```
MODULE Constants_Model
  IMPLICIT NONE
  !
  ! analysis model dimension: 2-D or 3-D
  ! ModelFlag = 0: 2-D
  ! ModelFlag = 1: 3-D
  INTEGER:: ModelFlag
  PARAMETER( ModelFlag = 0 )
  !
  ! computational zone: free space or periodic boundary condition
  !   BCFlag = 0: free space for y direction
  !   BCFlag = 1: periodic boundary condition for y direction
  INTEGER:: BCFlag
  PARAMETER( BCFlag = 1 )
  !
  ! Computational zone parameter: begin from 0
  ! MaxX, MaxY, MaxZ: the computation zone length
  !      for three directions
  ! N1, N2, N3: the mesh number for three direction
  ! StartEnd(3, 3): the cutoff number for different
  !           zone
  ! CutPoints(3,4): the coordinates for cutoff points
  REAL(KIND=8):: MaxX, MaxY, MaxZ
  PARAMETER( MaxX = 8, MaxY = 4, MaxZ = 0 )
  REAL(KIND=8):: SolidRelativePosi(3) = (/ 3.5d0, 2.0d0, 0.0d0 /)
  INTEGER:: N1, N2, N3, StartEnd(3,3)
  REAL(KIND=8):: CutPoints(3,4)
  REAL(KIND=8):: DeltaT, DeltaXYZ
  !
  ! grid stretching factor( 1.0d0< q0 )
  ! q0(1,:): x direction
  ! q0(2,:): y direction
  ! y0(3,:): z direction
  REAL(KIND=8):: q0(3,2) = RESHAPE( (/ 1.001d0, 1.0d0, 1.001d0, &
  &                          1.001d0, 1.0d0, 1.001d0 /), (/3,2/))
  !
  ! analysis model position and attack angle
  ! LN: the lagrange point number
  ! Centre(3): the position of the model
  ! AOA: Attack angle
  ! LRef(3): reference length for three direction
  INTEGER:: LN
  REAL(KIND=8):: Centre(3) = (/ 0.0d0, 0.0d0 , 0.0d0 /)
  REAL(KIND=8):: AOA
  PARAMETER( AOA = 0.0d0 )
  REAL(KIND=8):: LRef(3)
  INTEGER:: NumBody
  INTEGER, ALLOCATABLE:: MultiLN(:)
  !
  ! total iteration time and CFL number
  REAL(KIND=8):: TotalTime, CFL
  PARAMETER( TotalTime = 50, CFL = 0.5d0 )
```

```
    !
    ! PML grid number and absorbing coefficient
    !   and auxiliary grid: NA
    ! here BeitaFlag = 0 stands for no background flow
    !       BeitaFlag = 1 stands for existing background flow
    INTEGER:: NPML, NA
    INTEGER:: BeitaFlag
    REAL(KIND=8):: AFA, SIGMA0, Beita
    PARAMETER( BeitaFlag = 0 )
    PARAMETER( NPML = 40, NA = 3 )
    PARAMETER( AFA = 2, SIGMA0 = 2 )
    !
    ! for the inner PML zone for solid
    REAL(KIND=8):: R_PML, R_PML0
    REAL(KIND=8):: SolidAfa, SolidSigma0
    PARAMETER( SolidAfa = 1.0d0, SolidSigma0 = 2.0d0 )
    INTEGER:: SolidPMLPosition( 3, 2 )
    !
    ! MPI processor grid
    INTEGER:: NPX, NPY, NPZ
    PARAMETER( NPX = 1, NPY = 1, NPZ = 1 )
    !
    ! numerical probe information for OUTPUT
    ! Here ProbeAngle stands for number of degrees
    !   and should swith into radian
    REAL(KIND=8):: Rp
    PARAMETER( Rp = 4.0d0 )
    !
    INTEGER:: NumProbe(2)
!   PARAMETER( NumProbe = 36 )
    REAL(KIND=8):: PI
    PARAMETER( PI = 3.141592653d0 )
    REAL(KIND=8), ALLOCATABLE:: ProbeCoordinate( :, : )
    INTEGER, ALLOCATABLE:: Corner( :, : )
    REAL(KIND=8), ALLOCATABLE:: ProbeAngle( : )
    REAL(KIND=8), ALLOCATABLE:: SideLength( :, : )
    REAL(KIND=8), ALLOCATABLE:: RelativePosi( :, : )
    REAL(KIND=8), ALLOCATABLE:: ProbePressure( : )
    REAL(KIND=8), ALLOCATABLE:: ProbePressure0( : )
    REAL(KIND=8), ALLOCATABLE:: ProbePressure1( : )
    REAL(KIND=8), ALLOCATABLE:: ProbePressure2( : )
    INTEGER, ALLOCATABLE:: ProbeFlag( : )
    INTEGER, ALLOCATABLE:: ProbeFlagRoot( : )
    INTEGER:: ProbeTimes = 0
END MODULE Constants_Model
!
MODULE AcousticInitialCondition
    !-------------------------------------------------
    ! Initialize acoustic field
    ! ------------------------------------------------
    IMPLICIT NONE
    !-------------------------------------------------
    ! Here we consider the pulse acoustic, so initial
```

```
    !  acoustic pressure is like:
    !  p = A exp ( B * ( (x-x0)^2+(y-y0)^2+(z-z0)^2 ) ) &
    !       &* sin( omega * t + fai )
    !------------------------------------------------
    REAL(KIND=8):: A, B, Omega, Fai, PI
    REAL(KIND=8):: X0, Y0, Z0
    PARAMETER( PI = 3.141592653d0 )
    !
    REAL(KIND=8):: Thita, Rsource
    PARAMETER( Thita = 180.0d0 * PI / 180.0d0, Rsource = 1.5d0 )
    PARAMETER( A = 1.0d0, B = - DLOG( 2.0d0 ) * 50.0d0, Omega = 8.0 * PI, Fai = 0.0d0 )
    PARAMETER( X0 = Rsource * DCOS( Thita ), Y0 = Rsource * DSIN( Thita ), Z0 = 0.0d0 )
    !
    REAL(KIND=8):: VG, KX, KY
    PARAMETER( VG = 1.0d-5, KX = 5.0d0 * PI/ 2.0d0, KY = 5.0d0 * PI/ 2.0d0 )
END MODULE AcousticInitialCondition
!
MODULE BackgroundFlowField
    !---------------------introduction---------------------
    !Ma: the inlet mach number
    !Pt: inlet total pressure
    !ROUt: inlet total density
    !Pb: the background pressure
    !R: constant for air
    !lamda: ratio of specific heat
    !Re: Renould number
    !Miu_Ref: reference viscosity coefficent
    !Pr: Prantl number(0.72)
    !c_ref0: reference acoustic velocity
    !Tt: inlet total temperature
    !T_ref0: reference static temperature(inlet)
    !p_ref0: reference static pressure(inlet)
    !ROU_ref0: reference static density(inlet)
    !----------------------------------------------------
    !----------------------------------------------------
    IMPLICIT NONE
    REAL(KIND=8):: Ma, R, lamda, Re, Miu_Ref, Pr
    REAL(KIND=8):: Pt, ROUt, Tt, T_ref0, p_ref0, ROU_ref0, c_ref0, Pb
    PARAMETER( Ma = 0.0d0, Re = 100.0d0, R = 287.06d0, lamda = 1.4d0 )
    PARAMETER( Pt = 101325.0d0, Tt = 288.0d0, ROUT = Pt / ( R * Tt ) )
    PARAMETER( p_ref0 = Pt / ( 1 + 0.2d0 * Ma**2 )**( 1.4d0/0.4d0 ) )
    PARAMETER( ROU_ref0 = p_ref0 * ( 1 + 0.2d0 * Ma**2 ) / ( R * Tt ) )
    PARAMETER( T_ref0 = p_ref0 / ( ROU_ref0 * R ), c_ref0 = DSQRT( lamda * R * T_ref0 ) )
    PARAMETER( Pb = p_ref0 )
    PARAMETER( Pr = 0.72d0, Miu_Ref = 1.711 * 10.0d0**( -5.0d0 ) * &
    &           ( (T_ref0/273.0d0)**1.5 )*( 273.0d0+122.0d0 )/( T_ref0 + 122.0d0 ) )
    !PARAMETER(Ma=0.80d0, u_out=119d0, Pb=0.0d0)
      !119171.63      197800   47892.4    46704do
    !p0=47892.4d0, ROU0=1.2218d0
    !PARAMETER(p0=47892.4d0, ROU0=1.2218d0)
    !PARAMETER(T0=p0/(ROU0*R), c0=dsqrt(lamda*R*T0), Pt=p0*(1+0.2*Ma**2)**(lamda/(lamda-1)))
    !PARAMETER(Tt=T0*(1+0.2*Ma**2), ROUT=Pt/(R*Tt))
```

```
END MODULE BackgroundFlowField
!
!!
!-----------------------INTRODCUCTION-------------------------------------
!This MODULE is provided by WULONG( FAEL, BUAA, 2015.12.8), And it can get the
!spatial derivative using DRP or one-side DRP scheme by Hu Fangqiang.
!-------------------------------------------------------------------------
MODULE M_Spatial_dis
  IMPLICIT NONE
  REAL(KIND=8):: AA06(0:6) = (/ -2.192280339d0,4.748611401d0,-5.108851915d0, &
  &           4.461567104d0,-2.833498741d0,1.128328861d0,-0.203876371d0 /)
  REAL(KIND=8):: AA15(-1:5)= (/ -0.209337622d0,-1.084875676d0,2.147776050d0, &
  &           -1.388928322d0,0.768949766d0,-0.281814650d0,0.048230454d0 /)
  REAL(KIND=8):: AA24(-2:4)= (/ 0.049041958d0,-0.468840357d0,-0.474760914d0, &
  &           1.273274737d0,-0.518484526d0,0.166138533d0,-0.026369431d0 /)
  REAL(KIND=8):: AA33(-3:3)= (/ -0.02084314277031176d0,0.1667059044145804 69d0, &
  &           -0.7708823805182255 52d0,0.0d0,0.7708823805182255 52d0, &
  &             -0.1667059044145804 69d0,0.02084314277031176d0 /)
  REAL(KIND=8):: AA42(-4:2)= (/ 0.026369431d0,-0.166138533d0,0.518484526d0, &
  &           -1.273274737d0,0.474760914d0,0.468840357d0,-0.049041958d0 /)
  REAL(KIND=8):: AA51(-5:1)= (/ -0.048230454d0,0.281814650d0,-0.768949766d0, &
  &           1.388928322d0,-2.147776050d0,1.084875676d0,0.209337622d0 /)
  REAL(KIND=8):: AA60(-6:0)= (/ 0.203876371d0,-1.128328861d0,2.833498741d0, &
  &           -4.461567104d0,5.108851915d0,-4.748611401d0,2.192280339d0 /)
  !!
  CONTAINS
  !!
  SUBROUTINE DRP7(fx, f, delx, M1, M2, SIZE0 )
  IMPLICIT NONE
  INTEGER:: M1, M2, N1, N2
  INTEGER:: SIZE0(2)
  REAL(KIND=8):: fx( M1 : M2 ), f( SIZE0(1) : SIZE0(2) )
  REAL(KIND=8):: Delx
  INTEGER:: I
  N1 = SIZE0(1)
  N2 = SIZE0(2)
  IF ( ( M1 == N1 ) .AND. ( M2 == N2 ) ) THEN
    DO I = M1, M2
      IF ( I == M1 ) THEN
        fx(i) = 1.d0 / delx * ( AA06(0) * f(i) + AA06(1) * f( i+1 ) + &
  &         AA06(2) * f( i+2 ) + aa06(3) * f( i+3 ) + aa06(4) * f( i+4 ) &
  &           + aa06(5) * f( i+5 ) + aa06(6) * f( i+6 ) )
      ELSEIF ( I == M1+1 ) THEN
        fx(i) = 1.d0 / delx * ( aa15(-1) * f( i-1 ) + aa15(0) * f(i) + &
  &         aa15(1) * f( i+1 ) + aa15(2) * f( i+2 ) + aa15(3) * f( i+3 ) &
  &           + aa15(4) * f( i+4 ) + aa15(5) * f( i+5 ) )
      ELSEIF ( I == M1+2 ) THEN
        fx(i) = 1.d0 / delx * ( aa24(-2) * f( i-2 ) + aa24(-1) * f(i-1) &
  &         + aa24(0) * f(i) + aa24(1) * f( i+1 ) + aa24(2) * f( i+2 ) + &
  &           aa24(3) * f( i+3 ) + aa24(4) * f( i+4 ) )
      ELSEIF ( I == M2-2 ) THEN
        fx(i) = 1.d0 / delx * ( aa42(-4) * f( i-4 ) + aa42(-3) * f( i-3 ) &
```

```
     &            + aa42(-2) * f( i-2 ) + aa42(-1) * f( i-1 ) + aa42(0) * f(i) + &
     &               aa42(1) * f( i+1 ) + aa42(2) * f( i+2 ) )
          ELSEIF ( I == M2-1 ) THEN
            fx(i) = 1.d0 / delx * ( aa51(-5) * f( i-5 ) + aa51(-4) * f( i-4 ) &
     &            + aa51(-3) * f(i-3) + aa51(-2) * f( i-2 ) + aa51(-1) * f( i-1 ) &
     &               + aa51(0) * f(i) + aa51(1) * f( i+1 ) )
          ELSEIF ( I == M2 ) THEN
            fx(i) = 1.d0 / delx * ( aa60(-6) * f( i-6 ) + aa60(-5) * f( i-5 ) &
     &            + aa60(-4) * f( i-4 ) + aa60(-3) * f( i-3 ) + aa60(-2) * f( i-2 ) &
     &               + aa60(-1) * f( i-1 ) + aa60(0) * f(i) )
          ELSE
            fx(i) = 1.d0 / delx * ( aa33(0) * f(i) + aa33(1) * ( f( i+1 ) - f( i-1 ) ) &
     &            + aa33(2) * ( f( i+2 ) - f( i-2 ) ) + aa33(3) * ( f( i+3 ) - f( i-3 ) ) )
          ENDIF
        ENDDO
      ELSE
        DO I = M1, M2
          fx(i) = 1.d0 / delx * ( aa33(0) * f(i) + aa33(1) * ( f( i+1 ) - f( i-1 ) ) &
     &         + aa33(2) * ( f( i+2 ) - f( i-2 ) ) + aa33(3) * ( f( i+3 ) - f( i-3 ) ) )
        ENDDO
      END IF
      END SUBROUTINE DRP7
    END MODULE M_Spatial_dis
    !
    !!
    MODULE LDDRK46Coefficient
      IMPLICIT NONE
      REAL(KIND=8):: C1( 2 : 5 ) = (/ 1.0d0/3.0d0, 2.0d0/3.0d0, 1.0d0/3.0d0, -1.0d0/3.0d0 /)
      REAL(KIND=8):: AFA1( 2 : 5 ) = (/ 0.5d0, 0.5d0, 1.0d0, -0.5d0 /)
      REAL(KIND=8):: C2( 2 : 7 ) = (/ 0.132349d0, 0.137341d0, 1.123446d0, 0.369835d0, &
      &                          0.467395d0, -1.230367d0 /)
      REAL(KIND=8):: AFA2( 2 : 7 ) = (/ 0.0467621d0/0.132349d0, 0.137286d0/0.137341d0, &
      &                          0.170975d0/1.123446d0, 0.197572d0/0.369835d0, &
      &                          0.282263d0/0.467395d0, -0.165142d0/1.230367d0 /)
    END MODULE LDDRK46Coefficient
    !
    !!
