TP MPI

ANGD Plasmas froids

Guy Moebs

Laboratoire de Mathématiques Jean Leray,
CNRS, Université de Nantes, École Centrale de Nantes

Octobre 2011

Liste des TPs

Compilation / exécution d'un programme MPI

TP1 : hello world !

TP2 : communications en anneau

TP3 : ma première topologie

TP Laplace

Compilation / exécution d'un programme MPI

TP1 : hello world !

TP2 : communications en anneau

TP3 : ma première topologie

TP Laplace

Compilation / exécution d'un programme MPI

- On dispose d'un *wrapper* Fortran `mpif90` basé sur gfortran et OpenMPI

```
mpif90 -O2 source.f90 -o a.out
```

Compilation / exécution d'un programme MPI

- On dispose d'un *wrapper* Fortran `mpif90` basé sur gfortran et OpenMPI

  ```
  mpif90 -O2 source.f90 -o a.out
  ```

- On se dispense d'authentification sur les nœuds de la grappe
  - creer une clef ssh :

    ```
    ssh-keygen -t dsa
    ```
  - la placer au bon endroit :

    ```
    cd .ssh ; cp id_dsa.pub authorized_keys
    ```
  - lancer le ssh-agent :

    ```
    eval `ssh-agent`
    ```
  - ajouter la passphrase :

    ```
    ssh-add $HOME/.ssh/id_dsa
    ```

Compilation / exécution d'un programme MPI

- On dispose d'un *wrapper* Fortran `mpif90` basé sur gfortran et OpenMPI

```
mpif90 -O2 source.f90 -o a.out
```

- On se dispense d'authentification sur les nœuds de la grappe
  - creer une clef ssh :
    ```
    ssh-keygen -t dsa
    ```
  - la placer au bon endroit :
    ```
    cd .ssh ; cp id_dsa.pub authorized_keys
    ```
  - lancer le ssh-agent :
    ```
    eval `ssh-agent`
    ```
  - ajouter la passphrase :
    ```
    ssh-add $HOME/.ssh/id_dsa
    ```
- On lance son programme en précisant le nombre de tâches et les machines cibles

```
mpirun --prefix /softs/openmpi-1.4.3 -np 16 -host n1,n2,n3,n4
./a_mpi_03.out
```

Compilation / exécution d'un programme MPI

TP1 : hello world !

TP2 : communications en anneau

TP3 : ma première topologie

TP Laplace

TP1 : hello world

Ecrire un programme qui réalise les opérations suivantes :

- initialisation de l'environnement MPI
- lecture d'un entier par le processus de rang 0
- diffusion de cet entier à tous les processus

Correction du TP1

```
   program hello_bcast
   use mpi
   implicit none
!
   integer :: myrank, nbproc
   integer :: icode, i
!
   call MPI_Init (icode)
!
   call MPI_Comm_Rank (MPI_COMM_WORLD, myrank, icode)
   call MPI_Comm_Size (MPI_COMM_WORLD, nbproc, icode)
!
   if (myrank == 0) read (5,*) i
!
   call MPI_BCast (i, 1, MPI_INTEGER, 0, MPI_COMM_WORLD, icode)
   write (6,'(2(A,I6))') 'myrank = ', myrank, ' i = ', i
!
   call MPI_Finalize (icode)
   stop
   end program hello_bcast
```

Correction du TP1

```
echo 5 | mpirun --prefix /softs/openmpi-1.4.3 -np 16 -host n1,n2,n3,n4
./tp1_03.out
myrank =     4    i =    5
myrank =     8    i =    5
myrank =    12    i =    5
myrank =     5    i =    5
myrank =     9    i =    5
myrank =    13    i =    5
myrank =     0    i =    5
myrank =     2    i =    5
myrank =     6    i =    5
myrank =    10    i =    5
myrank =    14    i =    5
myrank =     1    i =    5
myrank =     3    i =    5
myrank =     7    i =    5
myrank =    11    i =    5
myrank =    15    i =    5
```

TP2 : communications en anneau

Ecrire un programme qui réalise les opérations suivantes :

- initialisation de l'environnement MPI
- envoi par le processus 0 d'un entier au processus de rang suivant
- attente d'un entier en provenance du processus de rang immédiatement inférieur
- après réception, incrément avec son rang et envoi du résultat au suivant
- envoi par le dernier au processus de rang 0
- la boucle est bouclée !
- calcul de la somme des valeurs locales et envoi à tous les processus

