Bureau d'étude de calcul distribué

Résolution de systèmes linéaires bloc tridiagonaux par la méthode itérative de Jacobi par blocs. Implantation d'une version parallèle sur réseau de stations.

1 Rappels Numériques

On cherche à résoudre un système linéaire $\mathbf{Ax} = \mathbf{b}$, où $\mathbf{A}$ est une matrice carrée creuse d'ordre N, et où $\mathbf{x}$ et $\mathbf{b}$ sont des vecteurs d'ordre N. La matrice $\mathbf{A}$ est non-symétrique, tri-diagonale par blocs et composée de $Nprocs$ blocs de taille M (voir Figure 1 pour $Nprocs = 4$). La matrice sera aussi supposée à diagonale strictement dominante. Le découpage en bloc de taille M de la matrice induira un découpage sur les vecteurs $\mathbf{x}$, $\mathbf{b}$ et $\mathbf{r}$ ($\mathbf{r} = \mathbf{b} - \mathbf{Ax}$). Par exemple, x_k correspond au k^{ieme} bloc de taille M du vecteur $\mathbf{x}$ (c.à.d $\mathbf{x}_k = \mathbf{x}((k-1) * M + 1 : k * M)$ avec les notations vectorielles du langage FORTRAN).

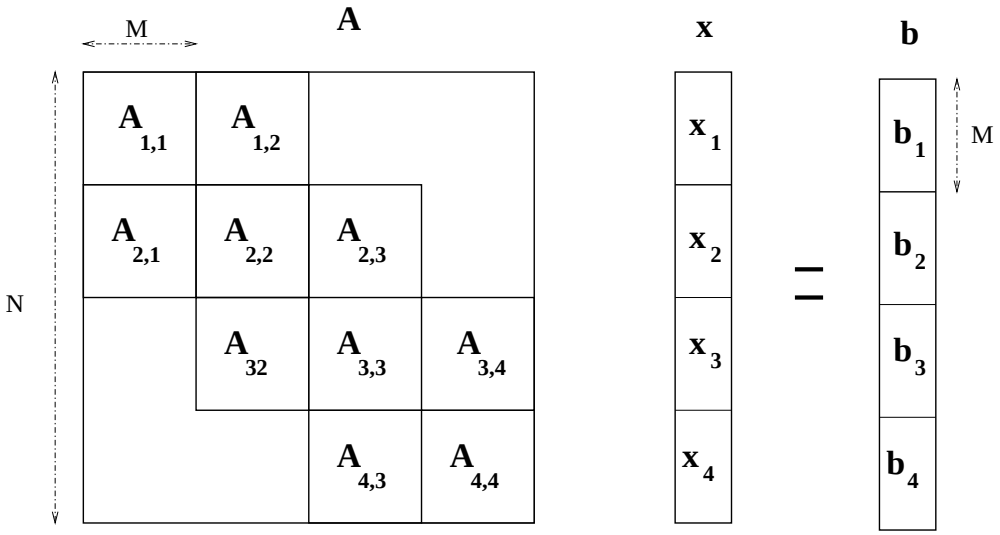


Figure 1: Description des structures de données et de leur découpage ($Nprocs = 4$)

On utilise la méthode itérative de Jacobi par blocs. On cherche donc une suite $\{\mathbf{x}^{(Iter)}\}$ de vecteurs d'ordre N convergeant vers la solution du système linéaire quel que soit le vecteur initial $\mathbf{x}^{(0)}$.
La méthode de Jacobi par blocs se déduit naturellement de la méthode de Jacobi. Avec la méthode de Jacobi par bloc le vecteur $\mathbf{x}^{(Iter+1)}$ se déduit du vecteur $x^{(Iter)}$ par la relation simple suivante:

$$\mathbf{x}^{(Iter+1)} = \mathbf{x}^{(Iter)} + \mathbf{D}^{-1} * (\mathbf{b} - \mathbf{A} * \mathbf{x}^{(Iter)}),$$

où $\mathbf{D}$ désigne la matrice bloc-diagonale constituée des blocs diagonaux $\mathbf{A}_{kk}$ de la matrice $\mathbf{A}$. La méthode de Jacobi par bloc converge en particulier pour des matrices à diagonale strictement dominante.