    MODULE FilterCoefficient
      ! ----------------------introduction-------------
      ! Here we will adopt the optimized coefficients given by Christophe Bogey
      ! (A family of low dispersive and low dissipative explicit
      ! schemes for flow and noise computations, 2004, JCP,)
      ! -------------------------------------------------------
      IMPLICIT NONE
      REAL(KIND=8) :: SigmaD
      PARAMETER( SigmaD = 0.01d0 )
      REAL(KIND=8) :: f11(-1:1)= (/ -0.25d0,0.5d0,-0.25d0 /)
      REAL(KIND=8) :: f22(-2:2)= (/ 0.0625d0,-0.25d0,0.375d0,-0.25d0,0.0625d0 /)
      !seven points
      REAL(KIND=8) :: f51(-5:1)= (/ -0.000027453993d0,-0.000747264596d0,-0.057347064865d0, &
      &                      0.223119093072d0,-0.356848072173d0,0.277628171524d0, &
      &                      -0.085777408970d0 /)
      REAL(KIND=8) :: f42(-4:2)= (/ 0.007318073189d0,-0.062038376258d0,0.186711738069d0, &
```

```fortran
&                    -0.294622121167d0,0.273321177980d0,-0.143339502575d0, &
&                    0.032649010764d0 /)
REAL(KIND=8) :: f33(-3:3)= (/ -0.0238530481912d0,0.1063035787698d0,-0.2261469518087d0, &
&                     0.2873928424602d0 - 3.469447d-18,-0.2261469518087d0, &
&                     0.1063035787698d0,-0.0238530481912d0 /)
REAL(KIND=8) :: f24(-2:4)= (/ 0.032649010764d0,-0.143339502575d0,0.273321177980d0, &
&                    -0.294622121167d0,0.186711738069d0,-0.062038376258d0, &
&                    0.007318073189d0 /)
REAL(KIND=8) :: f15(-1:5)= (/ -0.085777408970d0,0.277628171524d0,-0.356848072173d0, &
&                    0.223119093072d0,-0.057347064865d0,-0.000747264596d0, &
&                    -0.000027453993d0 /)
!nine points
REAL(KIND=8) :: f44(-4:4)= (/ 0.008228661760d0,-0.04521111936d0,0.120007591680d0, &
&                    -0.204788880640d0, 0.243527493120d0, -0.204788880640d0, &
&                    0.120007591680d0,-0.04521111936d0,0.008228661760d0 /)
!eleven POINTS
REAL(KIND=8) :: f55(-5:5)= (/ -0.002999540835d0,0.018721609157d0,-0.059227575576d0, &
&                    0.123755948787d0,-0.187772883589d0, 0.215044884112d0, &
&                    -0.187772883589d0,0.123755948787d0,-0.059227575576d0, &
&                    0.018721609157d0,-0.002999540835d0 /)
REAL(KIND=8) :: f46(-4:6)= (/ 0.008391235145d0,-0.047402506444d0,0.121438547725d0, &
&                    -0.200063042812d0,0.240069047836d0,-0.207269200140d0, &
&                    0.122263107844d0,-0.047121062819d0,0.009014891495d0, &
&                    0.001855812216d0,-0.001176830044d0 /)
REAL(KIND=8) :: f37(-3:7)= (/ -0.000054596010d0,0.042124772446d0,-0.173103107841d0, &
&                    0.299615871352d0,-0.276543612935d0,0.131223506571d0, &
&                    -0.023424966418d0,0.013937561779d0,-0.024565095706d0, &
&                    0.013098287852d0,-0.002308621090d0 /)
REAL(KIND=8) :: f28(-2:8)= (/ 0.052523901012d0,-0.206299133811d0,0.353527998250d0, &
&                    -0.348142394842d0,0.181481803619d0,0.009440804370d0, &
&                    -0.077675100452d0,0.044887364863d0,-0.009971961849d0, &
&                    0.000113359420d0,0.000113359420d0 /)
!boundary
REAL(KIND=8) :: f30(-3:0)= (/ -0.035d0,0.179117647059d0,-0.465d0,0.320882352941d0 /)
REAL(KIND=8) :: f03(0:3)= (/ 0.320882352941d0,-0.465d0,0.179117647059d0,-0.035d0 /)
!
CONTAINS
!
SUBROUTINE BasicFilter( df, f, M1, N1, M2, N2 )
IMPLICIT NONE
INTEGER:: I, J
INTEGER:: M1, N1, M2, N2
REAL(KIND=8):: df( M1 : N1 ), f( M2 : N2 )
!
IF ( M1 == N1 ) THEN
  ! use some partial-side filter for space

ELSE
  ! use center filter whole of the space
  DO I = M1, N1
    DO J = -3, 3
      df(I) = df(I) + f33(J) * f( I+J )
    END DO
```

```
        END DO
      END IF
      END SUBROUTINE BasicFilter
    END MODULE FilterCoefficient
    !! >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>..
    !! >>>>>>>>>>>>>>>>>>>>>>>>>>>> Main Program >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
      INCLUDE 'Module_Constants_sub.f90'
      INCLUDE 'NumericalProbe_sub.f90'
      INCLUDE 'AbsorbingCoefficient_sub.f90'
      INCLUDE 'ExchangeInterfaceData_sub.f90'
      INCLUDE 'ExchangeInterfaceDataNew_sub.f90'
      INCLUDE 'GetConservativeVariables_sub.f90'
      INCLUDE 'SolidPMLSource_sub.f90'
      INCLUDE 'GetMaxResidual_sub.f90'
      INCLUDE 'GetMeanFlow_sub.f90'
      INCLUDE 'GetMeshSize_sub.f90'
      INCLUDE 'GetOriginalVariables_sub.f90'
      INCLUDE 'InitializeAcousticField_sub.f90'
      INCLUDE 'LDDRK_sub.f90'
      INCLUDE 'Mesh_sub.f90'
      INCLUDE 'Models_sub.f90'
      INCLUDE 'TransformCoordinate_sub.f90'
      INCLUDE 'GetEffectMatrix_sub.f90'
      INCLUDE 'GetDuDt_sub.f90'
      INCLUDE 'ForcingSolver_sub.f90'
      INCLUDE 'CorrectAcousticField_sub.f90'
      INCLUDE 'Distribution_sub.f90'
    !   INCLUDE 'lapack_LINUX.a'
    !   INCLUDE 'blas_LINUX.a'
      INCLUDE 'SOR_sub.f90'
      INCLUDE 'DooliteSolve_sub.f90'
      INCLUDE 'OutputToTextFile_sub.f90'
      INCLUDE 'Filter_sub.f90'
    PROGRAM AcousticScatteringParallelComputing
      !------      ------------------  ---------    ----------------
      !------      -----------program introduction --------      ---------
      !
      !
      !
      !------      ------------------  ----------      ----------------
      !......
      !......
      !-----var     ---------------------------------------------------
      !-----iables  ---------------------------------------------------
      ! EffectRange: the influence width of Delta function for each lagrange
      !            point
      ! NormalVector: the normal vector for each lagrange point
      ! Shapes: the surface point for analysis model
      ! ShapesForP: shape points for pressure interpolation of solid surface
      ! Cell_S: area of length for each cell of analysis model
      ! AV:   effect matrices for velocity
      ! AT:   effect matrices for temperature
      ! LagForce: lagrange forcing
```

```fortran
! SurfaceUVW:   the fluid velocity for solid surface lagrange points
! Aprocessor:   (local)Effect matrices for each processor
! Atotal:  temperal (global) effect matrices
! DuDt0:     (local)velocity source for each processor
! DuDt1:     temperal (global) velocity source
! DuDt:   (global) velocity source
! ....................
! MeshX, MeshY, MeshZ: Cartesian Mesh grid coordinates
! DeltaX, DeltaY, DeltaZ: Cartesian grid size for each direction
! U0, V0, W0, P0, ROU0: the background flow field
! U, V, W, P, ROU: acoustic field variables
! Q1~4, F1~4, G1~4, H1~4: conservative variables
! ....................
! Sigma~ : the absorbing coefficient for each side
! Computational zone
!  ---------------------------------------------------
!  -      -      PML: SigmaY2             -      -
!  ---------------------------------------------------
!  -  PML  -                          -  PML  -
!  -  S  -         Main                -  S  -
!  -  i  -           Zone             -  i  -
!  -  g  -                          -  g  -
!  -  m  -                          -  m  -
!  -  aX1 -                          -  aX2 -
!  ---------------------------------------------------
!  -      -      PML: SigmaY2             -      -
!  ---------------------------------------------------
!......................
! Au1~, Au2~, Au3~:    auxiliary variables for each direction
! S1~4:    source term for PML equations
!-----intro    ---------------------------------------------------
!-----duction    ---------------------------------------------------
!
USE MPI
USE Constants_Model
IMPLICIT NONE
INTEGER:: I, J, k
! for lagrange points
INTEGER:: PLN
INTEGER, ALLOCATABLE:: EffectRange( :, : )
REAL(KIND=8), ALLOCATABLE:: NormalVector( :, : ), Shapes( :, : )
REAL(KIND=8), ALLOCATABLE:: ShapesForP( :, : ), Cell_S( : )
REAL(KIND=8), ALLOCATABLE:: AV( :, : ), AT( :, : )
REAL(KIND=8), ALLOCATABLE:: LagForce( :, : ), SurfUVW( :, : )
REAL(KIND=8), ALLOCATABLE:: AProcessor( :, : ), ATotal( :, : )
REAL(KIND=8), ALLOCATABLE:: DuDt0( : ), DuDt1( : ), DuDt( : )
INTEGER, ALLOCATABLE:: DuDtFlag( : ), DuDtFlagRoot( : )
!
! for Cartesian grid information
REAL(KIND=8), ALLOCATABLE:: MeshX( : ), MeshY( : ), MeshZ( : )
REAL(KIND=8), ALLOCATABLE:: DeltaX( : ), DeltaY( : ), DeltaZ( : )
!
```

```
! background flow field
REAL(KIND=8), ALLOCATABLE:: U0( :, :, : ), V0( :, :, : ), W0( :, :, : )
REAL(KIND=8), ALLOCATABLE:: P0( :, :, : ), ROU0( :, :, : ), C0( :, :, : )
!
! acoustic field: nonconservative variables and conservative variables
! output to file for main processor
REAL(KIND=8), ALLOCATABLE:: TotalP( :, :, : ), TotalROU( :, :, : )
REAL(KIND=8), ALLOCATABLE:: TotalU( :, :, : ), TotalV( :, :, : )
REAL(KIND=8), ALLOCATABLE:: TotalW( :, :, : ), TotalOmiga( :, :, : )
!
! calculation
REAL(KIND=8), ALLOCATABLE:: U( :, :, : ), V( :, :, : ), W( :, :, : )
REAL(KIND=8), ALLOCATABLE:: Omiga( :, :, : )
REAL(KIND=8), ALLOCATABLE:: P( :, :, : ), ROU( :, :, : )
REAL(KIND=8), ALLOCATABLE:: Du( :, :, : ), Dv( :, :, : ), Dw( :, :, : )
REAL(KIND=8), ALLOCATABLE:: Uold( :, :, : ), Vold( :, :, : ), Wold( :, :, : )
REAL(KIND=8), ALLOCATABLE:: Pold( :, :, : )
REAL(KIND=8), ALLOCATABLE:: Q1( :, :, : ), Q2( :, :, : )
REAL(KIND=8), ALLOCATABLE:: Q3( :, :, : ), Q4( :, :, : )
REAL(KIND=8), ALLOCATABLE:: F1( :, :, : ), F2( :, :, : )
REAL(KIND=8), ALLOCATABLE:: F3( :, :, : ), F4( :, :, : )
REAL(KIND=8), ALLOCATABLE:: G1( :, :, : ), G2( :, :, : )
REAL(KIND=8), ALLOCATABLE:: G3( :, :, : ), G4( :, :, : )
REAL(KIND=8), ALLOCATABLE:: H1( :, :, : ), H2( :, :, : )
REAL(KIND=8), ALLOCATABLE:: H3( :, :, : ), H4( :, :, : )
!
! PML zone
REAL(KIND=8):: SigmaX1( -(NPML-1) : 0 )
REAL(KIND=8):: SigmaX2( 1 : NPML )
REAL(KIND=8):: SigmaY1( -(NPML-1) : 0 )
REAL(KIND=8):: SigmaY2( 1 : NPML )
REAL(KIND=8):: SigmaZ1( -(NPML-1) : 0 )
REAL(KIND=8):: SigmaZ2( 1 : NPML )
REAL(KIND=8), ALLOCATABLE:: Au11( :, :, : ), Au12( :, :, : )
REAL(KIND=8), ALLOCATABLE:: Au13( :, :, : ), Au14( :, :, : )
REAL(KIND=8), ALLOCATABLE:: Au21( :, :, : ), Au22( :, :, : )
REAL(KIND=8), ALLOCATABLE:: Au23( :, :, : ), Au24( :, :, : )
REAL(KIND=8), ALLOCATABLE:: Au31( :, :, : ), Au32( :, :, : )
REAL(KIND=8), ALLOCATABLE:: Au33( :, :, : ), Au34( :, :, : )
REAL(KIND=8), ALLOCATABLE:: S1( :, :, : ), S2( :, :, : )
REAL(KIND=8), ALLOCATABLE:: S3( :, :, : ), S4( :, :, : )
REAL(KIND=8), ALLOCATABLE:: SAu11( :, :, : )
REAL(KIND=8), ALLOCATABLE:: SAu12( :, :, : )
REAL(KIND=8), ALLOCATABLE:: SAu13( :, :, : )
REAL(KIND=8), ALLOCATABLE:: SAu14( :, :, : )
REAL(KIND=8), ALLOCATABLE:: SAu21( :, :, : )
REAL(KIND=8), ALLOCATABLE:: SAu22( :, :, : )
REAL(KIND=8), ALLOCATABLE:: SAu23( :, :, : )
REAL(KIND=8), ALLOCATABLE:: SAu24( :, :, : )
REAL(KIND=8), ALLOCATABLE:: SAu31( :, :, : )
REAL(KIND=8), ALLOCATABLE:: SAu32( :, :, : )
REAL(KIND=8), ALLOCATABLE:: SAu33( :, :, : )
```

```
REAL(KIND=8), ALLOCATABLE:: SAu34( :, :, : )
!!
!MPI variables definition
INTEGER:: IERR, NUMPROCS
INTEGER:: MYID, MYROOT
INTEGER:: MYLEFT, MYRIGHT, MYUPPER, MYLOWER, MYFORWARD, MYREAR
INTEGER:: PX, PY, PZ
INTEGER:: HTYPE, VTYPE
INTEGER:: XTYPE, YTYPE, ZTYPE
INTEGER, ALLOCATABLE:: STATUS( :, : ), REQ(:)
INTEGER:: COUNT
INTEGER,ALLOCATABLE:: BLOCKLENS( : ), INDICES( : )
INTEGER:: SENDCNT
INTEGER, ALLOCATABLE:: RECVCNT( : )
INTEGER, ALLOCATABLE:: DISPLS( : )
!
! grid number for each processor( MPI )
! NA: ghost point for MPI information exchange
!     Here using DRP scheme, NA = 3.
INTEGER:: XN1, YN2, ZN3
!
! coordinate transformation
REAL(KIND=8), ALLOCATABLE:: Jacobi( :, :, : )
REAL(KIND=8), ALLOCATABLE:: KexiX( : ), EitaY( : ), TaoZ( : )
!
! loop control variables
INTEGER:: Tstep0, K0, MaxTimeStep, Loops, Flag
REAL(KIND=8):: MaxResidual, MaxResidual0
!
! output flag
INTEGER:: IPX, IPY, IPZ
INTEGER:: SizeN1, SizeN2, SizeN3
INTEGER:: OUTPUTFlag, FLAG0, SIZE0
CHARACTER(LEN=80):: FilenameINPUT, FilenameOUTPUT, FilenameFile
!!
!step- 1: initialize the parallel processors
CALL MPI_INIT( IERR )
CALL MPI_COMM_SIZE( MPI_COMM_WORLD, NUMPROCS, IERR )
CALL MPI_COMM_RANK( MPI_COMM_WORLD, MYID, IERR )
!
MYROOT = 0
!
! for data exchanger
ALLOCATE( RECVCNT( NUMPROCS ) )
ALLOCATE( DISPLS( NUMPROCS ) )
!
! for series I/O: output results( MPI_ISEND, MPI_IRECV )
IF ( MYID == MYROOT ) THEN
   SIZE0 = NUMPROCS - 1
   ALLOCATE( REQ( SIZE0 ) )
   ALLOCATE( STATUS( MPI_STATUS_SIZE, SIZE0 ) )
ELSE
   SIZE0 = 1
```

```
    ALLOCATE( REQ( SIZE0:SIZE0 ) )
    ALLOCATE( STATUS( MPI_STATUS_SIZE, SIZE0 ) )
  END IF
!
  IF ( ( NPX * NPY * NPZ ) .NE. NUMPROCS ) THEN
    WRITE( *, * ) 'The processor grid distribution doesn''t ', &
&         ' conform with the processor summation! '
    STOP
    CALL MPI_FINALIZE( IERR )
  END IF
!!
!get the current processor coordinates( PX, PY, PZ )
  PZ = MYID / ( NPX * NPY )
  PY = ( MYID - PZ * NPX * NPY ) / ( NPX )
  PX = ( MYID - PZ * NPX * NPY ) - PY * NPX
!
!determine the topology relationship of all the processors
!!
  MYLEFT = MYID - 1
  IF ( MOD( MYID, NPX ) .EQ. 0 ) MYLEFT = MPI_PROC_NULL
  MYRIGHT = MYID + 1
  IF ( MOD( MYRIGHT, NPX ) .EQ. 0 ) MYRIGHT = MPI_PROC_NULL
  MYREAR = MYID + NPX
  IF ( BCFlag == 0 ) THEN
    ! free space for y direction: using PML boundary condition
    IF ( MYREAR .GE. ( PZ+1 )*NPX*NPY ) MYREAR = MPI_PROC_NULL
  ELSEIF( BCFlag == 1 ) THEN
    ! periodic boundary condition for y direction: using periodic bc
    IF ( MYREAR .GE. ( PZ+1 )*NPX*NPY ) MYREAR = MYID - NPX * ( NPY - 1 )
  END IF
  MYFORWARD = MYID - NPX
  IF ( BCFlag == 0 ) THEN
    ! free space for y -direction: using PML boundary condition
    IF ( MYFORWARD .LT. PZ*NPX*NPY ) MYFORWARD = MPI_PROC_NULL
  ELSEIF( BCFlag == 1 ) THEN
    ! periodic boundary condition for y direction: using periodic bc
    IF ( MYFORWARD .LT. PZ*NPX*NPY ) MYFORWARD = MYID + NPX * ( NPY - 1 )
  END IF
  MYUPPER = MYID + NPX * NPY
  IF ( MYUPPER .GE. NUMPROCS ) MYUPPER = MPI_PROC_NULL
  MYLOWER = MYID - NPX * NPY
  IF( MYLOWER .LT. 0 ) MYLOWER = MPI_PROC_NULL
!