Correction du TP2 (1/2)

```
   program anneau
   use mpi
   implicit none
!
   integer :: myrank, nbproc
   integer :: icode, i, j
   integer :: inext, iprev
   integer, dimension(mpi_status_size) :: status
!
   call MPI_Init (icode)
   call MPI_Comm_Rank (MPI_COMM_WORLD, myrank, icode)
   call MPI_Comm_Size (MPI_COMM_WORLD, nbproc, icode)
!
   inext = mod (myrank+1,nbproc)
   iprev = mod (myrank-1+nbproc,nbproc)
   write (6,'(A,3I6)') 'myrank : iprev, inext', myrank, iprev, inext
!
   if (myrank == 0) then
      i = 1234
      call MPI_Send (i, 1, MPI_INTEGER, inext, 100+myrank,
  &                  MPI_COMM_WORLD, icode)
```

Correction du TP2 (2/2)

```
      call MPI_Recv (j, 1, MPI_INTEGER, iprev, 100+iprev,
     &               MPI_COMM_WORLD, status, icode)
!
      else
      call MPI_Recv (j, 1, MPI_INTEGER, iprev, 100+iprev,
     &               MPI_COMM_WORLD,status, icode)
!
      i = j + myrank
      call MPI_Send (i, 1, MPI_INTEGER, inext, 100+myrank,
     &               MPI_COMM_WORLD, icode)
      end if
!
      call MPI_AllReduce (j, is, 1, MPI_INTEGER, MPI_SUM, MPI_COMM_WORLD,
     &                    icode)
      write (6,'(4(A,I6,2x))') 'myrank = ', myrank, 'i = ', i, 'j = ', j,
     &                         'is = ', is
!
      call MPI_Finalize (icode)
      stop
      end program anneau
```

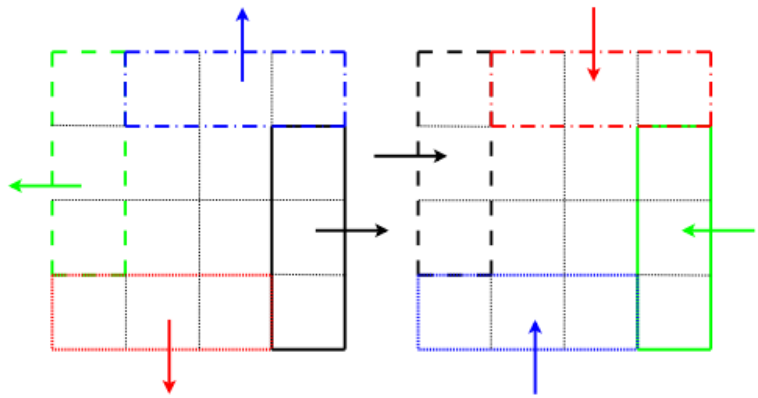
Correction du TP2

```
mpirun --prefix /softs/openmpi-1.4.3 -np 8 -host n1,n2,n3,n4
./tp2_03.out
myrank : iprev, inext 0 7 1
myrank : iprev, inext 4 3 5
myrank : iprev, inext 1 0 2
myrank : iprev, inext 5 4 6
myrank : iprev, inext 2 1 3
myrank : iprev, inext 3 2 4
myrank : iprev, inext 6 5 7
myrank : iprev, inext 7 6 0
myrank =   0 i = 1234 j = 1262 is = 9956
myrank =   4 i = 1244 j = 1240 is = 9956
myrank =   2 i = 1237 j = 1235 is = 9956
myrank =   3 i = 1240 j = 1237 is = 9956
myrank =   6 i = 1255 j = 1249 is = 9956
myrank =   7 i = 1262 j = 1255 is = 9956
myrank =   1 i = 1235 j = 1234 is = 9956
myrank =   5 i = 1249 j = 1244 is = 9956
```

TP3 : ma première topologie

Ecrire un programme qui réalise les opérations suivantes :

- initialisation de l'environnement MPI
- création d'une topologie 2D, avec recherche des voisins immédiats
- création de type ligne(3) et colonne(3) dans des matrices 4x4
- échange avec les voisins des zones colorées de / vers deux tableaux différents



Correction du TP3 (1/6)

```
   PROGRAM TOPOLOGIE
   USE mpi
   IMPLICIT NONE
!
   INTEGER, DIMENSION(4,4) :: MOI, EUX
!
   INTEGER :: nbproc, myrank, i, j
   INTEGER :: icode
   INTEGER :: tag1, tag2
!
   INTEGER, DIMENSION(4) :: voisin
   INTEGER, PARAMETER :: N=1, E=2, S=3, W=4
!
   INTEGER, DIMENSION(MPI_STATUS_SIZE) :: status
   INTEGER :: comm2d, colonne, ligne
   INTEGER, DIMENSION(2) :: dims
   LOGICAL, DIMENSION(2) :: periods
!
   icode = 0
   tag1 = 100
   tag2 = 200
```