2 Algorithme séquentiel de Jacobi par blocs

Soient $\mathbf{x}^{(0)} = 0$ le vecteur solution initial, $\mathbf{x}$ l'itéré courant, EPS la précision souhaitée, et $Iter$ le numéro de l'itération courante (on limite ici le nombre maximum d'itérations à $Maxiter$). $\mathbf{y}$ sera un vecteur réel de travail de taille M. On utilisera la norme max pour le test de convergence

$$\|\mathbf{r}\| = \max_{i=1,N} |\mathbf{r}_i|$$

.

Algorithme séquentiel de Jacobi par blocs

Début

- Initialisations : $Iter = 0 \quad \mathbf{x} = 0; \quad \mathbf{r} = \mathbf{b}$
- Factorisation de blocs diagonaux :
  Pour $k = 1$ à $Nprocs$ **Faire**
  Copie de $\mathbf{A}_{kk}$ dans $\mathbf{D}_k$
  Factorisation **LU** de $\mathbf{D}_k$ ($\mathbf{D}_k = \mathbf{P}_k * \mathbf{L}_k * \mathbf{U}_k$)
  Fin Pour
- **Tant Que** ($Iter < Maxiter$) et ($\|\mathbf{r}\| >$ EPS) **Faire**
  - Calcul du nouvel itéré :
    Pour $k = 1$ à $Nprocs$ **Faire**
    Calcul de $\mathbf{y}$ / $\mathbf{P}_k * \mathbf{L}_k * \mathbf{U}_k \ \mathbf{y} = \mathbf{r}_k$ (descente-remontée)
    $\mathbf{x}_k = \mathbf{x}_k + \mathbf{y}$
    Fin Pour
  - Mise à jour du résidu : $\mathbf{r} = \mathbf{b} - \mathbf{A} * \mathbf{x}$
  - Incrémenter $Iter$
- **Fin Tant Que**
- Si le nombre d'itérations est inférieur à $Maxiter$ alors $\mathbf{x}$ contient une approximation, à la précision demandée, de la solution du système.

Fin BlocJacobi séquentiel

3 Spécification de la version parallèle de Jacobi par blocs

3.1 Remarques

1. On notera que les itérations des deux boucles sur k de l'algorithme séquentiel sont indépendantes (chaque processus peut travailler en parallèle sur une instance de la boucle).
2. De plus, si l'on exploite la structure tri-diagonale par bloc de la matrice $\mathbf{A}$ alors
   la Mise a jour du résidu peut s'écrire sous la forme (on suppose ici que $Nprocs \geq 2$)
   $\mathbf{r}_1 = \mathbf{b}_1 - \mathbf{A}_{1,1}\,\mathbf{x}_1 - \mathbf{A}_{1,2}\,\mathbf{x}_2$
   Pour $k = 2$ à $Nprocs - 1$ **Faire**
   $\mathbf{r}_k = \mathbf{b}_k - \mathbf{A}_{k,k-1}\,\mathbf{x}_{k-1} - \mathbf{A}_{k,k}\,\mathbf{x}_k - \mathbf{A}_{k,k+1}\,\mathbf{x}_{k+1}$
   Fin Pour
   $\mathbf{r}_{Nprocs} = \mathbf{b}_{Nprocs} - \mathbf{A}_{Nprocs,Nprocs-1}\,\mathbf{x}_{Nprocs-1} - \mathbf{A}_{Nprocs,Nprocs}\,\mathbf{x}_{Nprocs}$
   Si on suppose que chaque processus, k, possède le k^{ieme} bloc de lignes de $\mathbf{A}$ alors il suffit qu'il possède aussi la valeur de la solution de ses voisins pour que le mise à jour de de $\mathbf{r}_k$ puisse être paralléliser.

3.2 Structures de données distribuées

1. Chaque esclave k possédera uniquement la partie non-nulle du k^{ieme} bloc de lignes de la matrice $\mathbf{A}$ (matrice ayant M lignes et au maximum de $3 * M$ colonnes)
2. Une copie du k^{ieme} bloc diagonal de la matrice sera rangée dans une matrice $\mathbf{D}$ locale à chaque processus.
3. Un vecteur local x de taille maximum $3 * M$ permettra de ranger à chaque étape la solution locale ainsi que les solutions locales des voisins concernés par la mise à jour du $\mathbf{r}_k$ local.