!!
!
  OUTPUTFlag = 0
  FilenameINPUT = './INPUT/'
  FilenameOUTPUT = './OUTPUT/'
  FilenameFile = 'ShapesFile.dat'
!DO I = 1, NumProbe
!   ProbeAngle( I ) = 2.0d0 * PI * I / NumProbe
!END DO
!
```

```
    !step- 2: read the number for surface mesh
    IF ( MYID == MYROOT ) THEN
       OPEN( UNIT = 10, FILE = TRIM( TRIM( FilenameINPUT ) // TRIM( FilenameFile ) ) )
!       READ( UNIT = 10, FMT = * ) LN
       READ( UNIT = 10, FMT = * ) NumBody
       CLOSE( UNIT = 10 )
    END IF
    !!
    !bcast LN to every processor
    COUNT = 1
    CALL MPI_BCAST( NumBody, COUNT, MPI_INTEGER, MYROOT, &
    &    MPI_COMM_WORLD, IERR )
    !
    ALLOCATE( MultiLN( 1:NumBody ) )
    !
    IF ( MYID == MYROOT ) THEN
       OPEN( UNIT = 10, FILE = TRIM( TRIM( FilenameINPUT ) // TRIM( FilenameFile ) ) )
!       READ( UNIT = 10, FMT = * ) LN
       READ( UNIT=10, FMT = * ) NumBody
       READ( UNIT=10, FMT = * ) MultiLN( 1: NumBody )
       CLOSE( UNIT = 10 )
       LN = SUM( MultiLN )
    END IF
    COUNT = NumBody
    CALL MPI_BCAST( MultiLN, COUNT, MPI_INTEGER, MYROOT, &
    &    MPI_COMM_WORLD, IERR )
    !
    COUNT = 1
    CALL MPI_BCAST( LN, COUNT, MPI_INTEGER, MYROOT, &
    &      MPI_COMM_WORLD, IERR )
    !!
    !step- 3: allocate array variables for immersed body
    !  allocate variable related to the immersed boundary
    ALLOCATE( EffectRange( LN, 6 ), NormalVector( LN, 3 ) )
    ALLOCATE( Shapes( LN, 3 ), ShapesForP( LN, 3 ), Cell_S( LN ) )
    ALLOCATE( AV( LN, LN ), AT( LN, LN ) )
    ALLOCATE( LagForce( LN, 3 ), SurfUVW( LN, 3 ) )
    PLN = CEILING( 1.0d0 * LN / NUMPROCS )
    ALLOCATE( AProcessor( LN, PLN ) )
    ALLOCATE( ATotal( LN, NUMPROCS * PLN ) )
    ALLOCATE( DuDt0( LN ) )
    ALLOCATE( DuDt( LN ) )
    ALLOCATE( DuDt1( LN ) )
    ALLOCATE( DuDtFlag( LN+1 ) )
    ALLOCATE( DuDtFlagRoot( LN ) )
    !
    !!
    !read the original body coordinates for the main processor
    IF ( MYID == MYROOT ) THEN
       OPEN( UNIT = 10, FILE = TRIM( TRIM( FilenameINPUT ) // TRIM( FilenameFile ) ) )
!       READ( UNIT = 10, FMT = * ) LN
       READ( UNIT = 10, FMT = * ) NumBody
```

```
         READ( UNIT = 10, FMT = * ) MultiLN( 1: NumBody )
         IF ( ModelFlag == 0 ) THEN
            DO I = 1, LN
               READ( UNIT = 10, FMT = * ) shapes( I, : )
!                 READ( UNIT=10, FMT=* ) TempShapes(1,1:4)
!                 Shapes(I,:) = TempShapes(1,1:3)
!                 CELL_S(I) = TempShapes(1,4)
            END DO
            Shapes( : , 3 ) = Centre( 3 )
         ELSE
            DO I = 1, LN
               READ( UNIT = 10, FMT = * ) shapes( I, : )
!                 READ( UNIT=10, FMT = * ) TempShapes(1,1:4)
!                 Shapes(I,:) = TempShapes( 1,1:3 )
!                 CELL_S(I) = TempShapes(1,4)
            END DO
         END IF
         CLOSE( UNIT = 10 )
      END IF
      !
      ! bcast the shapes to other processors
      !bcast LN to every processor
      COUNT = LN*3
      CALL MPI_BCAST( Shapes(1,1), COUNT, MPI_DOUBLE_PRECISION, MYROOT, &
      &    MPI_COMM_WORLD, IERR )
      ! f
      !step- 4: input the analysis model
      !IF ( MYID == MYROOT ) THEN
      CALL Models( LRef, R_PML0, Shapes, NormalVector, Cell_S, LN, NumBody, MultiLN, AOA, &
      &    Centre, ModelFlag )
      !END IF
      !!
      !step-5: get the size for Cartesian mesh
      CALL GetMeshSize( N1, N2, N3, StartEnd, CutPoints, DeltaXYZ, &
      &      R_PML, R_PML0, Shapes, Cell_S, LN, Centre, MaxX, MaxY, &
      &      MaxZ, SolidRelativePosi, q0, LRef, NPML, NPX, NPY, NPZ, &
      &      ModelFlag, MYID, FilenameOUTPUT )
      !!
      NumProbe(1) = LN
      NumProbe(2) = N1
      ALLOCATE( ProbeCoordinate( SUM(NumProbe), 3 ) )
      ALLOCATE( ProbeAngle( NumProbe(2) ), Corner( SUM(NumProbe), 3 ) )
      ALLOCATE( SideLength( SUM(NumProbe), 3 ) )
      ALLOCATE( RelativePosi( SUM(NumProbe), 3 ) )
      ALLOCATE( ProbePressure( SUM(NumProbe) ) )
      ALLOCATE( ProbePressure0( SUM(NumProbe) ) )
      ALLOCATE( ProbePressure1( SUM(NumProbe) ) )
      ALLOCATE( ProbePressure2( SUM(NumProbe) ) )
      ALLOCATE( ProbeFlag( SUM(NumProbe)+1 ) )
      ALLOCATE( ProbeFlagRoot( SUM(NuMProbe) ) )
      ProbePressure = 0.0d0
      !
```

```fortran
!   IF ( MYID == MYROOT ) WRITE( *, * ) N1, N2, N3
    !
    !get the mesh number for each processor
    XN1 = ( 2 * NPML + N1 ) / NPX
    YN2 = ( 2 * NPML + N2 ) / NPY
    IF ( ModelFlag == 0 ) THEN
       ! 2-D Model
       ZN3 = 1
    ELSE
       ! 3-D Model
       ZN3 = ( 2 * NPML + N3 ) / NPZ
    END IF
    !!
    !---------------------------------------------------------------------
    !              -  M  A  I  N                    ****
    !                 -  Z  O  N  E  -               ****
    !                    -  V  A  R  I  A  T  I  O  N-  ****
    !              -  P  M  L  -                    ****
    !                 -  Z  O  N  E  -               ****
    !---------------------------------------------------------------------
    !PML equation:
    ! : Q_t+(F-F0)_x + ( G-G0 )_y + ( H-H0 )_z + sigma(x) * Au(1) + sigma(y)
    !   * Au(2) + sigma(z) * Au(3) + beita * sigma(x) * (F-F0) = 0  ---- (1)
    ! : Au(1)_t + sigma(x) * Au(1) + (F-F0)_x + beita * sigma(x) * (F-F0)
    !   = 0 -------------------------------------------------------------(2)
    ! : Au(2)_t + sigma(y) * Au(2) + (G-G0)_y = 0 -----------------------(3)
    ! : Au(3)_t + sigma(z) * Au(3) + (H-H0)_z = 0 -----------------------(4)
    !-----------
    !-----------
    ! Because the governing equation doesn't include viscous terms, the
    ! equations (1)~(4) only need to be solved.
    ! (Here can refer to the paper< Absorbing boundary conditions for nonl-
    ! inear Euler and Navier-stokes equations based on the perfectly match-
    ! -ed layer techniques >. Fang Q. Hu, X.D Li, D.K Lin, 2008) for equation
    ! (38)~(44)
    !----------
    !----------
    !step-6: allocate array variables for Cartesian mesh of each processor
    ALLOCATE( MeshX( -(NPML-1) : (N1+NPML) ) )
    ALLOCATE( MeshY( -(NPML-1) : (N2+NPML) ) )
    ALLOCATE( MeshZ( -(NPML-1) : (N3+NPML) ) )
    ALLOCATE( DeltaX( -(NPML-1) : (N1+NPML-1) ) )
    ALLOCATE( DeltaY( -(NPML-1) : (N2+NPML-1) ) )
    ALLOCATE( DeltaZ( -(NPML-1) : (N3+NPML-1) ) )
    ALLOCATE( Jacobi( -(NPML-1):(N1+NPML), -(NPML-1):(N2+NPML), -(NPML-1):(N3+NPML) ) )
    ALLOCATE( KexiX( -(NPML-1) : ( N1 + NPML ) ) )
    ALLOCATE( EitaY( -(NPML-1) : ( N2 + NPML ) ) )
    ALLOCATE( TaoZ( -(NPML-1) : ( N3 + NPML ) ) )
    ALLOCATE( U0( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
    ALLOCATE( V0( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
    ALLOCATE( W0( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
    ALLOCATE( P0( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
```

```
ALLOCATE( ROU0( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
ALLOCATE( C0( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
ALLOCATE( U( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
ALLOCATE( V( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
ALLOCATE( W( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
ALLOCATE( Omiga( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
ALLOCATE( P( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
ALLOCATE( ROU( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
ALLOCATE( Du( 1 : XN1, 1 : YN2, 1 : ZN3 ) )
ALLOCATE( Dv( 1 : XN1, 1 : YN2, 1 : ZN3 ) )
ALLOCATE( Dw( 1 : XN1, 1 : YN2, 1 : ZN3 ) )
ALLOCATE( Q1( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
ALLOCATE( Q2( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
ALLOCATE( Q3( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
ALLOCATE( Q4( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
ALLOCATE( F1( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
ALLOCATE( F2( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
ALLOCATE( F3( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
ALLOCATE( F4( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
ALLOCATE( G1( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
ALLOCATE( G2( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
ALLOCATE( G3( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
ALLOCATE( G4( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
ALLOCATE( H1( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
ALLOCATE( H2( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
ALLOCATE( H3( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
ALLOCATE( H4( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
ALLOCATE( Au11( 1 : XN1, 1 : YN2, 1 : ZN3 ) )
ALLOCATE( Au12( 1 : XN1, 1 : YN2, 1 : ZN3 ) )
ALLOCATE( Au13( 1 : XN1, 1 : YN2, 1 : ZN3 ) )
ALLOCATE( Au14( 1 : XN1, 1 : YN2, 1 : ZN3 ) )
ALLOCATE( Au21( 1 : XN1, 1 : YN2, 1 : ZN3 ) )
ALLOCATE( Au22( 1 : XN1, 1 : YN2, 1 : ZN3 ) )
ALLOCATE( Au23( 1 : XN1, 1 : YN2, 1 : ZN3 ) )
ALLOCATE( Au24( 1 : XN1, 1 : YN2, 1 : ZN3 ) )
ALLOCATE( Au31( 1 : XN1, 1 : YN2, 1 : ZN3 ) )
ALLOCATE( Au32( 1 : XN1, 1 : YN2, 1 : ZN3 ) )
ALLOCATE( Au33( 1 : XN1, 1 : YN2, 1 : ZN3 ) )
ALLOCATE( Au34( 1 : XN1, 1 : YN2, 1 : ZN3 ) )
ALLOCATE( S1( 1 : XN1, 1 : YN2, 1 : ZN3 ) )
ALLOCATE( S2( 1 : XN1, 1 : YN2, 1 : ZN3 ) )
ALLOCATE( S3( 1 : XN1, 1 : YN2, 1 : ZN3 ) )
ALLOCATE( S4( 1 : XN1, 1 : YN2, 1 : ZN3 ) )
ALLOCATE( SAu11( 1 : XN1, 1 : YN2, 1 : ZN3 ) )
ALLOCATE( SAu12( 1 : XN1, 1 : YN2, 1 : ZN3 ) )
ALLOCATE( SAu13( 1 : XN1, 1 : YN2, 1 : ZN3 ) )
ALLOCATE( SAu14( 1 : XN1, 1 : YN2, 1 : ZN3 ) )
ALLOCATE( SAu21( 1 : XN1, 1 : YN2, 1 : ZN3 ) )
ALLOCATE( SAu22( 1 : XN1, 1 : YN2, 1 : ZN3 ) )
ALLOCATE( SAu23( 1 : XN1, 1 : YN2, 1 : ZN3 ) )
ALLOCATE( SAu24( 1 : XN1, 1 : YN2, 1 : ZN3 ) )
ALLOCATE( SAu31( 1 : XN1, 1 : YN2, 1 : ZN3 ) )
```

```
ALLOCATE( SAu32( 1 : XN1, 1 : YN2, 1 : ZN3 ) )
ALLOCATE( SAu33( 1 : XN1, 1 : YN2, 1 : ZN3 ) )
ALLOCATE( SAu34( 1 : XN1, 1 : YN2, 1 : ZN3 ) )
ALLOCATE( Uold( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
ALLOCATE( Vold( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
ALLOCATE( Wold( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
ALLOCATE( Pold( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) ) )
!
! total variables for outputting to file in main processor
IF ( MYID == MYROOT ) THEN
  SizeN1 = N1 + 2 * NPML
  SizeN2 = N2 + 2 * NPML
  IF ( ModelFlag == 0 ) THEN
    ! 2-D Model
    SizeN3 = 1
  ELSE
    ! 3-D Model
    SizeN3 = N3 + 2 * NPML
  END IF
  ALLOCATE( TotalP( SizeN1, SizeN2, SizeN3 ) )
  ALLOCATE( TotalRou( SizeN1, SizeN2, SizeN3 ) )
  ALLOCATE( TotalU( SizeN1, SizeN2, SizeN3 ) )
  ALLOCATE( TotalV( SizeN1, SizeN2, SizeN3 ) )
  ALLOCATE( TotalW( SizeN1, SizeN2, SizeN3 ) )
  ALLOCATE( TotalOmiga( SizeN1, SizeN2, SizeN3 ) )
END IF
!
! -----------------------------------------------------------------------------
! Define new datatype for output operation and the MPI message transfer
! -----------------------------------------------------------------------------
!
! create new datatype to output variables to file from each processor
! new datatype for sending process
COUNT = YN2 * ZN3
ALLOCATE( BLOCKLENS( COUNT ) )
ALLOCATE( INDICES( COUNT ) )
BLOCKLENS = XN1
DO J = 1, ZN3
  DO I = 1, YN2
    INDICES( YN2*(J-1)+I ) = (J-1) * (XN1+2*NA) * (YN2+2*NA) + (I-1) * (XN1+2*NA)
  END DO
END DO
CALL MPI_TYPE_INDEXED( COUNT, BLOCKLENS, INDICES, MPI_DOUBLE_PRECISION, HTYPE,...
CALL MPI_TYPE_COMMIT( HTYPE, IERR )
!
! new datattype for receving process
DO J = 1, ZN3
  DO I = 1, YN2
    INDICES( YN2*(J-1)+I ) = (J-1) * SizeN1 * SizeN2 + (I-1) * SizeN1
  END DO
END DO
CALL MPI_TYPE_INDEXED( COUNT, BLOCKLENS, INDICES, MPI_DOUBLE_PRECISION, VTYPE, ...
```

```
CALL MPI_TYPE_COMMIT( VTYPE, IERR )
!
! create new datatype for data exchange on the boundary for each processor
! TYPE- 1: XTYPE ------for forward and rearward side data exchange
DEALLOCATE( BLOCKLENS, INDICES )
COUNT = NA * ZN3
ALLOCATE( BLOCKLENS( COUNT ) )
ALLOCATE( INDICES( COUNT ) )
BLOCKLENS = XN1
DO J = 1, ZN3
   DO I = 1, NA
      INDICES( NA*(J-1)+I ) = (J-1) * (XN1+2*NA) * (YN2+2*NA) + (I-1) * (XN1+2*NA)
   END DO
END DO
CALL MPI_TYPE_INDEXED( COUNT, BLOCKLENS, INDICES, MPI_DOUBLE_PRECISION, XTYPE, ...
CALL MPI_TYPE_COMMIT( XTYPE, IERR )
!