Correction du TP3 (2/6)

```
!Initialisation de MPI
   CALL MPI_Init (icode)
!
!Savoir quel processeur je suis
   CALL MPI_Comm_Rank (MPI_COMM_WORLD, myrank, icode)
!
!Connaitre le nombre total de processeurs
   CALL MPI_Comm_Size (MPI_COMM_WORLD, nbproc, icode)
!
   dims(1:2) = 0
   CALL MPI_Dims_Create (nbproc, 2, dims, icode)
!
!Creation de la grille de processeurs 2D avec periodicite
   periods(1:2) = .TRUE.
!
   CALL MPI_Cart_Create (MPI_COMM_WORLD, 2, dims, periods, .TRUE.,
  &                      comm2d, icode)
!Recherche de ses 4 voisins pour chaque processeur
!Initialisation du tableau voisin
   voisins(1:4) = (/ MPI_PROC_NULL, MPI_PROC_NULL, MPI_PROC_NULL,
  &                  MPI_PROC_NULL /)
```

Correction du TP3 (3/6)

```
!Recherche des voisins Ouest et Est
   CALL MPI_Cart_Shift (comm2d, 0, 1, voisin(W), voisin(E), icode)
!
!Recherche des voisins Nord et Sud
   CALL MPI_Cart_Shift (comm2d, 1, 1, voisin(S), voisin(N), icode)
!
   write (6,'(7I4)') myrank, voisin(1:4), dims(1:2)
!
!Initialise les tableaux
   do j = 1, 4
      do i = 1, 4
         MOI(i,j) = 100*i + 1000 * j
         EUX(i,j) = 0
      end do
   end do
!
!Affiche les tableaux
   do i = 1, 4
      write (6,1010) (MOI(i,j),j=1,4), (EUX(i,j),j=1,4)
   end do
 1010 format (4I6,10x,4I6)
```

Correction du TP3 (4/6)

```
!Envoi au voisin W et reception du voisin E
!en utilisant le type predefini colonne
   CALL MPI_Type_Contiguous (3,                ! longueur d'un bloc
  &                          MPI_INTEGER,     ! type initial
  &                          colonne, icode)  ! nouveau type
   CALL MPI_Type_Commit (colonne, icode)       ! validation
!
!Envoi au voisin N et reception du voisin S
!en utilisant le type predefini ligne
   CALL MPI_Type_Vector (3,                    ! nombre de blocs
  &                      1,                    ! longueur d'un bloc
  &                      4,                    ! pas entre le debut de
  &                      MPI_INTEGER,          ! deux blocs
                                               ! consecutifs
  &                      ligne, icode)         ! nouveau type
   CALL MPI_Type_Commit (ligne, icode)         ! validation
```

Correction du TP3 (5/6)

```
!Envoi au voisin N et reception du voisin S
!en utilisant le type predefini ligne
   CALL MPI_SendRecv (MOI(1,2), 1, ligne, voisin(N), tag1,
  &                   EUX(4,1), 1, ligne, voisin(S), tag1,
  &                   comm2d, status, icode)
!
!Envoi au voisin S et reception du voisin N
!en utilisant le type predefini ligne
   CALL MPI_SendRecv (MOI(4,1), 1, ligne, voisin(S), tag2,
  &                   EUX(1,2), 1, ligne, voisin(N), tag2,
  &                   comm2d, status, icode)
!
!Envoi au voisin W et reception du voisin E
!en utilisant le type predefini colonne
!type_ligne
   CALL MPI_SendRecv (MOI(1,1), 1, colonne, voisin(W), tag1,
  &                   EUX(2,4), 1, colonne, voisin(E), tag1,
  &                   comm2d, status, icode)
```

Correction du TP3 (6/6)

```
!Envoi au voisin E et reception du voisin W
!en utilisant le type predefini colonne
   CALL MPI_SendRecv (MOI(2,4), 1, colonne, voisin(E), tag2,
  &                   EUX(1,1), 1, colonne, voisin(W), tag2,
  &                   comm2d, status, icode)
!
!Affiche les tableaux
   do i = 1, 4
      write (6,1010) (MOI(i,j),j=1,4), (EUX(i,j),j=1,4)
   enddo
!
!Liberation du type type_ligne et du communicateur comm2d
   CALL MPI_Type_Free (ligne, icode)
   CALL MPI_Type_Free (colonne, icode)
!
   CALL MPI_Comm_Free (comm2d, icode)
!
!Desactivation de MPI
   CALL MPI_Finalize (icode)
```