3.3 Propriétés de l'algorithme parallèle

1. Seul le processus maître possède initialement la matrice tri-diagonale $\mathbf{A}$ et le second membre $\mathbf{b}$. Le maître ne participera pas au calcul et se contentera de distribuer les données et d'assembler la solution finale $\mathbf{x}$.
2. La matrice $\mathbf{A}$ est d'ordre $N = Nprocs * M$ où $Nprocs$ désigne le nombre de processus PVM esclaves crées par le processus maître. Le maître distribuera $\mathbf{A}$ (et $\mathbf{b}$) afin que l'esclave k possède le k^{ieme} bloc ligne de $\mathbf{A}$ (et $\mathbf{b}$).
3. Au cours des itérations chaque processus esclave k ne connaîtra que (et ne pourra communiquer qu'avec) ses processus voisins. (Avec la convention que le voisin de droite de k a pour numéro $k+1$ (ou 1 si $k = Nprocs$) et que le voisin de gauche a pour numéro $k-1$ (ou $Nprocs$ si $k = 1$).)
4. L'esclave k est en charge du calcul de $\mathbf{x}_k$. A chaque étape de l'algorithme itératif il met à jour $\mathbf{x}_k$. Après un échange entre processus voisins des morceaux de solutions mises à jour, il calculera un résidu local $\mathbf{r}_k$ (k^{ieme} bloc ligne de $\mathbf{r}$) et sa norme.
5. Comme pour l'algorithme séquentiel, on utilisera aussi la norme max et le même critère de convergence. La norme du résidu global $\mathbf{r}$ devra être connue de chacun des processus esclaves. Un algorithme de type circulation de jeton sera utilisé pour calculer la norme maximum globale de $\mathbf{r}$ à partir des normes locales des $\mathbf{r}_k$. Le ou les jetons ne pourront circuler que de proche en proche (et de gauche à droite) et ceci jusqu'à ce que tous les processus aient connaissance de la norme globale du résidu.

4 Travail algorithmique (sur papier, 45mn maximum)

Écrire l'algorithme **détaillé** des processus esclaves. On s'attachera à

- respecter TOUTES les spécifications de l'algorithme (voir section 3: remarques, structures de données et propriétés),
- décrire l'algorithme retenu pour le calcul de $\mathbf{r}_k$ (on indiquera notamment le nombre de messages envoyés).
- décrire de façon précise les traitements spécifiques liés à des valeurs particulières de $Nprocs$ (par exemple $Nprocs = 1$) et du numéro du processus esclave (par exemple $k = 1$ ou $Nprocs$)

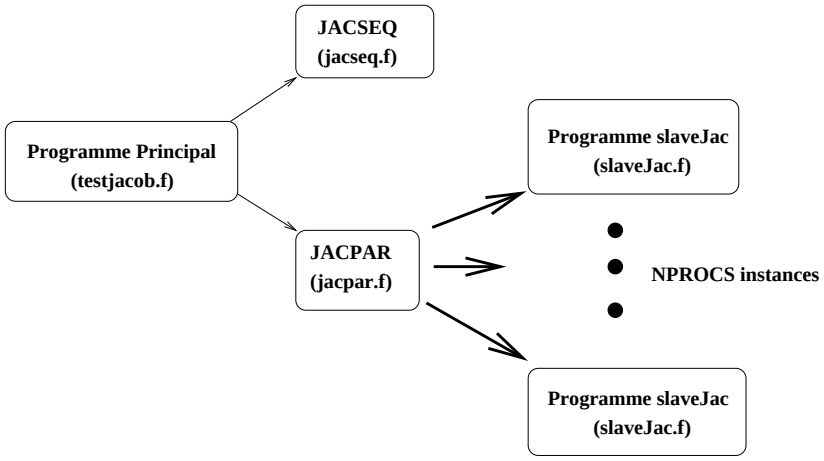
5 Travail sur machine

Le code correspondant à l'implantation de l'algorithme séquentiel, le code du maître de l'algorithme parallèle et le programme principal permettant de tester/comparer les deux algorithmes sont fournis. L'ensemble des codes est décrit dans la section suivante.