! TYPE- 2: YTYPE----------for left and right side data exchange
DEALLOCATE( BLOCKLENS, INDICES )
COUNT = YN2 * ZN3
ALLOCATE( BLOCKLENS( COUNT ) )
ALLOCATE( INDICES( COUNT ) )
BLOCKLENS = NA
DO J = 1, ZN3
   DO I = 1, YN2
      INDICES( YN2*(J-1)+I ) = (J-1) * (XN1+2*NA) * (YN2+2*NA) + (I-1) * (XN1+2*NA)
   END DO
END DO
CALL MPI_TYPE_INDEXED( COUNT, BLOCKLENS, INDICES, MPI_DOUBLE_PRECISION, YTYPE, ...
CALL MPI_TYPE_COMMIT( YTYPE, IERR )
!
! TYPE- 3: ZTYPE ---------for upper and lower side data exchange
DEALLOCATE( BLOCKLENS, INDICES )
COUNT = YN2 * NA
ALLOCATE( BLOCKLENS( COUNT ) )
ALLOCATE( INDICES( COUNT ) )
BLOCKLENS = XN1
DO J = 1, NA
   DO I = 1, YN2
      INDICES( YN2*(J-1)+I ) = (J-1) * (XN1+2*NA) * (YN2+2*NA) + (I-1) * (XN1+2*NA)
   END DO
END DO
CALL MPI_TYPE_INDEXED( COUNT, BLOCKLENS, INDICES, MPI_DOUBLE_PRECISION, ZTYPE, ...
CALL MPI_TYPE_COMMIT( ZTYPE, IERR )
! --------------------------------------------------------------------------------
! ----------------------------- new datatype for MPI ------------------------------
!----
!----
!--------------------------------------------------------------------
!           - M A I N                    ****
!              - Z O N E -               ****
!                 - V A R I A T I O N- ****
```

```
!             - P  M  L -                    ****
!                - Z  O  N  E -                ****
!----------------------------------------------------------------------
!!
!step- 7: grid generation
CALL Mesh( MeshX, MeshY, MeshZ, DeltaX, DeltaY, DeltaZ, N1, N2, N3, &
&      NPML, StartEnd, CutPoints, DeltaXYZ, q0, ModelFlag, MYID, FilenameOUTPUT )
!
CALL NumericalProbe( Corner, SideLength, RelativePosi, ProbeCoordinate, NumProbe, &
&      Rp, ProbeAngle, NormalVector, Shapes, MeshX, MeshY, MeshZ, DeltaXYZ, &
&         N1, N2, N3, NPML, ModelFlag )
!!
!step- 7.1: coordinate transformation/ get Jacobi matrix
CALL TransformCoordinate( Jacobi, KexiX, EitaY, TaoZ, &
&   MeshX, MeshY, MeshZ, N1, N2, N3, NPML, DeltaXYZ, ModelFlag )
!
!step- 7.2: get the absorbing coefficient SigmaX, SigmaY, SigmaZ for PML zone
CALL AbsorbingCoefficient( Beita, SigmaX1, SigmaX2, SigmaY1, SigmaY2, &
&         SigmaZ1, SigmaZ2, AFA, SIGMA0, NPML, MeshX, MeshY, MeshZ, &
&               N1, N2, N3, BeitaFlag, ModelFlag )
!!
!step- 8: get the background flow field
CALL GetMeanFlow( U0, V0, W0, P0, ROU0, C0, NA, XN1, YN2, ZN3, &
&   MeshX, MeshY, MeshZ, Centre, NPML, N1, N2, N3, PX, PY, PZ, &
&      NPX, NPY, NPZ, ModelFlag )
!!
!step- 9: initialize acoutic field
CALL InitializeAcousticField( U, V, W, P, ROU, Au11, Au12, Au13, Au14, &
&      Au21, Au22, Au23, Au24, Au31, Au32, Au33, Au34, NA, XN1, YN2, &
&         ZN3, MeshX, MeshY, MeshZ, C0, NPML, N1, N2, N3, PX, PY, PZ, &
&             NPX, NPY, NPZ, ModelFlag )
!!
!step- 10: enter loops
!!---------------------------Algorithm--------------------
!here we solve conservative form of acoustic propagation
!  governing equations, the equation group can be got in
!  < Acoustic Scattering in non-uniform flow >. Formula (7)
!  Derived by Cheng Long( 2015 )
!!-------------------------------------------------------
DeltaT = DeltaXYZ * CFL
Tstep0 = floor( 0.1d0 / DeltaT )
K0 = 0
MaxTimeStep = CEILING( TotalTime / DeltaT )
Q1 = 0.0d0
Q2 = 0.0d0
Q3 = 0.0d0
Q4 = 0.0d0
F1 = 0.0d0
F2 = 0.0d0
F3 = 0.0d0
F4 = 0.0d0
G1 = 0.0d0
G2 = 0.0d0
```

```
    G3 = 0.0d0
    G4 = 0.0d0
    H1 = 0.0d0
    H2 = 0.0d0
    H3 = 0.0d0
    H4 = 0.0d0
    DO Loops = 1, MaxTimeStep
      Pold = P
      !!
      !step- 10.1: transform variables to conservative form
      !  here using parallel computing and store in different
      !      processes
      CALL GetConservativeVariables( Q1, Q2, Q3, Q4, F1, F2, F3, F4, G1, G2, &
    &         G3, G4, H1, H2, H3, H4, Au11, Au12, Au13, Au14, Au21, Au22, &
    &         Au23, Au24, Au31, Au32, Au33, Au34, S1, S2, S3, S4, SAu11, &
    &         SAu12, SAu13, SAu14, SAu21, SAu22, SAu23, SAu24, SAu31, SAu32, &
    &         SAu33, SAu34, U0, V0, W0, P0, ROU0, C0, U, V, W, P, ROU, NA, XN1, &
    &         YN2, ZN3, Jacobi, KexiX, EitaY, TaoZ, MeshX, MeshY, MeshZ, &
    &         Beita, SigmaX1, SigmaX2, SigmaY1, SigmaY2, SigmaZ1, SigmaZ2, &
    &         NPML, N1, N2, N3, PX, PY, PZ, NPX, NPY, NPZ, Loops, DeltaT, ModelFlag )
      !
      ! step- 10.2: get the source term for the inner of solid
!       CALL SolidPMLSource( S1, S2, S3, S4, SAu11, SAu12, SAu13, SAu14, SAu21, &
!   &         SAu22, SAu23, SAu24, SAu31, SAu32, SAu33, SAu34, F1, F2, F3, F4, &
!   &         Au11, Au12, Au13, Au14, Au21, Au22, Au23, Au24, Au31, Au32, Au33, &
!   &         Au34, NA, XN1, YN2, ZN3, MeshX, MeshY, MeshZ, NPML, N1, N2, N3, &
!   &         CutPoints, StartEnd, Centre, DeltaXYZ, R_PML, SolidAfa, SolidSigma0, &
!   &         PX, PY, PZ, ModelFlag )
      !!
      !step- 10.3: LDDRK time-marching( 4/6 marching )
      !!---------------------Algorithm----------------------
      !refer paper('Hu F.Q. Low-dissipation and low-dispersion
      !  runge-kutta schemes for computational acoustics[J]')
      !!--------------------------------------------------
      !here involves to the allocation of processors, and
      !  referring to the storage form of 3-D matrix, we can
      !  have the following algorithm.
      !set processors in Z axis as one picture, so processors in
      !  x and y axis construct a 2-D matrix, then the processor
      !  distribution is as follow
      !!*************************************
      !!*   10   *  11   *  12   *  13   *  14   *
      !!*************************************
      !!*   5  *  6  *  7  *  8  *  9  *
      !!*************************************
      !!*   0  *  1  *  2  *  3  *  4  *
      !!*************************************
      !!MYID >>> coordinate for processors
      !!NPX: the processor number for x direction
      !!NPY: the processor number for y direction
      !!NPZ: the processor number for z direction
      !MYIDCOORDINATE = ( PX, PY, PZ )
```

```
       !write(*,*)px,py,pz, MYLEFT, MYRIGHT, MYFORWARD, MYREAR, MYUPPER, MYLOWER
       Flag = MOD ( Loops, 2 )
       CALL LDDRK( Q1, Q2, Q3, Q4, F1, F2, F3, F4, G1, G2, G3, G4, &
     &    H1, H2, H3, H4, Au11, Au12, Au13, Au14, Au21, Au22, Au23, &
     &      Au24, Au31, Au32, Au33, Au34, S1, S2, S3, S4, SAu11, &
     &      SAu12, SAu13, SAu14, SAu21, SAu22, SAu23, SAu24, SAu31, &
     &      SAu32, SAu33, SAu34, U0, V0, W0, P0, ROU0, C0, Jacobi, KexiX, &
     &      EitaY, TaoZ, MeshX, MeshY, MeshZ, SigmaX1, SigmaX2, SigmaY1, SigmaY2, &
     &      SigmaZ1, SigmaZ2, Beita, NA, XN1, YN2, ZN3, N1, N2, N3, NPML, DeltaT, &
     &      DeltaXYZ, PX, PY, PZ, NPX, NPY, NPZ, MYLEFT, MYRIGHT, MYFORWARD, &
     &      MYREAR, MYUPPER, MYLOWER, XTYPE, YTYPE, ZTYPE, Flag, Loops, &
     &      ModelFlag, BCFlag )
       !!
       !CALL ExchangeInterfaceData( Q1, Q2, Q3, Q4, NA, XN1, YN2, ZN3, &
     !&       MYLEFT, MYRIGHT, MYFORWARD, MYREAR, MYUPPER, MYLOWER, &
     !&         PX, PY, PZ, ModelFlag )
       !
       CALL ExchangeInterfaceDataNew( Q1, NA, XN1, YN2, ZN3, NPML, XTYPE, YTYPE, &
     &        ZTYPE, MYLEFT, MYRIGHT, MYFORWARD, MYREAR, MYUPPER, MYLOWER, &
     &          PX, PY, PZ, NPX, NPY, NPZ, ModelFlag, BCFlag )
       !
       CALL ExchangeInterfaceDataNew( Q2, NA, XN1, YN2, ZN3, NPML, XTYPE, YTYPE, &
     &        ZTYPE, MYLEFT, MYRIGHT, MYFORWARD, MYREAR, MYUPPER, MYLOWER, &
     &          PX, PY, PZ, NPX, NPY, NPZ, ModelFlag, BCFlag )
       !
       CALL ExchangeInterfaceDataNew( Q3, NA, XN1, YN2, ZN3, NPML, XTYPE, YTYPE, &
     &        ZTYPE, MYLEFT, MYRIGHT, MYFORWARD, MYREAR, MYUPPER, MYLOWER, &
     &          PX, PY, PZ, NPX, NPY, NPZ, ModelFlag, BCFlag )
       !
       CALL ExchangeInterfaceDataNew( Q4, NA, XN1, YN2, ZN3, NPML, XTYPE, YTYPE, &
     &        ZTYPE, MYLEFT, MYRIGHT, MYFORWARD, MYREAR, MYUPPER, MYLOWER, &
     &          PX, PY, PZ, NPX, NPY, NPZ, ModelFlag, BCFlag )
       !
       !step- 10.4: transform conservative variables in
       !  original variables, and here
       !!
       CALL GetOriginalVariables( U, V, W, P, ROU, U0, V0, W0, P0, ROU0, C0, &
     &      Q1, Q2, Q3, Q4, NA, XN1, YN2, ZN3, Jacobi, KexiX, EitaY, &
     &        TaoZ, N1, N2, N3, PX, PY, PZ, NPML, ModelFlag )
       !
       !!------------------Algorithm------------------------------
       !here we can efficiently use parallel computing as follows.
       !First category: GATHER all the acoustic field variables
       !  into main processor( MYROOT ), then main processor
       !  computes the effect matrix and interpolate to get
       !  the flow variables near the immersed boundary, finally
       !  solve the body force linear equation group using all
       !  processors with Jacobi iteration method or others.
       !Second category: ALLGATHER the acoustic field variables
       !  into all processors, then get the effect matrix and
       !  interpolate to get the flow variables near the
       !  immersed boundary using all processors, finally solve
```

```
!  the body force linear equation group parallelly.
!For Second category, assuming there has NUMPROCS processors
!  and LN boundary points, the loop distribution should
!  be adopted. That is,
!  <code> DO I = MYID + 1, LN, NUMPROCS
!          get the I-row elements for effect matrix.
!          get the flow variables for Ith boundary points.
!  <code> END DO
!Subroutine CorrectAcousticField relates to solving linear
!  equation group, here we use the lapack parallel computing
!  library which is efficient.
!--------------------------------------------------------
!!
!step- 10.5: get the effect matrix AProcessor and ATotal
CALL GetEffectMatrix( AProcessor, PLN, EffectRange, &
&        Shapes, Cell_S, NormalVector, LN, MeshX, MeshY, &
&           MeshZ, NPML, N1, N2, N3, StartEnd, CutPoints, &
&              DeltaXYZ, MYID, ModelFlag )
!!
!step- 10.6: get the source terms: Delta_U/Delta_T
CALL GetDuDt( DuDt0, DuDtFlag, EffectRange, Shapes, NormalVector, &
&        LN, U, V, W, NA, XN1, YN2, ZN3, MeshX, MeshY, MeshZ, NPML, &
&           N1, N2, N3, DeltaXYZ, DeltaT, PX, PY, PZ, NPX, NPY, &
&              NPZ, MYID, ModelFlag )
!!
!step- 10.7: gather all the elements of matrix A and the DuDt
COUNT = LN * PLN
CALL MPI_ALLGATHER( AProcessor( 1, 1 ), COUNT, MPI_DOUBLE_PRECISION, &
&        ATotal, COUNT, MPI_DOUBLE_PRECISION, MPI_COMM_WORLD, IERR )
AV( 1: LN, 1: LN ) = ATotal( 1: LN, 1: LN )
!
!!
CALL MPI_GATHER( DuDtFlag(1), 1, MPI_INTEGER, RECVCNT, 1, MPI_INTEGER, &
&        MYROOT, MPI_COMM_WORLD, IERR )
!
IF ( MYID == MYROOT ) THEN
  DISPLS(1) = 0
  DO I = 2, NUMPROCS
    DISPLS(I) = DISPLS(I-1) + RECVCNT(I-1)
  END DO
END IF
SENDCNT = DuDtFlag(1)
!
CALL MPI_GATHERV( DuDt0(1), SENDCNT, MPI_DOUBLE_PRECISION, &
&        DuDt1, RECVCNT, DISPLS, MPI_DOUBLE_PRECISION, MYROOT, &
&           MPI_COMM_WORLD, IERR )
!
CALL MPI_GATHERV( DuDtFlag(2), SENDCNT, MPI_INTEGER, DuDtFlagRoot, &
&        RECVCNT, DISPLS, MPI_INTEGER, MYROOT, MPI_COMM_WORLD, IERR )
!
IF ( MYID == MYROOT ) THEN
  ! adjust the sequence of DuDt according to DuDtFlagRoot
```

```
        DO I = 1, LN
          DuDt( DuDtFlagRoot(I) ) = DuDt1( I )
        END DO
      END IF
      !
      ! scattering DuDt to other processors
      CALL MPI_BCAST( DuDt, LN, MPI_DOUBLE_PRECISION, MYROOT, MPI_COMM_WORLD, IERR )
      !!
      !step- 10.8: solve the linear equation group(use SVD or
      !just linear slover?)-----here use ScaLAPACK(parallel)
      !!
      !CALL PLinearSolver( Fn, AV, DuDt, LN )
      CALL ForcingSolver( Du, Dv, Dw, AV, AT, DuDt, MeshX, MeshY, MeshZ, Shapes, &
    &          NormalVector, DeltaXYZ, DeltaT, Cell_S, StartEnd, LN, NPML, &
    &            N1, N2, N3, XN1, YN2, ZN3, PX, PY, PZ, Loops, ModelFlag )
      !!
      !step- 10.9: correct velocity and pressure field
      !   using effect matrix algorithm( IB method )
      !!
      CALL CorrectAcousticField( U, V, W, Du, Dv, Dw, MeshX, MeshY, MeshZ, &
    &            NA, XN1, YN2, ZN3, NPML, N1, N2, N3, PX, PY, PZ, &
    &              EffectRange, LN, ModelFlag )
      !!
      !step- 10.10: change information for the staggered mesh
      !   on the boundary between two processes.
      !!
      !CALL ExchangeInterfaceData( U, V, W, ROU, NA, XN1, YN2, ZN3, &
    !&          MYLEFT, MYRIGHT, MYFORWARD, MYREAR, MYUPPER, MYLOWER, &
    !&            PX, PY, PZ, ModelFlag )
      !
      CALL ExchangeInterfaceDataNew( U, NA, XN1, YN2, ZN3, NPML, XTYPE, YTYPE, &
    &            ZTYPE, MYLEFT, MYRIGHT, MYFORWARD, MYREAR, MYUPPER, MYLOWER, &
    &              PX, PY, PZ, NPX, NPY, NPZ, ModelFlag, BCFlag )
      !