Correction du TP3

```
mpirun --prefix /softs/openmpi-1.4.3 -np 2 -host n1,n2 ./tp3_03.out
[1,0]<stdout>: 0 0 1 0 1 2 1
[1,0]<stdout>: 1100 2100 3100 4100    0    0    0    0
[1,0]<stdout>: 1200 2200 3200 4200    0    0    0    0
[1,0]<stdout>: 1300 2300 3300 4300    0    0    0    0
[1,0]<stdout>: 1400 2400 3400 4400    0    0    0    0

[1,0]<stdout>: 1100 2100 3100 4100 4200 1400 2400 3400
[1,0]<stdout>: 1200 2200 3200 4200 4300    0    0 1100
[1,0]<stdout>: 1300 2300 3300 4300 4400    0    0 1200
[1,0]<stdout>: 1400 2400 3400 4400 2100 3100 4100 1300
[1,1]<stdout>: 1 1 0 1 0 2 1
[1,1]<stdout>: 1100 2100 3100 4100    0    0    0    0
[1,1]<stdout>: 1200 2200 3200 4200    0    0    0    0
[1,1]<stdout>: 1300 2300 3300 4300    0    0    0    0
[1,1]<stdout>: 1400 2400 3400 4400    0    0    0    0

[1,1]<stdout>: 1100 2100 3100 4100 4200 1400 2400 3400
[1,1]<stdout>: 1200 2200 3200 4200 4300    0    0 1100
[1,1]<stdout>: 1300 2300 3300 4300 4400    0    0 1200
[1,1]<stdout>: 1400 2400 3400 4400 2100 3100 4100 1300
```

Compilation / exécution d'un programme MPI

TP1 : hello world !

TP2 : communications en anneau

TP3 : ma première topologie

TP Laplace

TP Laplace

- Analyser le code séquentiel (le faire “tourner”)
- Analyser les algorithmes mis en œuvre
- Effectuer le découpage, construire la topologie (grille de processus, voisins, ...)
- Déterminer les dépendances entre les données
- Faire le bilan : qui a besoin de quoi, quand et où !
- Déterminer les structures des données à échanger
- Construire les types MPI ad hoc
- Dupliquer et modifier le code séquentiel !

Dépendance dans l'algorithme

```
do j = 1, ny
   do i = 1, nx
      V(i,j) = omega1 * V_old(i,j)
&             + ( sou(i,j)    (1)                    (2)
&               + ve(i,j) * V(i+1,j  ) + vw(i,j) * V(i-1,j  )
&               + vn(i,j) * V(i  ,j+1) + vs(i,j) * V(i  ,j-1)
&               ) * vc(i,j)
   end do
end do
```

- Il y a plusieurs dépendances à satisfaire

Dépendance dans l'algorithme

```
do j = 1, ny
   do i = 1, nx
      V(i,j) = omega1 * V_old(i,j)
&             + ( sou(i,j)    (1)                    (2)
&               + ve(i,j) * V(i+1,j  ) + vw(i,j) * V(i-1,j  )
&               + vn(i,j) * V(i  ,j+1) + vs(i,j) * V(i  ,j-1)
&               ) * vc(i,j)
   end do
end do
```

- Il y a plusieurs dépendances à satisfaire
- Des dépendances vers des valeurs justes calculées

Dépendance dans l'algorithme

```
do j = 1, ny
   do i = 1, nx
      V(i,j) = omega1 * V_old(i,j)
&             + ( sou(i,j)    (1)                    (2)
&               + ve(i,j) * V(i+1,j  ) + vw(i,j) * V(i-1,j  )
&               + vn(i,j) * V(i  ,j+1) + vs(i,j) * V(i  ,j-1)
&               ) * vc(i,j)
   end do
end do
```

- Il y a plusieurs dépendances à satisfaire
- Des dépendances vers des valeurs justes calculées
- Des dépendances vers des valeurs *anciennes*