1. Écrire le programme associé aux processus esclaves en respectant les spécifications et la description de l'algorithme.
2. Le code du maître est fourni et ne devra pas être modifié. Ceci créera des contraintes au niveau du code de l'esclave qu'il faudra absolument respecter.
3. Envoyer à amestoy@enseeiht.fr un message contenant
   - un résumé synthétique des résultats de test de validation.
   - le fichier **slaveJac.f** (et uniquement ce fichier car les autres ne doivent pas être modifiés)

6 Description des fichiers fournis

L'arbre d'appel des étapes principales est indiqué ci-après:



- **tesjacob.f** C'est le programme principal, il lit les données fournies par l'utilisateur, alloue et initialise les données, lance le calcul séquentiel (**JACSEQ**), puis le calcul parallèle (**JACPAR**). La solution calculée par chacun des codes est vérifiée et comparée à la solution exacte du système.
- **jacseq.f** (subroutine JACSEQ, fournie en Annexe).
  Calcul de la solution du système en utilisant l'algorithme bloc-Jacobi séquentiel (Algorithme 1). *C'est le code de référence et il ne faut pas le modifier.*
- **jacpar.f** (subroutine JACPAR, fournie en Annexe).
  Calcul de la solution du système en utilisant l'algorithme de bloc-Jacobi parallèle. C'est le code du maître. Il est en charge de la création des processus esclaves, de la distribution de la matrice initiale et de la réception des résultats. *Ce code est fourni et ne devra en aucun cas être modifié.*
- **slaveJac.f** (programme à compléter fourni en Annexe). C'est le programme principal exécuté par les processus esclaves. Ce code doit être écrit en respectant les contraintes imposées par le code jacpar.f.
- **util.f**
  Ces fichiers contiennent des subroutines/fonctions utilitaires (calcul de la norme, du résidu, de l'erreur numérique, copie d'un bloc de matrice ..). *A ne pas modifier.*

- **lapack.f**
  Ce fichier contient les codes utiles pour effectuer la factorisation ($\mathbf{A} = \mathbf{PLU}$) et la résolution ($\mathbf{PLU\ x} = \mathbf{b}$) de systèmes linéaires pleins. Le code **DGETRF** permet d'effectuer la factorisation d'une matrice et le code **DGRTRS** permet de résoudre un système linéaire. *A ne pas modifier.*
  Exemple d'utilisation sur une matrice **D** carrée d'ordre M allouée dynamiquement.

```
      DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE:: D, y, b
      INTEGER, DIMENSION(:), ALLOCATABLE:: IPIV
      INTEGER M, Ierr
      M = 153
      ALLOCATE( D(M,M), IPIV(M), y(M), b(M), stat=ierr)
      IF (ierr.GT.0) THEN
        write(6,*) ' Pas assez d espace memoire '
        stop
      END IF
C      --- Initialiser  la matrice
       D =   CALL InitMatrice( D, M, M)
C      ---------------------------------------------
C      factorisation LU de la matrice D (D = P L U)
C      ---------------------------------------------
       CALL DGETRF ( M, M, D, M, IPIV, INFO )
C         -- En sortie de dgetrf :
C         D(1:M,1:M) contient les matrices L et U
C         IPIV(1:M)  contient le vecteur de permutation
C                    associe a la matrice P
C      ---------------------------------------------
C      Resolution : calcul de y tel que  D y = b
C                              (P L U y = b)
C      ---------------------------------------------
       y(1:M) = b(1:M)
C      -- En entree y contient le second membre b
       CALL DGETRS ( 'N', M, 1, D, M,
     &                    IPIV, y, M, INFO )
C      -- En sortie y contient la solution de D y = b
```

