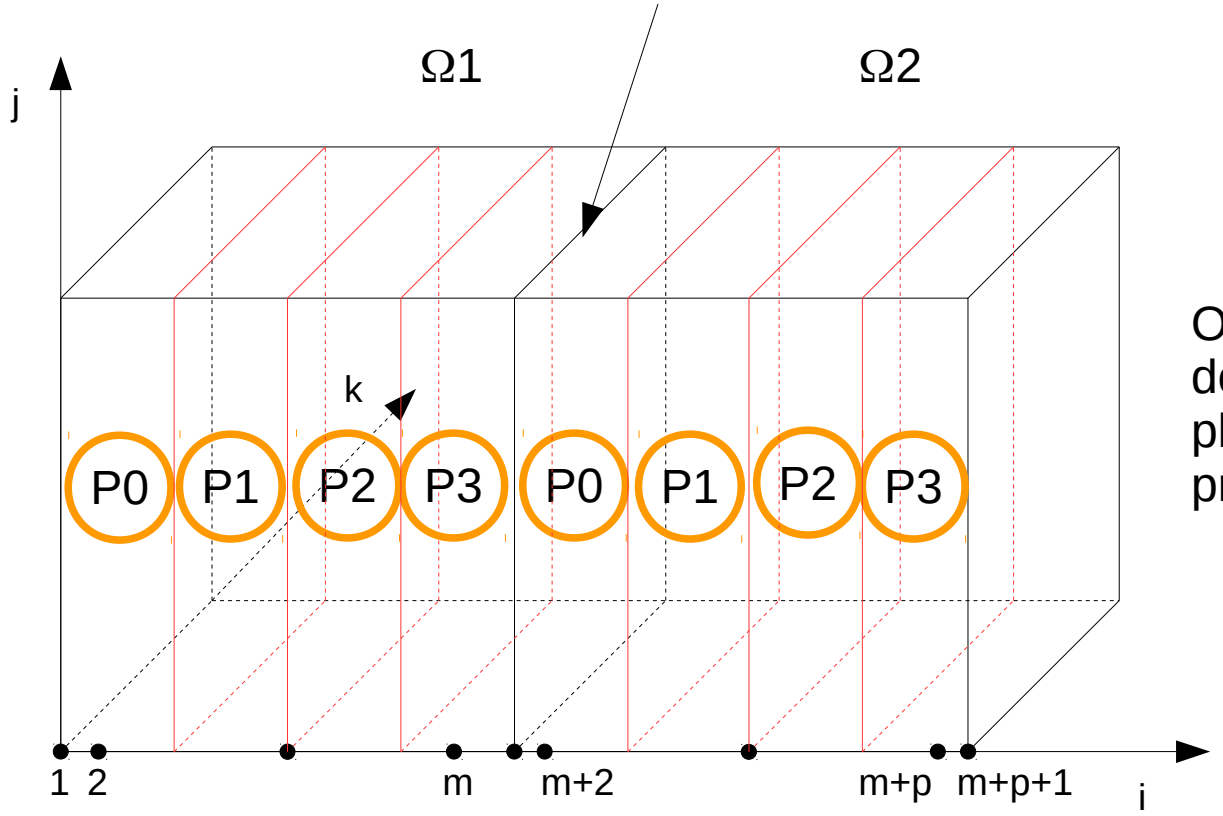
La transition entre Omega 1 et Omega 2



Omega2 a besoin des données calculées par le dernier processeur d'Omega1

Portage d'un code sous MPI (Message Passing Interface)

La transition entre Omega 1 et Omega 2



On va échanger les données du dernier plan de P3 avec le premier plan de P0

MPI : communications point à point

```
! Recuperer le dernier calcul (bufferdroit) de Omega 1 --> 2
      IF (rang==nbproc-1) THEN
        DO j=1,n+2       ! preparation des buffers
          DO k=1,nn+2
            bufferSS(j,k)=u(ifin_omega1,j,k)
          ENDDO
        ENDDO
      ENDIF
      IF (rang==nbproc-1) THEN
            CALL MPI_SEND(bufferSS(1,1),itaille,MPI_REAL8,0,111,comm,ierr) ! on
envoie au premier processeur
      ENDIF
```