      CALL ExchangeInterfaceDataNew( V, NA, XN1, YN2, ZN3, NPML, XTYPE, YTYPE, &
    &            ZTYPE, MYLEFT, MYRIGHT, MYFORWARD, MYREAR, MYUPPER, MYLOWER, &
    &              PX, PY, PZ, NPX, NPY, NPZ, ModelFlag, BCFlag )
      !
      CALL ExchangeInterfaceDataNew( W, NA, XN1, YN2, ZN3, NPML, XTYPE, YTYPE, &
    &            ZTYPE, MYLEFT, MYRIGHT, MYFORWARD, MYREAR, MYUPPER, MYLOWER, &
    &              PX, PY, PZ, NPX, NPY, NPZ, ModelFlag, BCFlag )
      !
      ! filtering to vanish the surpurious wave
!       CALL Filter( U, V, W, ROU, NA, XN1, YN2, ZN3, ModelFlag )
      !
      !CALL ExchangeInterfaceData( U, V, W, ROU, NA, XN1, YN2, ZN3, &
    !&          MYLEFT, MYRIGHT, MYFORWARD, MYREAR, MYUPPER, MYLOWER, &
    !&            PX, PY, PZ, ModelFlag )
      !
!       CALL ExchangeInterfaceDataNew( U, NA, XN1, YN2, ZN3, NPML, XTYPE, YTYPE, &
!   &            ZTYPE, MYLEFT, MYRIGHT, MYFORWARD, MYREAR, MYUPPER, MYLOWER, &
!   &              PX, PY, PZ, NPX, NPY, NPZ, ModelFlag, BCFlag )
```

```
     !
!     CALL ExchangeInterfaceDataNew( V, NA, XN1, YN2, ZN3, NPML, XTYPE, YTYPE, &
!  &           ZTYPE, MYLEFT, MYRIGHT, MYFORWARD, MYREAR, MYUPPER, MYLOWER, &
!  &              PX, PY, PZ, NPX, NPY, NPZ, ModelFlag, BCFlag )
     !
!     CALL ExchangeInterfaceDataNew( W, NA, XN1, YN2, ZN3, NPML, XTYPE, YTYPE, &
!  &           ZTYPE, MYLEFT, MYRIGHT, MYFORWARD, MYREAR, MYUPPER, MYLOWER, &
!  &              PX, PY, PZ, NPX, NPY, NPZ, ModelFlag, BCFlag )
     !
!     CALL ExchangeInterfaceDataNew( ROU, NA, XN1, YN2, ZN3, NPML, XTYPE, YTYPE, &
!  &           ZTYPE, MYLEFT, MYRIGHT, MYFORWARD, MYREAR, MYUPPER, MYLOWER, &
!  &              PX, PY, PZ, NPX, NPY, NPZ, ModelFlag, BCFlag )
     !
     ! revised the acoustic pressure
!     P = ROU * C0**2
     !!
     !step- 10.11: get the maximum residual between two next
     !  time step and present time step
!     CALL GetMaxResidual( MaxResidual, P, P, P, Pold, Pold, Pold, &
!  &        NA, XN1, YN2, ZN3 )
     !!
     !step- 10.12: reduce to get the maximum residual
!     CALL MPI_REDUCE( MaxResidual, MaxResidual0, 1, MPI_DOUBLE_PRECISION, &
!  &        MPI_MAX, MYROOT, MPI_COMM_WORLD, IERR )
     !!
     !
     ! get the probe pressure for the given points
     IF ( DeltaT * Loops .GE. MaxX ) THEN
        CALL GetProbePressure( ProbePressure0, ProbeFlag, Corner, SideLength, &
   &           RelativePosi, ProbeCoordinate, NumProbe, MeshX, MeshY, MeshZ, &
   &           N1, N2, N3, P, NA, XN1, YN2, ZN3, PX, PY, PZ, NPML, NPX, NPY, &
   &           NPZ, ModelFlag )
        !
        CALL MPI_GATHER( ProbeFlag(1), 1, MPI_INTEGER, RECVCNT, 1, MPI_INTEGER, &
   &           MYROOT, MPI_COMM_WORLD, IERR )
        !
        IF ( MYID == MYROOT ) THEN
           DISPLS(1) = 0
           DO I = 2, NUMPROCS
              DISPLS(I) = DISPLS(I-1) + RECVCNT(I-1)
           END DO
        END IF
        SENDCNT = ProbeFlag(1)
        !
        CALL MPI_GATHERV( ProbePressure0(1), SENDCNT, MPI_DOUBLE_PRECISION, &
   &           ProbePressure1, RECVCNT, DISPLS, MPI_DOUBLE_PRECISION, MYROOT, &
   &              MPI_COMM_WORLD, IERR )
        !
        CALL MPI_GATHERV( ProbeFlag(2), SENDCNT, MPI_INTEGER, ProbeFlagRoot, &
   &           RECVCNT, DISPLS, MPI_INTEGER, MYROOT, MPI_COMM_WORLD, IERR )
        !
        IF ( MYID == MYROOT ) THEN
```

```
          ! adjust the sequence of DuDt according to DuDtFlagRoot
          DO I = 1, SUM(NumProbe)
             ProbePressure2( ProbeFlagRoot(I) ) = ProbePressure1( I )
          END DO
       END IF
       ProbePressure = ProbePressure + ProbePressure2**2
       ProbeTimes = ProbeTimes + 1
    END IF
    !!
    !step- 10.13: print and output in main progress MYROOT
    !  and go to the next loop.
    !  And here we use parallel I/O to output the data to
    !  binary file. And this binary file can be switched
    !  into text file by program: ReadBinary.f90
    IF ( MOD( Loops, Tstep0 ) .EQ. 0 ) THEN
       FLAG0 = 1
       OUTPUTflag = OUTPUTFlag + 1
       IF ( MYID .NE. MYROOT ) THEN
          ! sending data
          CALL MPI_ISEND( P(1,1,1), 1, HTYPE, MYROOT, 100+MYID, &
 &                     MPI_COMM_WORLD, REQ(FLAG0), IERR )
          FLAG0 = FLAG0 + 1
       ELSE
          ! receving data
          DO I = 1, NUMPROCS - 1
             IPZ = I / ( NPX * NPY )
             IPY = ( I - IPZ * NPX * NPY ) / ( NPX )
             IPX = ( I - IPZ * NPX * NPY ) - IPY * NPX
             CALL MPI_IRECV( TotalP( IPX*XN1+1, IPY*YN2+1, IPZ*ZN3+1 ), &
 &                1, VTYPE, I, 100+I, MPI_COMM_WORLD, REQ(FLAG0), IERR )
             FLAG0 = FLAG0 + 1
          END DO
       END IF
       CALL MPI_WAITALL( SIZE0, REQ, STATUS, IERR )
       !
       ! output the total data to file
       IF ( MYID == MYROOT ) THEN
          TotalP( 1:XN1, 1:YN2, 1:ZN3 ) = P( 1:XN1, 1:YN2, 1:ZN3 )
          FLAG = 4
          CALL OutputToTextFile( TotalP, SizeN1, SizeN2, SizeN3, &
 &                           OUTPUTFlag, FLAG, FilenameOUTPUT )
       END IF
    END IF
 END DO
 !
 IF ( MYID == MYROOT ) THEN
    ProbePressure = DSQRT( ProbePressure / ProbeTimes )
    OPEN( UNIT = 11, FILE = TRIM( TRIM( FilenameOUTPUT ) &
 &               // TRIM( 'NumericalProbeAB.dat' ) ) )
    WRITE( UNIT=11, FMT=* ) MultiLN( 1 : NumBody )
    DO I = 1, SUM(NumProbe)
       WRITE( UNIT=11, FMT='(f18.8)' ) ProbePressure(I)
    END DO
```

```fortran
        CLOSE( UNIT=11 )
      END IF
      !!
      !step- 11: return and exit
      CALL MPI_FINALIZE( IERR )
    END PROGRAM AcousticScatteringParallelComputing
    ! ---------------------------------------------------------------------
    ! ---------------------------------------------------------------------
    ! -------------------------SUBROUTINES-----------------------------------
    SUBROUTINE Models( LRef, R_PML0, Shapes, NormalVector, Cell_S, LN, NumBody, &
      &       MultiLN, AOA, Centre, ModelFlag )
      !!
      !-----------------Models--introduction-----------------------
      !This subroutine can get the boundary points, and it can judge-
      !out the dimensions for the analysis( 2-D, 3-D ) using parameter
      !-ModelFlag.( ModelFlag=0: 2-D; ModelFlag=1: 3-D )- 20151202
      ! R_PML0: the minimum width of the original shape which will be
      !   used to get the radius of the solid PML zone
      !-----------------------------------------------------------
      !!
      IMPLICIT NONE
      INTEGER :: I, J, K, LN0
      INTEGER, INTENT(IN):: ModelFlag
      INTEGER, INTENT(IN):: LN
      INTEGER, INTENT(IN):: NumBody
      INTEGER, INTENT(IN):: MultiLN( 1: NumBody )
      REAL(KIND=8), INTENT(INOUT):: Shapes( LN,3 )
      REAL(KIND=8), INTENT(IN) :: AOA, Centre(3)
      REAL(KIND=8), INTENT(OUT) ::  NormalVector( LN,3 )
      REAL(KIND=8), INTENT(OUT) :: LRef(3), R_PML0
      REAL(KIND=8), INTENT(OUT) :: Cell_S(LN)
      REAL(KIND=8):: BodyCentre(3), TempCoor(3)
      REAL(KIND=8):: PI
      PARAMETER( PI = 3.141592653d0 )
      REAL(KIND=8):: ShapeLength(3)
      REAL(KIND=8):: X(2), Y(2), Z(2), R
      !!
      !
      !step- 1.0: get the PML radius for the solid inner
      DO I = 1, 3
        ShapeLength(I) = MAXVAL( Shapes( :, I ) ) - MINVAL( Shapes( :, I ) )
      END DO
      IF ( ModelFlag == 0  ) THEN
        ! 2-D Model
        R_PML0 = MINVAL( ShapeLength( 1 : 2 ) ) / 2.0d0
      ELSE
        R_PML0 = MINVAL( ShapeLength ) / 2.0d0
      END IF
      !!
      !step- 2: get the body centre
      DO I = 1, 3
        BodyCentre( I ) = ( MAXVAL( Shapes( : , I ) ) + &
```

```
    &     MINVAL( Shapes( : , I ) ) ) / 2.0d0
    END DO
    !!
    !step- 3: rotate body for AOA degree
    DO I = 1, LN
      CALL RotateBody( TempCoor, Shapes( I, 1 : 3 ), BodyCentre, AOA )
      DO J = 1, 3
        Shapes( I, J ) = TempCoor( J )
      END DO
    END DO
    !!
    DO J = 1, 3
      DO I = 1, LN
        Shapes( I, J ) = Shapes( I, J ) + Centre( J ) - BodyCentre( J )
      END DO
    END DO
    !!
    !step- 4: get the cell area: Cell_S and normal vector: NormalVector
!   Cell_S = ( MAXVAL( Shapes( :, 1 ) ) - MINVAL( Shapes( :, 1 ) ) ) * 0.5d0 * &
!   &          2.0d0 * PI / ( 1.0d0 * LN )
    DO J = 1, NumBody
      DO I = 1, MultiLN( J )
        IF ( J == 1 ) THEN
          IF ( I == 1 ) THEN
            Cell_S( I ) = DSQRT( (Shapes(I,1)-Shapes(I+1,1))**2 + &
    &                            (Shapes(I,2)-Shapes(I+1,2))**2 + &
    &                            (Shapes(I,3)-Shapes(I+1,3))**2 )
            X(1) = Shapes(MultiLN(J),1)
            Y(1) = Shapes(MultiLN(J),2)
            Z(1) = Shapes(MultiLN(J),3)
            X(2) = Shapes(I+1,1)
            Y(2) = Shapes(I+1,2)
            Z(2) = Shapes(I+1,3)
            R = DSQRT( (X(1)-X(2))**2+(Y(1)-Y(2))**2+(Z(1)-Z(2))**2 )
            NormalVector( I, 1 ) = (Y(2)-Y(1)) / R
            NormalVector( I, 2 ) = -(X(2)-X(1)) / R
            NormalVector( I, 3 ) = 0.0d0
          ELSEIF ( I == MultiLN(J) ) THEN
            Cell_S( I ) = DSQRT( (Shapes(I,1)-Shapes(1,1))**2 + &
    &                            (Shapes(I,2)-Shapes(1,2))**2 + &
    &                            (Shapes(I,3)-Shapes(1,3))**2 )
            X(1) = Shapes(I-1,1)
            Y(1) = Shapes(I-1,2)
            Z(1) = Shapes(I-1,3)
            X(2) = Shapes(1,1)
            Y(2) = Shapes(1,2)
            Z(2) = Shapes(1,3)
            R = DSQRT( (X(1)-X(2))**2+(Y(1)-Y(2))**2+(Z(1)-Z(2))**2 )
            NormalVector( I, 1 ) = (Y(2)-Y(1)) / R
            NormalVector( I, 2 ) = -(X(2)-X(1)) / R
            NormalVector( I, 3 ) = 0.0d0
          ELSE
            Cell_S( I ) = DSQRT( (Shapes(I,1)-Shapes(I+1,1))**2 + &
```

```
     &                    (Shapes(I,2)-Shapes(I+1,2))**2 + &
     &                    (Shapes(I,3)-Shapes(I+1,3))**2 )
            X(1) = Shapes(I-1,1)
            Y(1) = Shapes(I-1,2)
            Z(1) = Shapes(I-1,3)
            X(2) = Shapes(I+1,1)
            Y(2) = Shapes(I+1,2)
            Z(2) = Shapes(I+1,3)
            R = DSQRT( (X(1)-X(2))**2+(Y(1)-Y(2))**2+(Z(1)-Z(2))**2 )
            NormalVector( I, 1 ) = (Y(2)-Y(1)) / R
            NormalVector( I, 2 ) = -(X(2)-X(1)) / R
            NormalVector( I, 3 ) = 0.0d0
          END IF
        ELSE
          K = SUM( MultiLN(1:(J-1)) )
          IF ( I == 1 ) THEN
            Cell_S( I+K ) = DSQRT( (Shapes(I+K,1)-Shapes(1+K+I,1))**2 + &
     &                    (Shapes(I+K,2)-Shapes(1+k+I,2))**2 + &
     &                    (Shapes(I+K,3)-Shapes(1+K+I,3))**2 )
            X(1) = Shapes(MultiLN(J)+K,1)
            Y(1) = Shapes(MultiLN(J)+K,2)
            Z(1) = Shapes(MultiLN(J)+K,3)
            X(2) = Shapes(I+1+K,1)
            Y(2) = Shapes(I+1+K,2)
            Z(2) = Shapes(I+1+K,3)
            R = DSQRT( (X(1)-X(2))**2+(Y(1)-Y(2))**2+(Z(1)-Z(2))**2 )
            NormalVector( I+K, 1 ) = (Y(2)-Y(1)) / R
            NormalVector( I+K, 2 ) = -(X(2)-X(1)) / R
            NormalVector( I+K, 3 ) = 0.0d0
          ELSEIF ( I == MultiLN(J) ) THEN
            Cell_S( I+K ) = DSQRT( (Shapes(I+K,1)-Shapes(K+1,1))**2 + &
     &                    (Shapes(I+K,2)-Shapes(1+k,2))**2 + &
     &                    (Shapes(I+K,3)-Shapes(1+K,3))**2 )
            X(1) = Shapes(K+I-1,1)
            Y(1) = Shapes(K+I-1,2)
            Z(1) = Shapes(K+I-1,3)
            X(2) = Shapes(K+1,1)
            Y(2) = Shapes(K+1,2)
            Z(2) = Shapes(K+1,3)
            R = DSQRT( (X(1)-X(2))**2+(Y(1)-Y(2))**2+(Z(1)-Z(2))**2 )
            NormalVector( I+K, 1 ) = (Y(2)-Y(1)) / R
            NormalVector( I+K, 2 ) = -(X(2)-X(1)) / R
            NormalVector( I+K, 3 ) = 0.0d0
          ELSE
            Cell_S( I+K ) = DSQRT( (Shapes(I+K,1)-Shapes(1+K+I,1))**2 + &
     &                    (Shapes(I+K,2)-Shapes(1+k+I,2))**2 + &
     &                    (Shapes(I+K,3)-Shapes(1+K+I,3))**2 )
            X(1) = Shapes(K+I-1,1)
            Y(1) = Shapes(K+I-1,2)
            Z(1) = Shapes(K+I-1,3)
            X(2) = Shapes(K+I+1,1)
            Y(2) = Shapes(K+I+1,2)
            Z(2) = Shapes(K+I+1,3)
```

```
                R = DSQRT( (X(1)-X(2))**2+(Y(1)-Y(2))**2+(Z(1)-Z(2))**2 )
                NormalVector( I+K, 1 ) = (Y(2)-Y(1)) / R
                NormalVector( I+K, 2 ) = -(X(2)-X(1)) / R
                NormalVector( I+K, 3 ) = 0.0d0
              END IF
            END IF
          END DO
        END DO
        !!