7 Annexe : Codes principaux fournis

Code de la version séquentielle de BlocJacobi (fichier jacseq.f)

```
      SUBROUTINE JACSEQ (A, X, B, N, M, Nprocs, EPS, Maxiter )
      IMPLICIT NONE
C ----------------------------------------------------------------
C -Resolution du systeme A.X=B (de dimension NxN ) par
C  la methode iterative de Jacobi par blocs.
C -Soit A une matrice carree d'ordre N
C      tridiagonale par blocs composee
C      de Nprocs blocs diagonaux de taille M (N = Nprocs*M)
C
C -EPS: definit la precision sur la solution recherchee.
C
C -Algorithme utilise: Version sequentielle  de Jacobi par bloc
C ----------------------------------------------------------------
C ----------
C Parametres:
C ----------
C    Input  : A, B, N, M, Nprocs, EPS
C    Output : X
       INTEGER N, M, Nprocs, Maxiter
       DOUBLE PRECISION A(N,N),B(N),X(N),EPS
C ------------------
C Variables locales
C ---------------------------------
C R(N): Vecteur residu a chaque iteration
C Y(M): Vecteur iterant supplementaire
C
```

```
C Pour la factorization LU des matrices Akk (Nprocs blocs diagonaux de A)
C    Akk = Pk*Lk*Uk , ou
C      Pk est une matrice de permutation
C      Lk et Uk sont respectivement des matrices triangulaires
C                  inferieures et superieures.
C   on introduit:
C     - D(M,M,Nprocs):  Matrice des blocs diagonaux factorises
C           D(1:M,1:M,k) (notee Dk) contiendra
C           - dans un premier temps un copie de Akk,
C           - puis Lk Uk associes a la factorisation de Akk
C     - IPIV (M, Nprocs) :
C              IPIV(1..M, k) vecteur de permutations
C              correspondant a une matrice de permutation Pk
C              et construit durant la factorization LU  de Akk
C Err : Norme max du residu
C Iter: Compteur d'iterations
C --------------------------------
       DOUBLE PRECISION R(N), Y(M), D (M, M, Nprocs), Err
       INTEGER IPIV(M,Nprocs)
C      -- Fonction de calcul de la norme max
       EXTERNAL NORM
       DOUBLE PRECISION NORM
       INTEGER I, K, Iter, INFO
C ----------------------------------------
C Initialisations de
C     Iter = compteur d'iterations (K=0)
C     X    = vecteur iterant (X(i)=0, i=1,N)
C     R    = vecteur residu  (R(i)=B(i), i=1,N)
C     Err  = || R ||
C ----------------------------------------
       Iter=0
       X   = DFLOAT(0)
       R   = B
C      -- Calcul de la premiere norme de residu
       Err=NORM(R,N)
       WRITE(*,*)'iteration ',Iter,' norme du residu ',Err
C ----------------------------------------------
C Copie et
C Factorisation LU des Nprocs blocs diagonaux Aj
C ----------------------------------------------
       DO k=1, Nprocs
C       -- copy de Akk dans Dk
        CALL COPYDIAG (A((k-1)*M+1, (k-1)*M+1), D(1,1,k), N, M)
C       -- factorisation LU de Dk  (Dk = Pk*Lk*Uk)
        CALL DGETRF ( M, M, D(1,1,k), M, IPIV(1,k), INFO )
C       -- Dk contient les matrices Lk et Uk.
C       -- IPIV (1:N,k) est le vecteur de permutation associe a Pk
       ENDDO
C -----------------------
C Boucle de convergence
C -----------------------
       DO WHILE ((Err.GT.EPS).AND.(Iter.LE.Maxiter))
C        --------------------------
C        Boucle sur les blocs de A
C        --------------------------
         DO k =1, Nprocs
C          -- calcul de Y tel que Dk Y = Rk
C               ou Rk = R((k-1)*M+1, k*M))
C               (en entree de DGETRS Y contient Rk)
           Y( 1 : M ) = R( (k-1)*M+1 : k*M )
           CALL DGETRS ( 'N', M, 1, D(1,1,k), M,
     &                   IPIV(1,k), Y, M, INFO )
C               (en sortie de DGETRS Y contient la solution de
C                  Dk Y = Rk )
C          -- Calcul du nouvel iterant
           X( (k-1)*M+1 : k*M ) = X( (k-1)*M+1 : k*M ) + Y( 1 : M )
         END DO
C        ---------------------------------------
C        Calcul du nouveau residu et de sa norme
C        ---------------------------------------
         CALL RESIDU (A, X, B, R, N, N)
```

```
        Err = NORM(R,N)
        Iter=Iter+1
        WRITE(*,*)'iteration ',Iter,' norme du residu ',Err
      END DO
      RETURN
      END
```