(voir fichier ExerciceMPI_8.f90)

MPI : communications point à point

```
IF (rang==0) THEN ! donnee d'omega 1
      CALL MPI_RECV(bufferRS(1,1),itaille,MPI_REAL8,nbproc-
1,111,comm,STATUS,ierr)
ENDIF
IF (rang==0) THEN ! donnee d'omega 1
  DO j=1,n+2
    DO k=1,nn+2
      u(ideb_omega2-1,j,k)=bufferRS(j,k)
    ENDDO
  ENDDO
ENDIF
```

Portage d'un code sous MPI (Message Passing Interface)

Qu'a-t-on fait ?

Partage en sous-domaines

Calculs locaux

Echange de données entre tranche

Transfert des résultats de la dernière tranche du dernier sous-domaine d'Omega 1 à Omega 2

Si on execute le programme, est-ce bon ?

Portage d'un code sous MPI (Message Passing Interface)

b) **Qu'on réitère ces calculs tant que la normmax1 est > à epsilon**

- Les processeurs vivent leur vie …
- Chacun réalise leur calcul locaux et réalise les communications entre sous-domaines
- Mais comment savoir si TOUS les processeurs ont convergé ???

Portage d'un code sous MPI (Message Passing Interface)

b) **Qu'on réitère ces calculs tant que la normmax1 est > à epsilon**

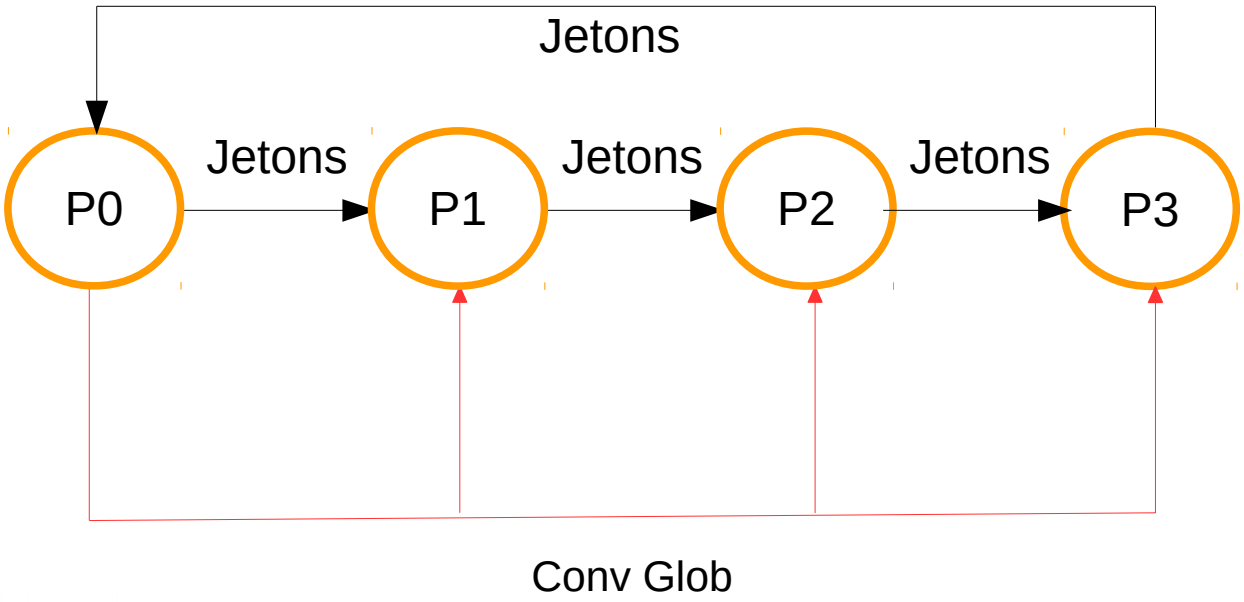
- On va réaliser un test d'arrêt.
- ie trouver un moyen de connaître l'état des convergences des processeurs afin de pouvoir stopper leur exécution.