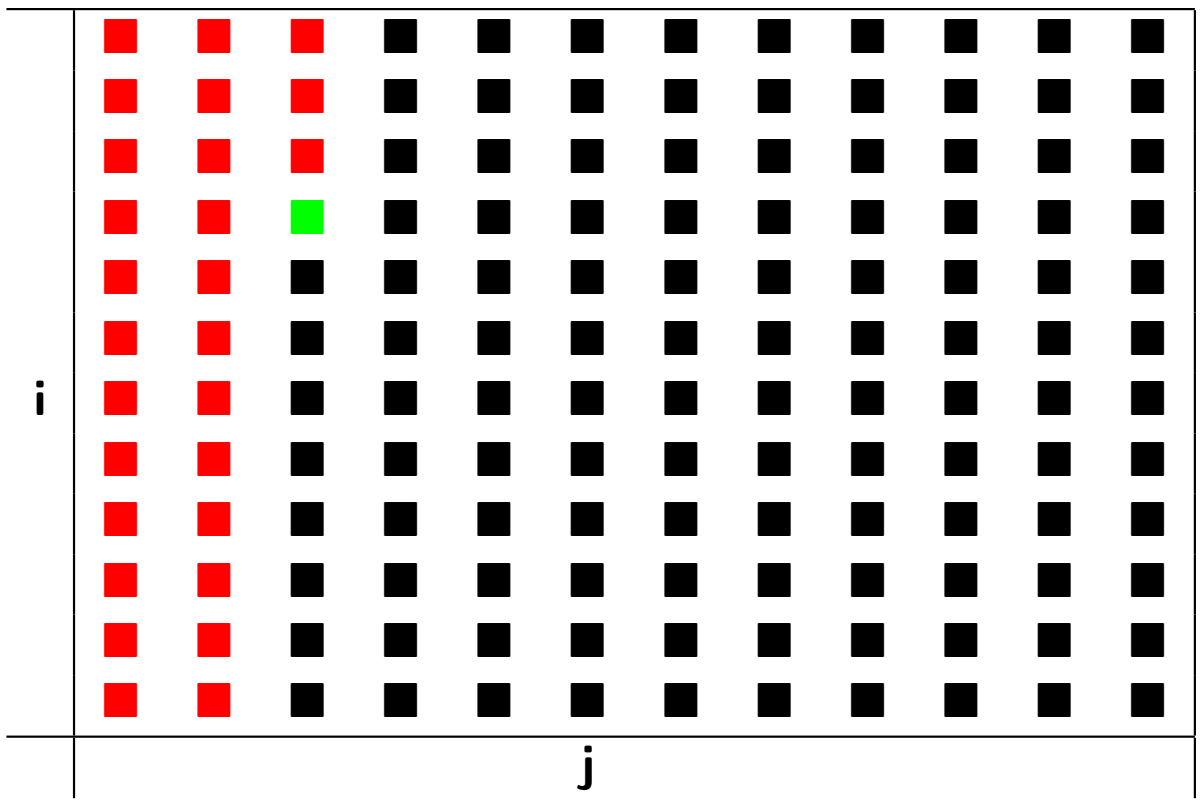
Dépendance dans l'algorithme

```
do j = 1, ny
   do i = 1, nx
      V(i,j) = omega1 * V_old(i,j)
&             + ( sou(i,j)    (1)                    (2)
&               + ve(i,j) * V(i+1,j  ) + vw(i,j) * V(i-1,j  )
&               + vn(i,j) * V(i  ,j+1) + vs(i,j) * V(i  ,j-1)
&               ) * vc(i,j)
   end do
end do
```

- Il y a plusieurs dépendances à satisfaire
- Des dépendances vers des valeurs justes calculées
- Des dépendances vers des valeurs *anciennes*
- Elles doivent toutes être satisfaites, mais on n'a pas les mêmes contraintes