Code du maître de l'algorithme parallèle (fichier jacpar.f)

```
      SUBROUTINE JACPAR (A, X, B, N, M, Nprocs, EPS, Maxiter )
      IMPLICIT NONE
C -------------------------------------------------------
C -Resolution du systeme A.X=B (de dimension NxN ) par
C  la methode iterative de Jacobi par blocs.
C -Soit A une matrice carree d'ordre N
C      tridiagonale par blocs composee
C      de Nprocs blocs diagonaux de taille M (N = Nprocs*M)
C -EPS       definit la precision sur la solution recherchee.
C -Maxiter   definit le nombre maximum d'iterations
C -Algorithme utilise: Version parallele  de Jacobi par bloc
C    Le processus maitre effectue:
C      - Creation des processus esclaves PVM,
C      - Distribution des donnees
C      - Collecte des resultats en provenance des esclaves
C        (morceaux de solution X de Ax=B)
C    Chaque processus esclave est en charge du calcul
C    d'un bloc de la solution x de taille M, et possede
C    un bloc de taille M de lignes de la matrice.
C -------------------------------------------------------
C ----------
C Parametres:
C ----------
C    Entree (non modifies en sortie) :
C         A(N,N), B(N), N, M, Nprocs, EPS
C    Sortie (ne devant pas etre initialise en entree) :
C         X(N)
      INTEGER N, M, Nprocs, Maxiter
      DOUBLE PRECISION A(N,N),B(N),X(N),EPS
C -------------------
C Variables Locales
C -------------------
*     Le bloc de la matrice a envoyer
*     a chaque esclave est caracterise  par:
*         Ideb: indice (de ligne) de debut de A
*         M   : nombre de lignes (consant)
*         Jdeb: indice (de colonne) de debut de A
*         Jfin: indice (de colonne) de fin de A
*            (NBCOL = Jfin - Jdeb + 1)
*     Prec et Suiv correspondent aux numero logiques
*          des processus suivants et precedants (modulo Nprocs)
*     NoSlave : numero logique de l'esclave
*     Iter    : nombre d'iterations des esclaves
*
      INTEGER I, J, K, NoSlave, Ideb, Jdeb, Jfin, Suiv, Prec,
     &        NBCOL, Iter
      CHARACTER*32 HFILE, MACHI(Nprocs)
*
*      -- Variables PVM
      INTEGER my_id,  tids(Nprocs), info, type, bufidR, bufidS,
     &        FLAG, numt
      include '/users/commun/PVM_LIB/pvm-3.3.6/include/fpvm3.h'
* -------------------------
* S'enregister sous PVM
* -------------------------
      call pvmfmytid(my_id)
      call enrolexpvm()
      if (my_id .lt. 0) then
         write(*,*) ' failure in enrolling on host'
         stop
      endif
* -------------------------
```

```
* Creer Nprocs esclaves
* -------------------------
      IF (Nprocs.LE.0) RETURN
      IF (Nprocs.GE.1) THEN
       WRITE(6,*) 'Nom du fichier host:'
       read(5,*) HFILE
       open(unit =10, file =HFILE, STATUS='OLD',IOSTAT=info)
       IF (info.GT.0) THEN
        write(6,*) ' ** Erreur ouverture ',HFILE,'n existe pas??'
        RETURN
       ENDIF
       DO I=1,Nprocs
        call read_file(MACHI(I))
       ENDDO
       rewind(10)
       close (10)
      ENDIF
      numt = 0
      FLAG=PVMTASKHOST
      DO i=1, Nprocs
           call pvmfspawn( 'slaveJac',FLAG, MACHI(i), 1,
     &     tids(i),info)