Portage d'un code sous MPI (Message Passing Interface)

b) **Qu'on réitère ces calculs tant que la normmax1 est > à epsilon**

On peut utiliser un moyen décentralisé :

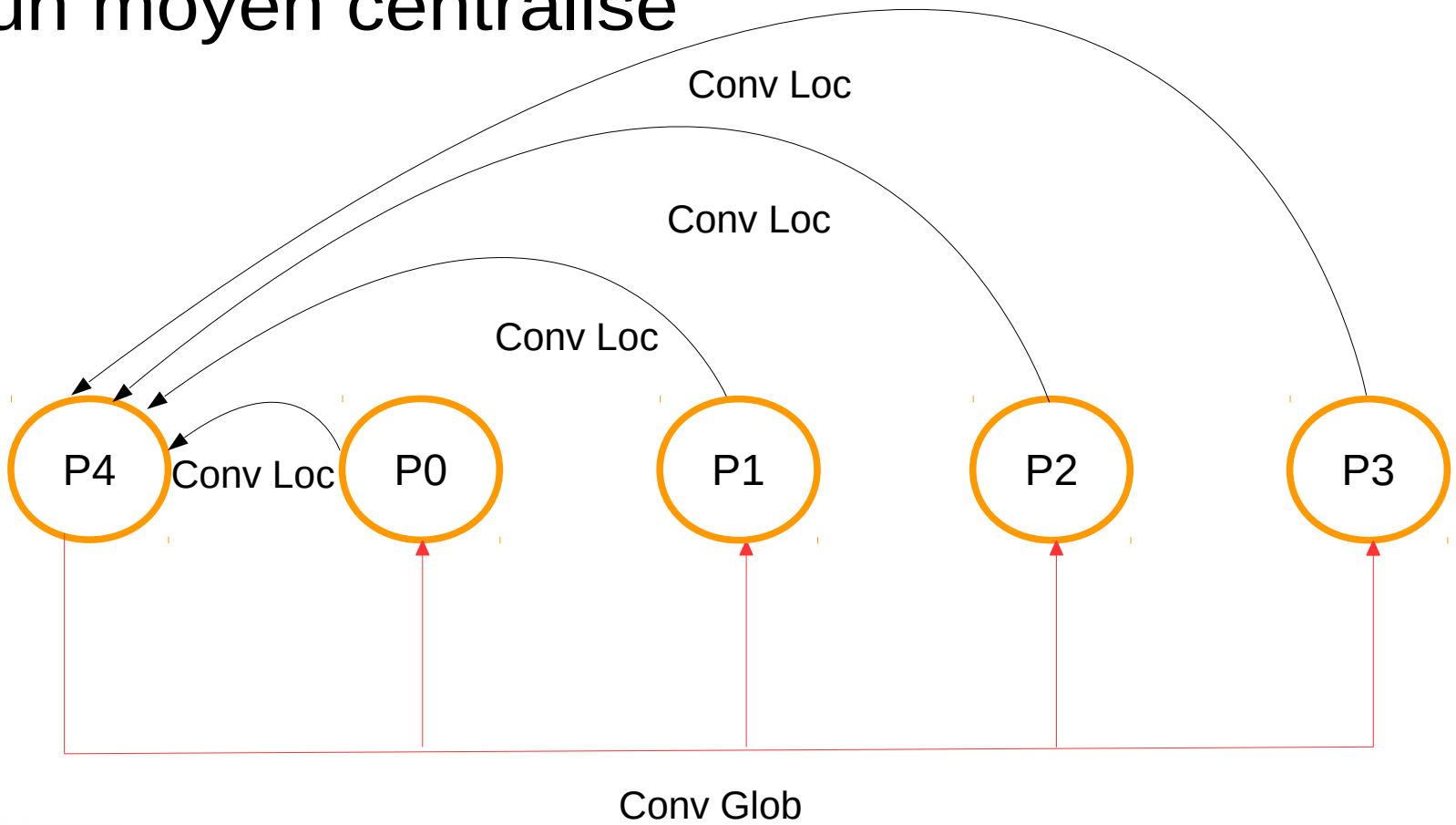
Jetons = conv loc de tous les Pn



Portage d'un code sous MPI (Message Passing Interface)