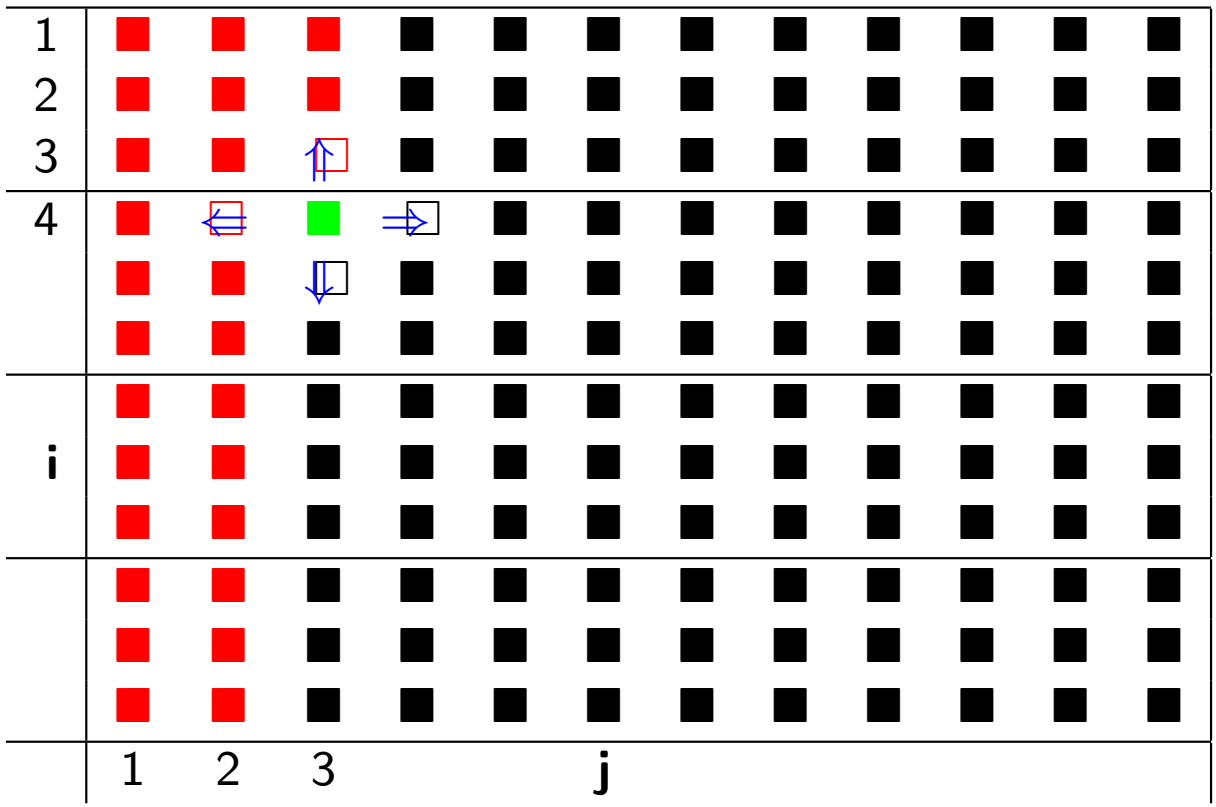
Dépendance

- Dépendance dans le calcul :
  - nouvelles valeurs, itération courante
  - valeur en cours de calcul, elle a besoin de ses voisins immédiats
  - anciennes valeurs, itération précédente



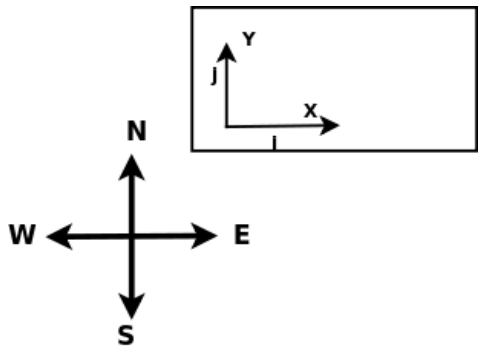
Découpage

- Le découpage se fait selon une topologie 1D cartésienne
- Chaque processus reçoit un rectangle de points à calculer
- Chaque sous-domaine a des voisins ou des bords extérieurs



Variables pour la topologie

```
! Environnement MPI
Integer :: nbproc, myrank, ierr
! Communicateur et types dérivés
Integer :: ip_y, comm1d
! Répartition des processus
Integer, Dimension(1) :: idims
! Coordonnées des processus dans la grille
Integer, Dimension(1) :: icoords
! Périodicité des conditions aux limites
Logical, Dimension(1) :: periods
! Rang des processus voisins
Integer, Dimension(2) :: voisins
Integer, Parameter :: ivw = 1, ive = 2
! Renumérotation des processus ?
Logical :: reorg = .TRUE.
```



Construction de la topologie

```
! Repartition des processus
   idims(1) = 0
   CALL MPI_Dims_Create (nbproc, 1, idims, ierr)
!
! Creation de la grille de processeurs 1D sans periodicite
   periods(1) = .FALSE.
   CALL MPI_Cart_Create (MPI_COMM_WORLD, 1, idims, periods, reorg,
  &                      comm1d, ierr)
!
! Recherche de ses 2 voisins pour chaque processeur
   voisins(1:2) = (/ MPI_PROC_NULL, MPI_PROC_NULL /)
!
! Recherche des voisins Ouest et Est
   CALL MPI_Cart_Shift (comm1d, 0, 1, voisins(ivw), voisins(ive), ierr)
!
! Recherche des coordonnees dans la grille
   CALL MPI_Cart_Get (comm1d, 1, idims, periods, icoords, ierr)
```