           numt = numt + info
           if (info.eq.1) then
            write(6,*) ' ACTIVATION PVM proc.: ',I,' pvmtid:',
     &                 tids(i), ' on machine:', MACHI(i)
           else
            write(6,*) ' failure spawning process no: ',i,
     *       ' on machine:', MACHI(i)
            write(6,*) ' Error code is ', tids(i)
           endif
      ENDDO
      if (numt .ne. Nprocs) then
         write(*,*) ' failure in spawning one slave '
         stop
      endif
C
* -------------------------------
* Distribution des donnees
* -------------------------------
      write(6,*) ' -- Maitre : DEBUT de la distribution des donnees '
      type = 0
      DO NoSlave = 1, Nprocs
*       -- calcul de Ideb, Jdeb, Jfin, Prec, Suiv
        Ideb = (NoSlave-1)*M + 1
        IF (NoSlave.EQ.1) THEN
          Jdeb = 1
          Prec = Nprocs
        ELSE
          Jdeb = (NoSlave-2)*M + 1
          Prec =  NoSlave-1
        END IF
        IF (NoSlave.EQ.Nprocs) THEN
          Jfin = N
          Suiv = 1
        ELSE
          Jfin = (NoSlave+1)*M
          Suiv = (NoSlave+1)
        END IF
        write(6,*) ' Slave no', NoSlave,
     &             ' No du proc. precedent/suivant : ', Prec, Suiv
        write(6,*) ' Slave no', NoSlave,
     &            ' Envoi du bloc de A : Ideb,Jdeb,Jfin, NBCOL =',
     &               Ideb,Jdeb,Jfin, NBCOL
*       ---------------------------------
*       Initialiser/Remplir/envoyer buffer
*       ---------------------------------
        call pvmfinitsend( PVMDATADEFAULT, bufidS)
*
        call pvmfpack(INTEGER4, Nprocs, 1, 1, info)
        call pvmfpack(INTEGER4, NoSlave, 1, 1, info)
```

```
        call pvmfpack(INTEGER4, M, 1, 1, info)
        Call pvmfpack(INTEGER4, tids(Prec), 1, 1, info)
        call pvmfpack(INTEGER4, tids(Suiv), 1, 1, info)
*       -- Ranger le bloc de lignes dans le buffer
        NBCOL = Jfin-Jdeb+1
        call pvmfpack(INTEGER4, NBCOL, 1, 1, info)
        call pvmfpack(INTEGER4, Maxiter, 1, 1, info)
        call pvmfpack(REAL8, EPS, 1, 1, info)
        DO J= Jdeb, Jfin
         call pvmfpack(REAL8, A(Ideb,J), M, 1, info)
        END DO
*       -- Ranger du B(Ideb:Ideb+M-1) dans le buffer
        call pvmfpack(REAL8, B(Ideb), M , 1, info)
*
*       -- Envoi du contenu du buffer a l'esclave NoSlave
        call pvmfsend(tids(NoSlave), type, info)
*
      END DO
      write(6,*) ' -- Maitre : FIN de la distribution des donnees '
* -----------------------
* Reception des resultats
* -----------------------
      write(6,*) ' -- Maitre : DEBUT de reception resultats '
      type = 4
      do K = 1, Nprocs
*       - reception dans un ordre quelconque
        call pvmfrecv(-1, type, bufidR)
        call pvmfunpack(INTEGER4, NoSlave, 1, 1, info)
        call pvmfunpack(INTEGER4, Iter, 1, 1, info)
        write(*,*) ' Reception des resultats de ', NoSlave,
     &    ' Nb d''iterations = ', Iter
        call pvmfunpack(REAL8,X((NoSlave-1)*M+1),M,1,info)
      enddo
      write(6,*) ' -- Maitre : FIN de reception resultats '
* ------------
* Quitter PVM
* ------------
      call pvmfexit(info)
      RETURN
      END
```