b) **Qu'on réitère ces calculs tant que la normmax1 est > à epsilon**

Ou un moyen centralisé



Portage d'un code sous MPI (Message Passing Interface)

b) **Qu'on réitère ces calculs tant que la normmax1 est > à epsilon**

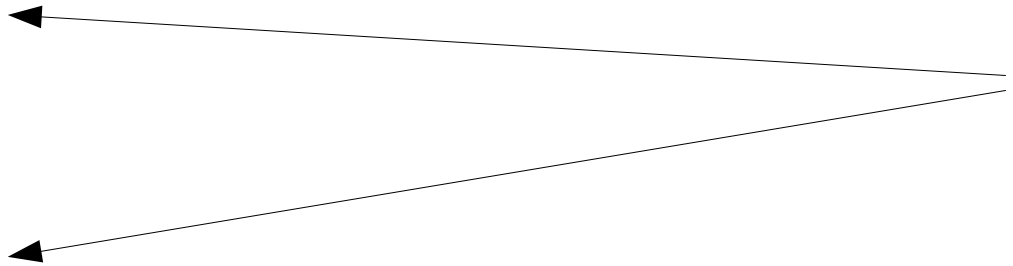
```
IF (rang==master) THEN
      letag=0
      DO WHILE (.NOT.(letag==1))
        CALL MPI_RECV(cmd,1,MPI_INTEGER,MPI_ANY_SOURCE,MPI_ANY_TAG,comm,STATUS,ierr)
        lenum=STATUS(MPI_SOURCE)
        letag=STATUS(MPI_TAG)
        IF (letag==2) THEN
          DO i=0,nbproc-1
             convloc(i)=0
          ENDDO
          bcast=0
        ENDIF
        IF (letag==6) THEN
          IF (cmd==2) THEN
            CALL MPI_SEND(convloc(1),nbproc,MPI_INTEGER,lenum,letag,comm,ierr)
          ELSE
             convloc(lenum)=cmd
          ENDIF
       ENDIF
     ENDDO
ELSE ! les processeurs qui calculent
```

Boucle sur une réceptio
Attend un message
De n'importe qui ...

Portage d'un code sous MPI (Message Passing Interface)

b) **Qu'on réitère ces calculs tant que la normmax1 est > à epsilon**

```
IF (rang==master) THEN
      letag=0
      DO WHILE (.NOT.(letag==1))
        CALL MPI_RECV(convl,1,MPI_INTEGER,MPI_ANY_SOURCE,MPI_ANY_TAG,comm,STATUS,ierr)
        lenum=STATUS(MPI_SOURCE)
        letag=STATUS(MPI_TAG)
        IF (letag==2) THEN
          DO i=0,nbproc-1
             convloc(i)=0
          ENDDO
          bcast=0
        ENDIF
        IF (letag==6) THEN
          IF (convl==2) THEN
            CALL MPI_SEND(convloc(1),nbproc,MPI_INTEGER,lenum,letag,comm,ierr)
          ELSE
             convloc(lenum)=convl ! 0 non convergé – 1 convergé
          ENDIF
       ENDIF
     ENDDO
ELSE ! les processeurs qui calculent
```