Construction des sous-domaines

```
! Determiner les indices de chaque sous-domaine
   imin = 1  +(icoords(1)*nx) / idims(1)
   imax = ((icoords(1)+1)*nx) / idims(1)
!
!Allocation des tableaux
   ALLOCATE (x(imin:imax), xm(imin-1:imax), ym(1-1:ny),
  &          y(1:ny), V(imin-1:imax+1,0:ny+1),
  &          V_old(imin-1:imax+1,0:ny+1),
  &          sou(imin:imax,1:ny), vn(imin:imax,1:ny),
  &          ve (imin:imax,1:ny), vs(imin:imax,1:ny),
  &          vw (imin:imax,1:ny), vc(imin:imax,1:ny),
  &          eps(imin-1:imax+1,0:ny+1), STAT=ierr)
   if (ierr /= 0) then
      Write (6,'(A)') 'ERREUR ALLOCATION DYNAMIQUE'
      CALL MPI_Abort (MPI_COMM_WORLD, -1, ierr)
      stop
   end if
```

Traitement des dépendances

```
   do j = 1, ny
! Réception bloquante point interface bord ouest (2)
      do i = imin, imax
         V(i,j) = omega1 * V_old(i,j)
   &              + ( sou(i,j)    (1)                        (2)
   &                + ve(i,j) * V(i+1,j  ) + vw(i,j) * V(i-1,j  )
   &                + vn(i,j) * V(i  ,j+1) + vs(i,j) * V(i  ,j-1)
   &                ) * vc(i,j)
      end do
! Envoi non bloquant interface point bord est
   end do
! Réception non bloquante interface bord est (1)
! Envoi interface bord ouest
```

Construction des types dérivés

- Bloc de données équidistantes : `V(i, 1:ny)`

```
 CALL MPI_Type_Vector (ny-1+1,          ! nombre de blocs
&                      1,                    ! longueur d'un bloc
&                      imax-imin+3,          ! pas entre le debut de
&                      MPI_DOUBLE_PRECISION,! deux blocs
                                             ! consecutifs
&                      ip_y, ierr)           ! nouveau type
 CALL MPI_Type_Commit (ip_y, ierr)           ! validation
```

Boucle complète (1/5)

```
   do k = 1, iter_max
! Attente reception bord est, envoi bord ouest
      if (k /= 1) then
         CALL MPI_Wait (ireq1, status, ierr)
         CALL MPI_Wait (ireq2, status, ierr)
      end if
!
   do j = 1, ny
!
! Reception bloquante point interface bord ouest (2)
      CALL MPI_Recv (V(imin-1,j), 1, MPI_DOUBLE_PRECISION,
  &                  voisins(ivw), j, comm1d, status, ierr)
!
! Validation envoi point bord est
      if (k /= 1) Call MPI_Wait (ireq3(j), status, ierr)
```

Boucle complète (2/5)

```
      do i = imin, imax
         V(i,j) = omega1 * V_old(i,j)
     &           + ( sou(i,j)    (1)                    (2)
     &             + ve(i,j) * V(i+1,j  ) + vw(i,j) * V(i-1,j  )
     &             + vn(i,j) * V(i  ,j+1) + vs(i,j) * V(i  ,j-1)
     &             ) * vc(i,j)
      end do
!
! Envoi interface point bord est
      CALL MPI_ISend (V(imax,j), 1, MPI_DOUBLE_PRECISION,
    &                 voisins(ive), j, comm1d, ireq3(j), ierr)
     end do
!
! Réception non bloquante interface bord est (1)
     irecvtag = 3000+myrank
     CALL MPI_IRecv (V(imax+1,1), 1, ip_y, voisins(ive), irecvtag,
    &                comm1d, ireq1, ierr)
```

Boucle complète (3/5)

```
! Envoi interface bord ouest
   isendtag = 3000+voisins(ivw)
   CALL MPI_ISend (V(imin  ,1), 1, ip_y, voisins(ivw), isendtag,
  &                comm1d, ireq2, ierr)
!
   end do ! k = 1, iter_max
!
! calcul du critere de convergence
   erreur_loc = 0.0d0
   do j = 1, ny
      do i = imin, imax
         erreur = abs (V(i,j) - V_old(i,j) )
         erreur_loc = max (erreur_loc, erreur)
      end do
   end do
!
   CALL MPI_AllReduce (erreur_loc, erreur_max, 1, MPI_DOUBLE_PRECISION,
  &                    MPI_MAX, comm1d, ierr)
```