Code à completer de l'esclave de l'algorithme parallèle (fichier slaveJac.f)

```
      PROGRAM slaveJac
      IMPLICIT NONE
C-------------------------------------------------------------------------
C Code de l'esclave associe a la Version parallele  de Jacobi par bloc
C  Chaque processus esclave est en charge du calcul
C   d'un bloc de la solution x de taille m, et possede
C   un bloc de taille m de lignes de la matrice.
C EPS       definit la precision sur la solution recherchee.
C           par rapport a la norme max du residu global
C Maxiter   definit le nombre maximum d'iterations
C-------------------------------------------------------------------------
      INTEGER M, Maxiter, NBCOL, Nprocs
      INTEGER, DIMENSION(:), ALLOCATABLE          :: IPIV
      DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE:: A, D
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE  :: B, X, R, Y
C --------------------------------
C A(M,NBCOL) Bloc ligne de taille m de la matrice initiale
C R(M): Vecteur residu a chaque iteration
C Y(M): Vecteur iterant supplementaire
C D(M,M):  Matrice des blocs diagonaux factorises
C              D(1:m,1:m) contiendra la factorization LU
C              du kieme bloc diagonal de la matrice A
C IPIV (M) :
C              IPIV(1..M) vecteur de permutations
C                           associe a la factorization LU
C                           du kieme bloc diagonal de A
C Err : Norme max du residu
C Iter: Compteur d'iterations
```

```
C ---------------------------------
      DOUBLE PRECISION Err, Errtemp, EPS
      INTEGER I, J, NoSlave, Iter, INFO, tidPrec, tidSuiv,
     &        ierr, Jdiag, my_id, p_id, type, bufidR,  bufidS
      EXTERNAL NORM
      DOUBLE PRECISION NORM
      include '/users/commun/PVM_LIB/pvm-3.3.6/include/fpvm3.h'
* --------------------------
* S'enregister sous PVM
* --------------------------
      call pvmfmytid(my_id)
      call pvmfparent(p_id)
      if (p_id.le.0) stop
* ----------------------
* Reception des donnees
* ----------------------
      tidPrec = 0
      tidSuiv = 0
      type = 0
      call pvmfrecv(p_id, type, bufidR)
C     -- extraire la taille des donnees
      call pvmfunpack(INTEGER4, Nprocs, 1, 1, info)
      call pvmfunpack(INTEGER4, NoSlave, 1, 1, info)
      call pvmfunpack(INTEGER4, M, 1, 1, info)
      write(6,*) ' Esclave ', NoSlave, ' Nprocs,M =',  Nprocs,M
C
      call pvmfunpack(INTEGER4,tidPrec,1,1,info)
      call pvmfunpack(INTEGER4,tidSuiv,1,1,info)
      write(6,*) NoSlave,' : Mes voisins : gauche ',tidPrec,
     &            ' et droite ',tidSuiv
      call pvmfunpack(INTEGER4, NBCOL, 1, 1, info)
      call pvmfunpack(INTEGER4, Maxiter, 1, 1, info)
      write(6,*) NoSlave,' : Matrice locale ', M, ' Lignes',
     &             NBCOL, ' Col'
C     ------- Allocation  des donnees en memoire
      ALLOCATE (A(M,NBCOL), D(M,M), B(M), X(NBCOL),
     &                  IPIV(M), R(M), Y(M), stat=ierr)
      IF (ierr.GT.0) THEN
        write(6,*) ' Testez un pb de plus petite taille'
        write(6,*) ' Pas assez d espace memoire '
        stop
      END IF
C
C     ----- Extraire la sous matrice et le morceau de second membre
      call pvmfunpack(REAL8, EPS, 1, 1, info)
      DO J=1,NBCOL
        call pvmfunpack(REAL8, A(1,J), M, 1, info)
      END DO
      call pvmfunpack(REAL8, B, M, 1, info)

C -----------------------------------------
C Initialisations de
C     Iter = compteur d'iterations
C     Xloc = vecteur iterant (X(i)=0, i=1,NBCOL)
C     R    = vecteur residu  (R(i)=B(i), i=1,M)
C     Err  = || R ||
C -----------------------------------------
       Iter=0
       X(1:NBCOL) = DFLOAT(0)
       R(1:M)     = B(1:M)
C      Calcul de la premiere norme du residu
       Err=NORM(R,M)
C      Affichage de la premiere norme du residu
       WRITE(*,*)'iteration ',Iter,' norme du residu ',Err

C ---------------------------------------------
C Copie et
C Factorisation LU des Nprocs blocs diagonaux Aj
C ---------------------------------------------
*      -- copy du bloc diagonal de A dans D(1:m, 1:m)
       Jdiag = M+1
```

```
       IF (NoSlave.EQ.1) Jdiag =1
       CAll COPYDIAG (A(1, Jdiag), D, M, M)
C      -- factorisation LU du bloc k de D
       CALL DGETRF ( M, M, D, M, IPIV, INFO )

C -----------------------
C Boucle de convergence
C -----------------------
       DO WHILE ((Err.GT.EPS).AND.(Iter.LE.Maxiter))

  Iter=Iter+1
C
C          --- calcul de Y tel que D Y = R
           Y(1:M) = R(1:M)
C          --- en entree de DGETRS Y contient R
           CALL DGETRS ( 'N', M, 1, D, M,
     &                   IPIV, Y, M, INFO )
C          --- Calcul du nouvel iterant
           X (Jdiag : Jdiag+m-1) = X (Jdiag : Jdiag+m-1) + Y (1:M)
C
C          -- envoi de X aux voisins etc..
C          (On notera que tidPrec et tidSuiv sont
C           respectivement les identificateurs de tache PVM des
C           voisins de gauche et de droite)
C
C
C -----------
C A completer
C -----------
C
C        Affichage de la norme du residu
         WRITE(*,*)'iteration ',Iter,' norme du residu ',Err
       END DO
C
C      --- envoi du morceau de solution au maitre
C
C -----------
C A completer
C -----------

       call pvmfexit(info)
       STOP
       END
```