        !step- 5: get the normal vector for immersed boundary
!       DO I = 1, LN
!         NormalVector( I, 1 ) = DCOS( ( I - 1 ) * 2.0d0 * PI / ( 1.0d0 * LN ) )
!         NormalVector( I, 2 ) = DSIN( ( I - 1 ) * 2.0d0 * PI / ( 1.0d0 * LN ) )
!         NormalVector( I, 3 ) = 0.0d0
!       END DO
        !!
        !step- 5: get the characteristic length for the obstacle
        DO I = 1, 3
          LRef( I ) = ( MAXVAL( Shapes( :, I ) ) - MINVAL( Shapes( :, I ) ) )
        END DO
        !
        END SUBROUTINE Models
        !!
        !--------------------------------------SUBROUTINE----------------------
        SUBROUTINE RotateBody( B, A, Centre, AOA )
        IMPLICIT NONE
        INTEGER:: I, J
        REAL(KIND=8):: A(3), B(3), Centre(3)
        REAL(KIND=8):: A1(3), B1(3)
        REAL(KIND=8):: AOA, AOA0, PI
        PARAMETER( PI = 3.141592653d0 )
        !!
        DO I = 1, 2
          A1(I) = A(I) - Centre(I)
        END DO
        !!
        AOA0 = AOA
        AOA0 = AOA0 * PI/180d0
        !!
        IF ( A1(1) > 0.0d0 ) THEN
          IF ( AOA0 < 0.0d0 ) THEN
            !clockwise rotating
            B1(1) = ( A1(1)**2 * DCOS( AOA0 ) + A1(2) * &
        &          DABS( A1(1) * DSIN( AOA0 ) ) )/A1(1)
            B1(2) = A1(2) * DCOS( AOA0 ) - DABS( A1(1) * DSIN( AOA0 ) )
          ELSE
            !anticlockwise rotating
            B1(1) = ( A1(1)**2 * DCOS( AOA0 ) - A1(2) * &
        &          DABS( A1(1) * DSIN( AOA0 ) ) )/A1(1)
            B1(2) = A1(2) * DCOS( AOA0 ) + DABS( A1(1) * DSIN( AOA0 ) )
          END IF
        ELSE
```

```fortran
         IF ( AOA0 < 0.0d0 ) THEN
            !clockwise rotating
            B1(1) = ( A1(1)**2 * DCOS( AOA0 ) - A1(2) * &
      &           DABS( A1(1) * DSIN( AOA0 ) ) )/A1(1)
            B1(2) = A1(2) * DCOS( AOA0 ) + DABS( A1(1) * DSIN( AOA0 ) )
         ELSE
            !anticlockwise rotating
            B1(1) = ( A1(1)**2 * DCOS( AOA0 ) + A1(2) * &
      &           DABS( A1(1) * DSIN( AOA0 ) ) )/A1(1)
            B1(2) = A1(2) * DCOS( AOA0 ) - DABS( A1(1) * DSIN( AOA0 ) )
         END IF
      END IF
      B(1) = B1(1) + Centre(1)
      B(2) = B1(2) + Centre(2)
      B(3) = A(3)
   END SUBROUTINE RotateBody
   ! ---------------------------------------------------
   ! >>>>>>>
   SUBROUTINE GetMeshSize( N1, N2, N3, StartEnd, CutPoints, MinDxDyDz, &
      &     R_PML, R_PML0, Shapes, Cell_S, LN, Centre, MaxX, MaxY, &
      &        MaxZ, SolidRelativePosi, q0, LRef, NPML, NPX, NPY, &
      &           NPZ, ModelFlag, MYID, FilenameOUTPUT )
      !!-----------------GetMeshSize-Introduction-------------------
      !   ModelFlag = 0:   2-D
      !   ModelFlag = 1:   3-D
      !-------------------------------------------------------
      IMPLICIT NONE
      INTEGER:: I, J
      INTEGER, INTENT(IN):: NPML, MYID
      INTEGER, INTENT(IN):: NPX, NPY, NPZ
      INTEGER, INTENT(IN):: LN, ModelFlag
      CHARACTER(LEN=80):: FilenameOUTPUT
      REAL(KIND=8), INTENT(IN):: R_PML0
      REAL(KIND=8), INTENT(IN):: LRef(3), MaxX, MaxY, MaxZ, SolidRelativePosi(3)
      REAL(KIND=8), INTENT(IN):: q0( 3, 2 )
      REAL(KIND=8), INTENT(IN):: Centre(3), Cell_S(LN)
      REAL(KIND=8), INTENT(INOUT):: Shapes( LN, 3 )
      INTEGER, INTENT(OUT):: N1, N2, N3, StartEnd( 3, 3 )
      REAL(KIND=8), INTENT(OUT):: CutPoints( 3, 4 ), MinDxDyDz
      REAL(KIND=8), INTENT(OUT):: R_PML
      REAL(KIND=8):: L(3), Length(3), TempCentre(3)
      !!
      !
      !step- 1: get mesh size for Cartesian mesh
      MinDxDyDz = MAXVAL( Cell_S ) * 1.0d0
      Length(1) = MaxX
      Length(2) = MaxY
      Length(3) = MaxZ
      R_PML = R_PML0 - 5.0d0 * MinDxDyDz
      !!
      DO I = 1, 3
         L(I) = SolidRelativePosi(I) - LRef(I)/2.0d0 - 10.0d0 * MinDxDyDz
      END DO
```

```
IF ( ModelFlag == 0 ) THEN
   L(3) = 0.0d0
END IF
!!
DO I = 1, 3
   CutPoints( I, 1 ) = 0.0d0 - SolidRelativePosi(I)
   CutPoints( I, 2 ) = L(I) - SolidRelativePosi(I)
   CutPoints( I, 3 ) = CutPoints( I, 2 ) + MinDxDyDz * &
&       CEILING( ( LRef(I) + 20.0d0 * MinDxDyDz )/MinDxDyDz )
END DO
IF ( ModelFlag == 0 ) THEN
   CutPoints( 3, 1:3 ) = Centre(3)
END IF
!
DO I = 1, 3
   CutPoints( I, 4 ) = Length(I) - SolidRelativePosi(I)
END DO
!!
IF ( q0(1,1) == 1.0d0 ) THEN
   ! uniform grid for the first section of x direction
   StartEnd( 1, 1 ) = 1 + CEILING( L(1) / MinDxDyDz )
ELSE
   StartEnd( 1, 1 ) = CEILING( 1 + DLOG( 1.0d0 - L(1) * &
&          ( 1.0d0 - q0(1,1) ) / ( q0(1,1) * MinDxDyDz ) ) &
&              / DLOG( q0(1,1) ) )
END IF
!
StartEnd( 1, 2 ) = CEILING( ( LRef(1) + 20.0d0 * MinDxDyDz )/ &
&       MinDxDyDz ) + StartEnd( 1, 1 )
!
IF ( q0(1,2) == 1.0d0 ) THEN
   ! uniform grid for the second section of x direction
   StartEnd( 1, 3 ) = StartEnd(1,2) + CEILING( ( CutPoints( 1, 4 ) &
&       -CutPoints( 1, 3 ) ) / MinDxDyDz )
ELSE
   StartEnd( 1, 3 ) = CEILING( DLOG( 1.0d0 - ( CutPoints( 1, 4 ) &
&       -CutPoints( 1, 3 ) ) * ( 1.0d0 - q0(1,2) )/ &
&       ( q0(1,2) * MinDxDyDz ) )/DLOG( q0(1,2) ) ) + StartEnd( 1, 2 )
END IF
StartEnd( 1, 3 ) = CEILING( ( StartEnd( 1, 3 ) + 2 * NPML ) * 1.0d0 &
&                        / NPX ) * NPX - 2 * NPML
!!
IF ( q0(1,1) == 1.0d0 ) THEN
   ! uniform grid
   CutPoints( 1, 2 ) = CutPoints( 1, 1 ) +  ( StartEnd( 1, 1 ) - 1 ) * MinDxDyDz
ELSE
   CutPoints( 1, 2 ) = CutPoints( 1, 1 ) +  ( 1.0d0 - q0(1,1)** &
&       ( StartEnd( 1, 1 ) - 1 ) ) * q0(1,1) * MinDxDyDz/( 1 - q0(1,1) )
END IF
!!
CutPoints( 1, 3 ) = CutPoints( 1, 2 ) + MinDxDyDz * &
&       ( StartEnd( 1, 2 ) - StartEnd( 1, 1 ) )
!!
```

```fortran
IF ( q0(1,2) == 1.0d0 ) THEN
   ! uniform grid
   CutPoints( 1, 4 ) = CutPoints( 1, 3 ) + ( StartEnd( 1, 3 ) &
&      - StartEnd( 1, 2 ) ) * MinDxDyDz
ELSE
   CutPoints( 1, 4 ) = CutPoints( 1, 3 ) + ( 1.0d0-q0(1,2)**( StartEnd( 1, 3 ) &
&      - StartEnd( 1, 2 ) ) ) * q0(1,2) * MinDxDyDz/( 1 - q0(1,2) )
END IF
!!
DO I = 2, 3
   ! use stretching mesh for y&z direction.
   IF ( q0(I,1) == 1.0d0 ) THEN
      ! using uniform grid for y&z direction
      StartEnd( I, 1 ) = 1 + CEILING( L(I) / MinDxDyDz )
   ELSE
      StartEnd( I, 1 ) = CEILING( 1.0d0 + DLOG( 1.0d0 - L(I) * &
&         ( 1.0d0 - q0(I,1) ) / ( q0(I,1) * MinDxDyDz ) )/DLOG( q0(I,1) ) )
   END IF
   !!
   StartEnd( I, 2 ) = CEILING( ( LRef(I) + 20.0d0 * MinDxDyDz )/MinDxDyDz ) &
&         + StartEnd(I,1)
   !
   IF ( q0(I,2) == 1.0d0 ) THEN
      ! uniform grid for y&z direction
      StartEnd( I, 3 ) = StartEnd(I,2) + CEILING( ( CutPoints(I,4) - CutPoints(I,3) ) &
&                        / MinDxDyDz )
   ELSE
      StartEnd( I, 3 ) = CEILING( DLOG( 1.0d0 - ( CutPoints(I,4) - CutPoints(I,3) ) &
&         * ( 1.0d0 - q0(I,2) ) / ( q0(I,2) * MinDxDyDz ) ) / DLOG( q0(I,2) ) ) &
&            + StartEnd(I,2)
   END IF
   !!
   ! Normalization
   IF ( I == 2 ) THEN
      StartEnd( I, 3 ) = CEILING( ( StartEnd( I, 3 ) + 2 * NPML ) &
&            * 1.0d0 / NPY ) * NPY - 2 * NPML
   ELSE
      StartEnd( I, 3 ) = CEILING( ( StartEnd( I, 3 ) + 2 * NPML ) &
&            * 1.0d0 / NPZ ) * NPZ - 2 * NPML
   END IF
   !!
   IF ( q0(I,1) == 1.0d0 ) THEN
      ! uniform grid
      CutPoints( I, 2 ) = CutPoints(I,1) + ( StartEnd(I,1) - 1 ) * MinDxDyDz
   ELSE
      CutPoints( I, 2 ) = CutPoints(I,1) + ( 1.0d0 - q0(I,1)** &
&         ( StartEnd(I,1) - 1 ) ) * q0(I,1) * MinDxDyDz/( 1 - q0(I,1) )
   END IF
   !!
   CutPoints( I, 3 ) = CutPoints(I,2) + MinDxDyDz * ( StartEnd(I,2) - &
&         StartEnd(I,1) )
   !!
   IF ( q0(I,2) == 1.0d0 ) THEN
```

```fortran
        ! uniform grid
        CutPoints( I, 4 ) = CutPoints(I,3) + ( StartEnd(I,3) - &
    &         StartEnd(I,2) ) * MinDxDyDz
      ELSE
        CutPoints( I, 4 ) = CutPoints(I,3) + ( 1.0d0 - q0(I,2)** &
    &         ( StartEnd(I,3) - StartEnd(I,2) ) ) * q0(I,2) * &
    &             MinDxDyDz / ( 1 - q0(I,2) )
      END IF
    END DO
    !!
    IF ( ModelFlag == 0 ) THEN
      StartEnd( 3, : ) = 1
      CutPoints( 3, : ) = Centre(3)
    END IF
    !!
    N1 = StartEnd( 1, 3 )
    N2 = StartEnd( 2, 3 )
    N3 = StartEnd( 3, 3 )
    DO I = 1, 3
      TempCentre(I) = ( CutPoints( I, 2 ) + CutPoints( I, 3 ) ) / 2.0d0
      CutPoints( I, : ) = CutPoints( I, : ) - TempCentre(I)
    END DO
    !
    DO J = 1, 3
      DO I = 1, LN
        Shapes( I, J ) = Shapes( I, J ) - Centre(J)
      END DO
    END DO
    !
    !!
    IF ( MYID == 0 ) THEN
      OPEN( UNIT =10, FILE = TRIM( TRIM( FilenameOUTPUT )//TRIM( 'ShapesNew.dat' ) ) )
      IF( ModelFlag == 0 ) THEN
        DO I = 1, LN
          WRITE( UNIT = 10, FMT = '(2F18.8)' )Shapes( I, 1:2 )
        END DO
      ELSE
        DO I = 1, LN
          WRITE( UNIT = 10, FMT = '(3F18.8)' )Shapes( I, 1:3 )
        END DO
      END IF
      CLOSE( UNIT = 10 )
    END IF
  END SUBROUTINE GetMeshSize
  ! -----------------------------
  ! >>>>>>>>>>
  SUBROUTINE Mesh( MeshX, MeshY, MeshZ, DeltaX, DeltaY, DeltaZ, N1, &
    &   N2, N3, NPML, StartEnd, CutPoints, DeltaXYZ, q0, &
    &       ModelFlag, MYID, FilenameOUTPUT )
    IMPLICIT NONE
    INTEGER:: I, J, SizeXYZ(3)
    INTEGER, INTENT(IN):: MYID
    INTEGER, INTENT(IN):: ModelFlag
```

```
INTEGER, INTENT(IN):: N1, N2, N3, NPML, StartEnd( 3, 3 )
REAL(KIND=8), INTENT(IN):: CutPoints( 3, 4 ), DeltaXYZ
REAL(KIND=8), INTENT(IN):: q0(3,2)
CHARACTER(LEN=80):: FilenameOUTPUT
REAL(KIND=8), INTENT(OUT):: MeshX( -(NPML-1) : ( N1 + NPML ) )
REAL(KIND=8), INTENT(OUT):: MeshY( -(NPML-1) : ( N2 + NPML ) )
REAL(KIND=8), INTENT(OUT):: MeshZ( -(NPML-1) : ( N3 + NPML ) )
REAL(KIND=8), INTENT(OUT):: DeltaX( -(NPML-1) : ( N1 + NPML - 1 ) )
REAL(KIND=8), INTENT(OUT):: DeltaY( -(NPML-1) : ( N2 + NPML - 1 ) )
REAL(KIND=8), INTENT(OUT):: DeltaZ( -(NPML-1) : ( N3 + NPML - 1 ) )
!!
SizeXYZ(1) = N1
SizeXYZ(2) = N2
SizeXYZ(3) = N3
MeshX(1) = CutPoints( 1, 1 )
MeshY(1) = CutPoints( 2, 1 )
MeshZ(1) = CutPoints( 3, 1 )
!!
!x- direction mesh points
DO I = - ( NPML - 1 ), ( N1 + NPML - 1 )
   IF ( I < StartEnd( 1, 1 ) ) THEN
      DeltaX(I) = DeltaXYZ * q0(1,1)**( StartEnd(1,1) - I )
   ELSEIF ( I < StartEnd( 1, 2 ) ) THEN
      DeltaX(I) = DeltaXYZ
   ELSE
      DeltaX(I) = DeltaXYZ * q0(1,2)**( I - StartEnd(1,2) + 1 )
   END IF
   !MeshX(I+1) = MeshX(I) + DeltaX(I)
END DO
!
DO I = 1, ( N1 + NPML - 1 )
   MeshX( I + 1 ) = MeshX(I) + DeltaX(I)
END DO
!
DO I = 1, - ( NPML - 2 ), - 1
   MeshX( I - 1 ) = MeshX(I) - DeltaX(I - 1)
END DO
!!
!get y-direction mesh points
J = 2
DO I = - ( NPML - 1 ) , ( N2 + NPML - 1 )
   IF ( I < StartEnd(J,1) ) THEN
      DeltaY(I) = DeltaXYZ * q0(2,1)**( StartEnd(J,1) - I )
   ELSEIF ( I < StartEnd(J,2) ) THEN
      DeltaY(I) = DeltaXYZ
   ELSE
      DeltaY(I) = DeltaXYZ * q0(2,2)**( I - StartEnd(J,2) + 1 )
   END IF
   !MeshY(I + 1) = MeshY(I) + DeltaY(I)
END DO
!
DO I = 1, ( N2 + NPML - 1 )
   MeshY( I + 1 ) = MeshY(I) + DeltaY(I)
```

```
    END DO
    !