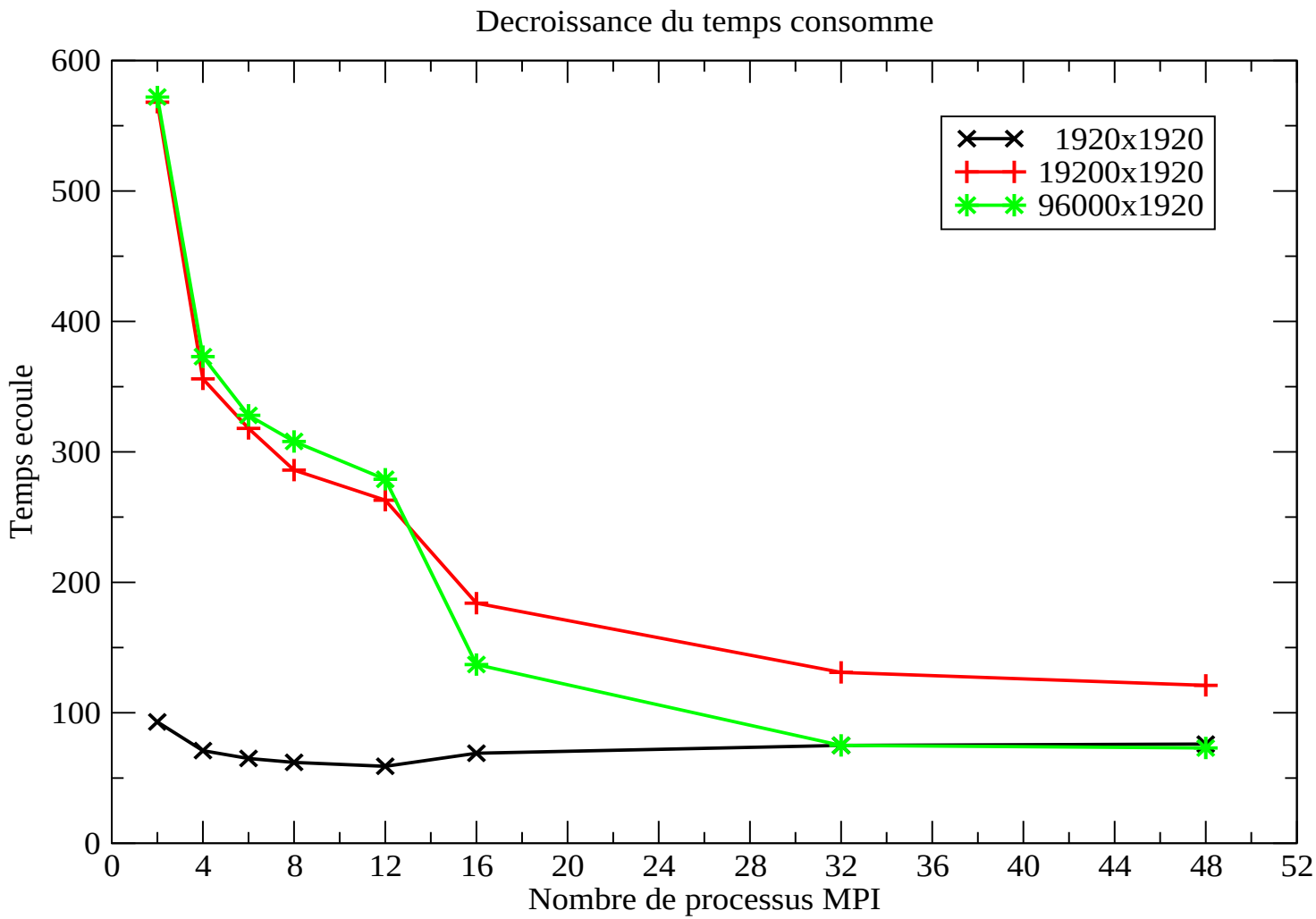
Boucle complète (4/5)

```
! critere d'arret
   if (erreur_max < erreur_tol) then
      if (myrank == 0) write (6,*) 'erreur_max ', erreur_max, ' it ', k
      go to 10
   end if
!
   if ( MOD(k, 100) == 0) then
      if (myrank == 0) write (6,*) 'k = ', k, ' erreur = ', erreur_max
   end if
!
end do ! k = 1, iter_max
!
   if (myrank == 0) write (6,*) 'erreur_max finale : ', erreur_max
!
   10   continue
```

Boucle complète (5/5)

```
   CALL MPI_Wait (ireq1, status, ierr)
   CALL MPI_Wait (ireq2, status, ierr)
!
   if ( voisins(ive) /= MPI_PROC_NULL) then
      do j = 1, ny
         call MPI_Wait (ireq3(j), status, ierr)
      end do
   end if
```

Quelques résultats ...



Quelques résultats ...