Suivant les tag, on réalise
Des actions

Portage d'un code sous MPI (Message Passing Interface)

b) **Qu'on réitère ces calculs tant que la normmax1 est > à epsilon**

```
IF (rang==master) THEN
      letag=0
      DO WHILE (.NOT.(letag==1))
        CALL MPI_RECV(convl,1,MPI_INTEGER,MPI_ANY_SOURCE,MPI_ANY_TAG,comm,STATUS,ierr)
        lenum=STATUS(MPI_SOURCE)
        letag=STATUS(MPI_TAG)
        IF (letag==2) THEN
          DO i=0,nbproc-1
             convloc(i)=0
          ENDDO
          bcast=0
        ENDIF
        IF (letag==6) THEN
          IF (convl==2) THEN
            CALL MPI_SEND(convloc(1),nbproc,MPI_INTEGER,lenum,letag,comm,ierr)
          ELSE
             convloc(lenum)=convl ! 0 non convergé – 1 convergé
          ENDIF
       ENDIF
     ENDDO
ELSE ! les processeurs qui calculent
```

Ici on gère un tableau des Convergences locales

Portage d'un code sous MPI (Message Passing Interface)

b) **Qu'on réitère ces calculs tant que la normmax1 est > à epsilon**

Dans les sous-programmes PTSINT1 et PTSINT2, on rajoute :

```
CALL MPI_SEND(0,1,MPI_INTEGER,master,2,comm,ierr)
! initialise le tableau
CALL MPI_SEND(2,1,MPI_INTEGER,master,6,comm,ierr)
! envoyer le tableau
CALL MPI_RECV(convloc(1),nbproc,MPI_INTEGER,master,6,comm,STATUS,ierr)
```

Portage d'un code sous MPI (Message Passing Interface)

b) **Qu'on réitère ces calculs tant que la normmax1 est > à epsilon**

```
!=====================================================================
! tester la convergence locale
!=====================================================================
        IF (normmax1.LE.epsi) THEN
          IF (convloc(rang+1)==0) THEN
            ! Envoyer au processeur maitre la convergence locale
            CALL MPI_SEND(1,1,MPI_INTEGER,master,6,comm,ierr)
            convloc(rang+1)=1
          ENDIF
        ELSE
          IF (convloc(rang+1)==1) THEN
           ! Envoyer au processeur maitre la non convergence
           CALL MPI_SEND(0,1,MPI_INTEGER,master,6,comm,ierr)
           convloc(rang+1)=0
         ENDIF
        ENDIF
```

Portage d'un code sous MPI (Message Passing Interface)

b) **Qu'on réitère ces calculs tant que la normmax1 est > à epsilon**

```
!=============================================================
! tester la convergence globale
!=============================================================
        CALL MPI_SEND(2,1,MPI_INTEGER,master,6,comm,ierr)
        CALL MPI_RECV(convloc(1),nbproc,MPI_INTEGER,master,6,comm,STATUS,ierr)

        convglob=1
        DO i=1,nbproc
          IF (convloc(i)==0) convglob=0
        ENDDO
```

Portage d'un code sous MPI (Message Passing Interface)

b) **Qu'on réitère ces calculs tant que la normmax1 est > à epsilon**

Dans les sous-programmes PTSINT1 et PTSINT2, on modifie :

convglob=0

do while(convglob==0)

et on termine le programme master avec

```
! Terminer le dernier processeur
IF (rang==0)
CALL MPI_SEND(0,1,MPI_INTEGER,master,1,comm,ierr)
```

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 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

Portage d'un code sous MPI (Message Passing Interface)

- On va changer les primitives dans les communications entre sous-domaine.
- On va rajouter MPI_Barrier entre Omega 1 et Omega 2 afin de synchroniser les processeurs qui ont détecté la convergence.
- On va ajouter MPI_Barrier à la fin des traitements des deux domaines avant de recommencer la boucle en temps.

Portage d'un code sous MPI (Message Passing Interface)

b) **Qu'on réitère ces calculs tant que la normmax1 est > à epsilon**

```
IF (letag==3) THEN   ! barriere
  barrier(lenum)=1
  bcast=1
  DO i=0,nbproc-1
    IF (barrier(i)==0) bcast=0
  ENDDO
  IF (bcast==1) THEN
    CALL MPI_BCAST(bcast,1,MPI_INTEGER,master,comm,ierr)
    DO i=0,nbproc-1
      barrier(i)=0
    ENDDO
      bcast=0
  ENDIF
ENDIF
```