    DO I = 1, - ( NPML - 2 ), -1
      MeshY( I - 1 ) = MeshY(I) - DeltaY( I - 1 )
    END DO
    !!
    !get Z-direction mesh points
    J = 3
    IF ( ModelFlag == 0 ) THEN
      !   2-D model
      DeltaZ = 0.0d0
      MeshZ = 0.0d0
    ELSE
      !   3-D model
      DO I = - ( NPML - 1 ) , ( N3 + NPML - 1 )
        IF ( I < StartEnd( J,1 ) ) THEN
          DeltaZ(I) = DeltaXYZ * q0(3,1)**( StartEnd(J,1) - I )
        ELSEIF ( I < StartEnd(J,2) ) THEN
          DeltaZ(I) = DeltaXYZ
        ELSE
          DeltaZ(I) = DeltaXYZ * q0(3,2)**( I - StartEnd(J,2) + 1 )
        END IF
        !MeshZ(I + 1) = MeshZ(I) + DeltaZ(I)
      END DO
      !
      DO I = 1, ( N3 + NPML - 1 )
        MeshZ( I + 1 ) = MeshZ(I) + DeltaZ(I)
      END DO
      !
      DO I = 1, - ( NPML - 2 ), - 1
        MeshZ( I - 1 ) = MeshZ(I) - DeltaZ( I - 1 )
      END DO
    END IF
    !
    !output mesh data to external file
    IF ( MYID == 0 ) THEN
      OPEN( UNIT = 10, FILE = TRIM( TRIM( FilenameOUTPUT )//TRIM( 'MeshX.dat' ) ) )
      DO I = -(NPML-1) , ( N1 + NPML )
        WRITE( UNIT=10, FMT= '(f18.8)' ) MeshX(I)
      END DO
      CLOSE(UNIT = 10)
      OPEN( UNIT = 10, FILE = TRIM( TRIM( FilenameOUTPUT )//TRIM( 'MeshY.dat' ) ) )
      DO I = -(NPML-1) , ( N2 + NPML )
        WRITE( UNIT = 10, FMT = '( f18.8 )' ) MeshY(I)
      END DO
      CLOSE(UNIT = 10)
      OPEN( UNIT = 10, FILE = TRIM( TRIM( FilenameOUTPUT )//TRIM( 'MeshZ.dat' ) ) )
      DO I = -(NPML-1) , ( N3 + NPML )
        WRITE( UNIT = 10, FMT = '( f18.8 )' ) MeshZ(I)
      END DO
      CLOSE(UNIT = 10)
    END IF
  END SUBROUTINE Mesh
```

```
! ----------------------------------------------
! >>>>>>>>>>>>>>>>>>
SUBROUTINE NumericalProbe( Corner, SideLength, RelativePosi, ProbeCoordinate, &
  &          NumProbe, Rp, ProbeAngle, NormalVector, Shapes, MeshX, MeshY, MeshZ, &
  &              DeltaXYZ, N1, N2, N3, NPML, ModelFlag )
  IMPLICIT NONE
  INTEGER:: I, J, K, L
  INTEGER, INTENT(IN):: ModelFlag
  INTEGER, INTENT(IN):: NumProbe(2)
  INTEGER, INTENT(IN):: N1, N2, N3, NPML
  REAL(KIND=8), INTENT(IN):: DeltaXYZ
  REAL(KIND=8), INTENT(IN):: ProbeAngle(NumProbe(2)), Rp
  REAL(KIND=8), INTENT(IN):: MeshX( -(NPML-1) : (N1+NPML) )
  REAL(KIND=8), INTENT(IN):: MeshY( -(NPML-1) : (N2+NPML) )
  REAL(KIND=8), INTENT(IN):: MeshZ( -(NPML-1) : (N3+NPML) )
  REAL(KIND=8), INTENT(IN):: NormalVector( NumProbe(1), 3 )
  REAL(KIND=8), INTENT(IN):: Shapes( NumProbe(1), 3 )
  REAL(KIND=8), INTENT(INOUT):: ProbeCoordinate( SUM(NumProbe), 3 )
  INTEGER, INTENT(OUT):: Corner( SUM(NumProbe), 3 )
  REAL(KIND=8), INTENT(OUT):: SideLength( SUM(NumProbe), 3 )
  REAL(KIND=8), INTENT(OUT):: RelativePosi( SUM(NumProbe), 3 )
  REAL(KIND=8):: PI
  PARAMETER( PI = 3.141592653d0 )
  !
!   DO I = 1, NumProbe
!      ProbeCoordinate( I, 1 ) = Rp * DCOS( ProbeAngle(I) * PI / 180.0d0 )
!      ProbeCoordinate( I, 2 ) = Rp * DSIN( ProbeAngle(I) * PI / 180.0d0 )
!      IF ( ModelFlag == 0 ) THEN
!         ProbeCoordinate( I, 3 ) = 0.0d0
!      ELSE
!         ProbeCoordinate( I, 3 ) = 0.0d0
!      END IF
!   END DO
  !
  ! get the coordinate for the analysis model surface
  DO I = 1, NumProbe(1)
     ProbeCoordinate( I, 1 ) = Shapes( I, 1 ) + 0.5d0*DeltaXYZ * NormalVector( I, 1 )
     ProbeCoordinate( I, 2 ) = Shapes( I, 2 ) + 0.5d0*DeltaXYZ * NormalVector( I, 2 )
     ProbeCoordinate( I, 3 ) = Shapes( I, 3 ) + 0.5d0*DeltaXYZ * NormalVector( I, 3 )
  END DO
  !
  ! get the far field numerical probe coordinate
  ProbeCoordinate( (1+NumProbe(1)):SUM(NumProbe), 1 ) = MeshX( 1 : N1 )
  ProbeCoordinate( (1+NumProbe(1)):SUM(NumProbe), 2 ) = 0.0d0
  ProbeCoordinate( (1+NumProbe(1)):SUM(NumProbe), 3 ) = 0.0d0
  !
  DO L = 1, SUM(NumProbe)
     DO I = - ( NPML - 1 ), N1 + NPML - 1
        IF ( ( MeshX(I) - ProbeCoordinate( L, 1 ) ) * &
  &           ( MeshX(I+1) - ProbeCoordinate( L, 1 ) ) <= 0 ) THEN
           Corner( L, 1 ) = I
           SideLength( L, 1 ) = MeshX(I+1) - MeshX(I)
```

```
            RelativePosi( L, 1 ) = ProbeCoordinate( L, 1 ) - MeshX(I)
          END IF
        END DO
        !
        DO I = - ( NPML - 1 ), N2 + NPML - 1
          IF ( ( MeshY(I) - ProbeCoordinate( L, 2 ) ) * &
     &          ( MeshY(I+1) - ProbeCoordinate( L, 2 ) ) <= 0 ) THEN
            Corner( L, 2 ) = I
            SideLength( L, 2 ) = MeshY(I+1) - MeshY(I)
            RelativePosi( L, 2 ) = ProbeCoordinate( L, 2 ) - MeshY(I)
          END IF
        END DO
        !
        IF ( ModelFlag == 1 ) THEN
          DO I = - ( NPML - 1 ), N3 + NPML - 1
            IF ( ( MeshZ(I) - ProbeCoordinate( L, 3 ) ) * &
     &            ( MeshZ(I+1) - ProbeCoordinate( L, 3 ) ) <= 0 ) THEN
              Corner( L, 3 ) = I
              SideLength( L, 3 ) = MeshZ(I+1) - MeshZ(I)
              RelativePosi( L, 3 ) = ProbeCoordinate( L, 3 ) - MeshZ(I)
            END IF
          END DO
        END IF
      END DO
    END SUBROUTINE NumericalProbe
    !
    !SUBROUTINE GetProbePressure( ProbePressure, Corner, SideLength, RelativePosi, &
    !  &          NumProbe, P, NA, XN1, YN2, ZN3, PX, PY, PZ, NPML, ModelFlag )
    !  IMPLICIT NONE
    !  INTEGER:: I, J, L, I0, J0, K0, I1, J1, K1
    !  INTEGER, INTENT(IN):: NumProbe
    !  INTEGER, INTENT(IN):: PX, PY, PZ, NPML
    !  INTEGER, INTENT(IN):: ModelFlag
    !  INTEGER, INTENT(IN):: NA, XN1, YN2, ZN3
    !  INTEGER, INTENT(IN):: Corner( NumProbe, 3 )
    !  REAL(KIND=8), INTENT(IN):: SideLength( NumProbe, 3 )
    !  REAL(KIND=8), INTENT(IN):: RelativePosi( NumProbe, 3 )
    !  REAL(KIND=8), INTENT(IN):: P( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
    !  REAL(KIND=8), INTENT(OUT):: ProbePressure( NumProbe )
    !  REAL(KIND=8):: X2, Y2, Z2
    !  !
    !  I1 = PX * XN1
    !  J1 = PY * YN2
    !  K1 = PZ * ZN3
    !  DO L = 1, NumProbe
    !    X2 = SideLength( L, 1 ) - RelativePosi( L, 1 )
    !    Y2 = SideLength( L, 2 ) - RelativePosi( L, 2 )
    !    IF ( ModelFlag == 0 ) THEN
    !      ! 2-D Model
    !      I0 = Corner( L, 1 ) + NPML - I1
    !      J0 = Corner( L, 2 ) + NPML - J1
    !      k0 = Corner( L, 3 ) + ModelFlag * ( NPML - K1 )
```

```
!          ProbePressure(L) = ( X2 * Y2 * P( I0, J0, 1 ) + RelativePosi( L, 1 ) * Y2 * P( I0+1, J0, 1 ) &
!    &          + RelativePosi( L, 1 ) * RelativePosi( L, 2 ) * P( I0+1, J0+1, 1 ) + &
!    &             X2 * RelativePosi( L, 2 ) * P( I0, J0+1, 1 ) ) / &
!    &                ( SideLength( L, 1 ) * SideLength( L, 2 ) )
!
!       ELSE
!          ! 3-D Model
!
!       END IF
!    END DO
!END SUBROUTINE GetProbePressure
!
!
SUBROUTINE GetProbePressure( ProbePressure, ProbeFlag, Corner, SideLength, &
   &          RelativePosi, ProbeCoordinate, NumProbe, MeshX, MeshY, MeshZ, &
   &          N1, N2, N3, P, NA, XN1, YN2, ZN3, PX, PY, PZ, NPML, NPX, NPY, &
   &          NPZ, ModelFlag )
   IMPLICIT NONE
   INTEGER:: I, J, L, I0, J0, K0, I1, J1, K1
   INTEGER, INTENT(IN):: NumProbe(2)
   INTEGER, INTENT(IN):: PX, PY, PZ, NPML
   INTEGER, INTENT(IN):: ModelFlag
   INTEGER, INTENT(IN):: NA, XN1, YN2, ZN3
   INTEGER, INTENT(IN):: NPX, NPY, NPZ
   INTEGER, INTENT(IN):: N1, N2, N3
   INTEGER, INTENT(IN):: Corner( SUM(NumProbe), 3 )
   REAL(KIND=8), INTENT(IN):: ProbeCoordinate( SUM(NumProbe), 3 )
   REAL(KIND=8), INTENT(IN):: MeshX( -(NPML-1):(N1+NPML) )
   REAL(KIND=8), INTENT(IN):: MeshY( -(NPML-1):(N2+NPML) )
   REAL(KIND=8), INTENT(IN):: MeshZ( -(NPML-1):(N3+NPML) )
   REAL(KIND=8), INTENT(IN):: SideLength( SUM(NumProbe), 3 )
   REAL(KIND=8), INTENT(IN):: RelativePosi( SUM(NumProbe), 3 )
   REAL(KIND=8), INTENT(IN):: P( -(NA-1):(XN1+NA), -(NA-1):(YN2+NA), -(NA-1):(ZN3+NA) )
   REAL(KIND=8), INTENT(OUT):: ProbePressure( SUM(NumProbe) )
   INTEGER, INTENT(OUT):: ProbeFlag( SUM(NumProbe)+1 )
   REAL(KIND=8):: X2, Y2, Z2
   REAL(KIND=8):: A(2), B(2), C(2)
   INTEGER:: Flag0
   !
   I1 = PX * XN1
   J1 = PY * YN2
   K1 = PZ * ZN3
   !
   IF ( PX == NPX - 1 ) THEN
      A(1) = MeshX( I1 + 1 - NPML )
      A(2) = MeshX( I1 + XN1 - NPML )
   ELSE
      A(1) = MeshX( I1 + 1 - NPML )
      A(2) = MeshX( I1 + XN1 - NPML + 1 )
   END IF
   !
   IF ( PY == NPY - 1 ) THEN
      B(1) = MeshY( J1 + 1 - NPML )
```

```
      B(2) = MeshY( J1 + YN2 - NPML )
   ELSE
      B(1) = MeshY( J1 + 1 - NPML )
      B(2) = MeshY( J1 + YN2 - NPML + 1  )
   END IF
   !
   IF ( ModelFlag == 0 ) THEN
      ! 2-D Model
      C(1) = MeshZ( ZN3 )
      C(2) = MeshZ( ZN3 )
   ELSE
      IF ( PZ == NPZ - 1 ) THEN
         C(1) = MeshZ( K1 + 1 - NPML )
         C(2) = MeshZ( K1 + ZN3 - NPML )
      ELSE
         C(1) = MeshZ( K1 + 1 - NPML )
         C(2) = MeshZ( K1 + ZN3 - NPML + 1 )
      END IF
   END IF
   !
   Flag0 = 2
   DO L = 1, SUM(NumProbe)
      IF ( ( ProbeCoordinate(L,1) >= A(1) ) ) THEN
      IF ( ( ProbeCoordinate(L,1) < A(2) ) .OR. ( ( ProbeCoordinate(L,1) == A(2) ) &
   &                    .AND. ( PX == (NPX-1) ) ) ) THEN
         IF ( ( ProbeCoordinate(L,2) >= B(1) ) ) THEN
         IF ( ( ProbeCoordinate(L,2) < B(2) ) .OR. ( ( ProbeCoordinate(L,2) == B(2) ) &
   &                    .AND. PY == ( NPY-1 ) ) ) THEN
            IF ( ( ProbeCoordinate(L,3) >= C(1) ) ) THEN
            IF ( ( ProbeCoordinate(L,3) < C(2) ) .OR. ( ( ProbeCoordinate(L,3) == C(2) ) &
   &                    .AND. PZ == ( NPZ-1 ) ) ) THEN
               ProbeFlag( Flag0 ) = L
               X2 = SideLength( L, 1 ) - RelativePosi( L, 1 )
               Y2 = SideLength( L, 2 ) - RelativePosi( L, 2 )
               Z2 = SideLength( L, 3 ) - RelativePosi( L, 3 )
               IF ( ModelFlag == 0 ) THEN
                  ! 2-D Model
                  I0 = Corner( L, 1 ) + NPML - I1
                  J0 = Corner( L, 2 ) + NPML - J1
                  k0 = Corner( L, 3 ) + ModelFlag * ( NPML - K1 )
                  ProbePressure(Flag0-1) = ( X2 * Y2 * P( I0, J0, 1 ) + &
   &                    RelativePosi( L, 1 ) * Y2 * P( I0+1, J0, 1 ) &
   &                    + RelativePosi( L, 1 ) * RelativePosi( L, 2 ) &
   &                    * P( I0+1, J0+1, 1 ) + X2 * RelativePosi( L, 2 ) &
   &                    * P( I0, J0+1, 1 ) ) / ( SideLength( L, 1 ) * &
   &                    SideLength( L, 2 ) )
               ELSE
                  ! 3-D Model

               END IF
               Flag0 = Flag0 + 1
            END IF
            END IF
```

```
          END IF
          END IF
        END IF
        END IF
      END DO
      ProbeFlag( 1 ) = Flag0 - 2
    END SUBROUTINE GetProbePressure
    ! ------------------------------------------------
    ! >>>>>>>>>>>>>>>>>>>>>>>>.
    SUBROUTINE TransformCoordinate( Jacobi, KexiX, EitaY, TaoZ, &
      &      MeshX, MeshY, MeshZ, N1, N2, N3, NPML, DeltaXYZ, ModelFlag )
      USE M_Spatial_dis
      IMPLICIT NONE
      INTEGER:: I, J, K
      INTEGER, INTENT(IN):: NPML
      INTEGER, INTENT(IN):: N1, N2, N3, ModelFlag
      REAL(KIND=8), INTENT(IN):: DeltaXYZ
      REAL(KIND=8), INTENT(IN):: MeshX( -(NPML-1) : ( N1 + NPML ) )
      REAL(KIND=8), INTENT(IN):: MeshY( -(NPML-1) : ( N2 + NPML ) )
      REAL(KIND=8), INTENT(IN):: MeshZ( -(NPML-1) : ( N3 + NPML ) )
      REAL(KIND=8), INTENT(OUT):: Jacobi( -(NPML-1) : ( N1 + NPML ), &
      &                            -(NPML-1) : ( N2 + NPML ), &
      &                            -(NPML-1) : ( N3 + NPML ) )
      REAL(KIND=8), INTENT(OUT):: KexiX( -(NPML-1) : ( N1 + NPML ) )
      REAL(KIND=8), INTENT(OUT):: EitaY( -(NPML-1) : ( N2 + NPML ) )
      REAL(KIND=8), INTENT(OUT):: TaoZ( -(NPML-1) : ( N3 + NPML ) )
      REAL(KIND=8):: XKexi( -(NPML-1) : ( N1 + NPML ) )
      REAL(KIND=8):: YEita( -(NPML-1) : ( N2 + NPML ) )
      REAL(KIND=8):: ZTao( -(NPML-1) : ( N3 + NPML ) )
      INTEGER:: SIZE0(2)
      !!
      !get x-direction derivative
      SIZE0(1) = - ( NPML - 1 )
      SIZE0(2) = N1 + NPML
      CALL DRP7( Xkexi, MeshX, DeltaXYZ, -(NPML-1), N1 + NPML, SIZE0 )
      !!
      !get y-direction derivatives
      SIZE0(1) = - ( NPML - 1 )
      SIZE0(2) = N2 + NPML
      CALL DRP7( YEita, MeshY, DeltaXYZ, -(NPML-1), N2 + NPML, SIZE0)
      !!
      !get z-direction derivatives
      IF ( ModelFlag == 0 ) THEN
        !2-D model
        Ztao = 1.0d0
      ELSE
        !3-D model
        SIZE0(1) = - ( NPML - 1 )
        SIZE0(2) = N3 + NPML
        CALL DRP7( ZTao, MeshZ, DeltaXYZ, -(NPML-1), N3 + NPML, SIZE0 )
      END IF
      !!
      !get Jacobi matrix and kexiX, EitaY, TaoZ
```

```fortran
    !IF ( ModelFlag == 0 ) THEN
      ! 2-D Model
    !   SIZE0 = N3
    !ELSE
      SIZE0(1) = - ( NPML - 1 )
      SIZE0(2) = N3 + NPML
    !END IF
    !
    DO K = SIZE0(1), SIZE0(2)
      DO J = - ( NPML - 1 ), N2 + NPML
        DO I = - ( NPML - 1 ), N1 + NPML
          Jacobi( I, J, K ) = XKexi(I) * YEita(J) * ZTao(K)
        END DO
      END DO
    END DO
    KexiX = 1.0d0 / XKexi
    EitaY = 1.0d0 / YEita
    TaoZ = 1.0d0 / ZTao
  END SUBROUTINE TransformCoordinate
  ! --------------------------------------
  ! >>>>>>>>>>>>>>>
  SUBROUTINE AbsorbingCoefficient( Beita, SigmaX1, SigmaX2, SigmaY1, SigmaY2, &
    &        SigmaZ1, SigmaZ2, AFA, SIGMA0, NPML, MeshX, MeshY, MeshZ, &
    &          N1, N2, N3, BeitaFlag, ModelFlag )
    !--------------------------introduction----------------
    ! This subroutine can get the absorbing coefficient for
    !   PML zone according to formula:
    !     SIGMAx = SIGMA0/DeltaX * ( (X-X0)/D )^afa
    !       here D: PML zone width
    !         SIGMA0: max absorbing coefficient
    !           X0: the interface baseline
    !          afa: absorbing coefficient(2&4)
    !   Beita:
    !       Beita = Um_ / ( 1 - Um_^2 )
    !       Um_ = int( u(y), a, b ) / ( b - a )
    !     usually for flow:
    !       inviscid: beita = 0.63
    !       Re = 5000: beita = 0.63
    !       Re = 500: beita = 0.63
    !       Re = 50: beita = 0.64
    !     No background flow: beita = 0.0d0
    !---------
    !     BeitaFlag = 0: no background flow
    !     BeitaFlag = 1: background flow
    !------------------------------------------------------
    !------------------------------------------------------
    IMPLICIT NONE
    INTEGER:: I, J, K
    INTEGER, INTENT(IN):: BeitaFlag
    INTEGER, INTENT(IN):: ModelFlag
    INTEGER, INTENT(IN):: NPML, N1, N2, N3
    REAL(KIND=8), INTENT(IN):: AFA, Sigma0
```

```
REAL(KIND=8), INTENT(IN):: MeshX( -(NPML-1) : (N1+NPML) )
REAL(KIND=8), INTENT(IN):: MeshY( -(NPML-1) : (N2+NPML) )
REAL(KIND=8), INTENT(IN):: MeshZ( -(NPML-1) : (N3+NPML) )
REAL(KIND=8), INTENT(OUT):: SigmaX1( -(NPML-1) : 0 )
REAL(KIND=8), INTENT(OUT):: SigmaX2( 1 : NPML )
REAL(KIND=8), INTENT(OUT):: SigmaY1( -(NPML-1) : 0 )
REAL(KIND=8), INTENT(OUT):: SigmaY2( 1 : NPML )
REAL(KIND=8), INTENT(OUT):: SigmaZ1( -(NPML-1) : 0 )
REAL(KIND=8), INTENT(OUT):: SigmaZ2( 1 : NPML )
REAL(KIND=8), INTENT(OUT):: Beita
REAL(KIND=8):: DeltaX0(2), DeltaY0(2), DeltaZ0(2)
REAL(KIND=8):: D(3,2)
!!
D(1,1) = ( MeshX( 1 ) - MeshX( - ( NPML - 1 ) ) )
D(1,2) = ( MeshX( N1 + NPML ) - MeshX( N1 ) )
D(2,1) = ( MeshY( 1 ) - MeshY( - ( NPML - 1 ) ) )
D(2,2) = ( MeshY( N2 + NPML ) - MeshY( N2 ) )
D(3,1) = ( MeshZ( 1 ) - MeshZ( - ( NPML - 1 ) ) )
D(3,2) = ( MeshZ( N3 + NPML ) - MeshZ( N3 ) )
DO I = 1, 2
   DeltaX0(I) = D(1, I) / NPML
   DeltaY0(I) = D(2, I) / NPML
   DeltaZ0(I) = D(3, I) / NPML
END DO
!
DO I = - ( NPML - 1 ) , 0
   !absorbing coefficient for x direction
   SigmaX1( I ) = ( Sigma0 / DeltaX0(1) ) * ( DABS( &
&      MeshX( I ) - MeshX( 1 ) ) / D(1, 1) )**afa
   SigmaX2( I + NPML ) = ( Sigma0 / DeltaX0(2) ) * &
&     ( ( MeshX( I + N1+ NPML ) - &
&         MeshX( N1 ) ) / D(1, 2) )**afa
   !
   !absorbing coefficient for y direction
   SigmaY1( I ) = ( Sigma0 / DeltaY0(1) ) * ( DABS( &
&      MeshY( I ) - MeshY( 1 ) ) / D(2, 1) )**afa
   SigmaY2( I + NPML ) = ( Sigma0 / DeltaY0(2) ) * &
&      ( ( MeshY( I + N2+ NPML ) - &
&         MeshY( N2 ) ) / D(2, 2) )**afa
END DO
!
IF ( ModelFlag == 0 ) THEN
   ! 2-D model
   SigmaZ1 = 0.0d0
   SigmaZ2 = 0.0d0
ELSE
   ! 3-D model
   DO I = - ( NPML - 1 ), 0
      !absorbing coefficient for z direction
      SigmaZ1( I ) = ( Sigma0 / DeltaZ0(1) ) * ( DABS( &
   &      MeshZ( I ) - MeshZ( 1 ) ) / D(3, 1) )**afa
      SigmaZ2( I + NPML ) = ( Sigma0 / DeltaZ0(2) ) * &
   &      ( ( MeshZ( I + N3+ NPML ) - &
```

```
        &         MeshZ( N3 ) ) / D(3, 2) )**afa
        END DO
      END IF
      !
      IF ( BeitaFlag == 0 ) THEN
        ! No background flow
        Beita = 0.0d0
      ELSE
        ! background flow: recommended by Hu & Li(2008)
        Beita = 0.63d0
      END IF
    END SUBROUTINE AbsorbingCoefficient
    ! --------------------------------------------------
    ! >>>>>>>>>>>>>>>>>>>>.
    SUBROUTINE GetMeanFlow( U0, V0, W0, P0, ROU0, C0, NA, XN1, YN2, ZN3, &
      &  MeshX, MeshY, MeshZ, Centre, NPML, N1, N2, N3, PX, PY, PZ, &
      &      NPX, NPY, NPZ, ModelFLag )
      ! --------------------------introduction------------------
      ! Here we consider the mean flow around the immersed body(
      !   here is a cylinder). The mean flow model can be got from
      !       potential flow solution and may be 2-D or 3-D.
      !   Flag = 0: the basic flow- 2-D & 3-D
      !   Flag = 1: incompressible potential flow- 2-D
      !   Flag = 2: incompressible potential flow- 3-D
      !   Flag = 3: compressible potential flow- 2-D
      !   Flag = 4: compressible potential flow- 3-D
      ! -------------------------------------------------------
      USE BackgroundFlowField
      IMPLICIT NONE
      INTEGER:: I, J, K
      INTEGER, INTENT(IN):: NA, XN1, YN2, ZN3
      INTEGER, INTENT(IN):: NPML, N1, N2, N3
      INTEGER, INTENT(IN):: PX, PY, PZ, NPX, NPY, NPZ
      INTEGER, INTENT(IN):: ModelFlag
      REAL(KIND=8), INTENT(IN):: Centre(3)
      REAL(KIND=8), INTENT(IN):: MeshX( -(NPML-1):(N1+NPML) )
      REAL(KIND=8), INTENT(IN):: MeshY( -(NPML-1):(N2+NPML) )
      REAL(KIND=8), INTENT(IN):: MeshZ( -(NPML-1):(N3+NPML) )
      REAL(KIND=8), INTENT(OUT):: U0( -(NA-1):(XN1+NA),-(NA-1):(YN2+NA),-(NA-1):(ZN3+NA) )
      REAL(KIND=8), INTENT(OUT):: V0( -(NA-1):(XN1+NA),-(NA-1):(YN2+NA),-(NA-1):(ZN3+NA) )
      REAL(KIND=8), INTENT(OUT):: W0( -(NA-1):(XN1+NA),-(NA-1):(YN2+NA),-(NA-1):(ZN3+NA) )
      REAL(KIND=8), INTENT(OUT):: P0( -(NA-1):(XN1+NA),-(NA-1):(YN2+NA),-(NA-1):(ZN3+NA) )
      REAL(KIND=8), INTENT(OUT):: ROU0( -(NA-1):(XN1+NA),-(NA-1):(YN2+NA),-(NA-1):(ZN3+NA) )
      REAL(KIND=8), INTENT(OUT):: C0( -(NA-1):(XN1+NA),-(NA-1):(YN2+NA),-(NA-1):(ZN3+NA) )
      INTEGER:: Flag, I0, J0, K0, IFlag, JFlag, KFlag, XFlag(2), YFlag(2), ZFlag(2)
      REAL(KIND=8):: R1, R0, Kexi
      !
      Flag = 1
      R0 = 0.5d0
      Kexi = 1.0d-6
      !
      I0 = PX * XN1
```

```
J0 = PY * YN2
K0 = PZ * ZN3
!
U0 = Ma
V0 = 0.0d0
W0 = 0.0d0
P0 = 1.0d0
ROU0 = 1.0d0
C0 = 1.0d0
!
IF ( ModelFlag == 0 ) THEN
   ! 2-D Model
   XFlag(1) = - ( NA - 1 )
   XFlag(2) = ( XN1 + NA )
   YFlag(1) = - ( NA - 1 )
   YFlag(2) = ( YN2 + NA )
   ZFlag = 1
ELSE
   XFlag(1) = - ( NA - 1 )
   XFlag(2) = ( XN1 + NA )
   YFlag(1) = - ( NA - 1 )
   YFlag(2) = ( YN2 + NA )
   ZFlag(1) = - ( NA - 1 )
   ZFlag(2) = ZN3 + NA
END IF
!
! MPI boundary judgement
! forward and rearward side
IF ( PX == 0 ) THEN
   XFlag(1) = 1
ELSEIF( PX == NPX - 1 ) THEN
   XFlag(2) = XN1
END IF
!
! left and right side
IF ( PY == 0 ) THEN
   YFlag(1) = 1
ELSEIF( PY == NPY - 1 ) THEN
   YFlag(2) = YN2
END IF
!
! upper and lower side
IF ( PZ == 0 ) THEN
   ZFlag(1) = 1
ELSEIF( PZ == NPZ - 1 ) THEN
   ZFlag(2) = ZN3
END IF
!
IF ( Flag == 0 ) THEN
   ! static ambiance
   U0 = 0.0d0
   V0 = 0.0d0
ELSEIF ( Flag == 1 ) THEN
```

```
        ! compressible background flow
        ! U = U0 * ( 1 - R^2 * cos( 2 * Thita ) / r^2 )
        ! V = - V0 * R^2 * sin( 2 * Thita ) / r^2
        DO K = ZFlag(1), ZFlag(2)
          DO J = YFlag(1), YFlag(2)
            DO I = XFlag(1), XFlag(2)
              IFlag = I + I0 - NPML
              JFlag = J + J0 - NPML
              KFlag = K + ModelFlag * ( K0 - NPML )
              IF ( ( IFlag .GE. -(NPML-1) ) .AND. ( IFlag .LE. (N1+NPML) ) &
     &           .AND. ( JFlag .GE. -(NPML-1) ) .AND. ( JFlag .LE. (N2+NPML) ) &
     &           .AND. ( KFlag .GE. -(NPML-1) ) .AND. ( KFlag .LE. (N3+NPML) ) ) THEN
                R1 = DSQRT( ( MeshX(IFlag)-Centre(1) )**2 + &
     &                ( MeshY(JFlag)-Centre(2) )**2 ) + kexi
                IF ( R1 >= R0 ) THEN
                  U0( I, J, K ) = Ma * ( 1.0d0 - R0**2 * ( 2.0d0 * ( (MeshX(IFlag) - &
     &                  Centre(1) ) / R1 )**2 - 1.0d0 ) / R1**2 )
                  V0( I, J, K ) = - Ma * R0**2 * 2.0d0 * ( MeshX(IFlag)-Centre(1) ) * &
     &                  ( MeshY(JFlag)-Centre(2) ) / R1**4
                  ROU0( I, J, K ) = ( 1.0d0 - ( 1.4d0 - 1.0d0 ) * ( U0( I, J, K )**2 + &
     &                  V0( I, J, K )**2 - Ma**2 ) / 2.0d0 )**( 1.0d0 / ( 1.4d0 - 1 ) )
                  C0( I, J, K ) = ( 1.0d0 - ( 1.4d0 - 1 ) * ( U0( I, J, K )**2 + &
     &                  V0( I, J, K )**2 - Ma**2 ) / 2.0d0 )**0.5d0
                ELSE
                  U0( I, J, K ) = Ma * ( 1.0d0 - R1**2 * ( 2.0d0 * ( (MeshX(IFlag) - &
     &                  Centre(1) ) / R1 )**2 - 1.0d0 ) / R0**2 )
                  V0( I, J, K ) = - Ma * R1**2 * 2.0d0 * ( MeshX(IFlag)-Centre(1) ) * &
     &                  ( MeshY(JFlag)-Centre(2) ) / R0**4
                  ROU0( I, J, K ) = ( 1.0d0 - ( 1.4d0 - 1.0d0 ) * ( U0( I, J, K )**2 + &
     &                  V0( I, J, K )**2 - Ma**2 ) / 2.0d0 )**( 1.0d0 / ( 1.4d0 - 1 ) )
                  C0( I, J, K ) = ( 1.0d0 - ( 1.4d0 - 1 ) * ( U0( I, J, K )**2 + &
     &                  V0( I, J, K )**2 - Ma**2 ) / 2.0d0 )**0.5d0
                END IF
              END IF
            END DO
          END DO
        END DO
      ELSEIF ( Flag == 2 ) THEN
        !incompressible background flow: rou0 = c0 = 1
        DO K = ZFlag(1), ZFlag(2)
          DO J = YFlag(1), YFlag(2)
            DO I = XFlag(1), XFlag(2)
              IFlag = I + I0 - NPML
              JFlag = J + J0 - NPML
              KFlag = K + ModelFlag * ( K0 - NPML )
              IF ( ( IFlag .GE. -(NPML-1) ) .AND. ( IFlag .LE. (N1+NPML) ) &
     &           .AND. ( JFlag .GE. -(NPML-1) ) .AND. ( JFlag .LE. (N2+NPML) ) &
     &           .AND. ( KFlag .GE. -(NPML-1) ) .AND. ( KFlag .LE. (N3+NPML) ) ) THEN
                R1 = DSQRT( ( MeshX(IFlag)-Centre(1) )**2 + &
     &                ( MeshY(JFlag)-Centre(2) )**2 ) + kexi
                IF ( R1 >= R0 ) THEN
                  U0( I, J, K ) = Ma * ( 1.0d0 - R0**2 * ( 2.0d0 * ( (MeshX(IFlag) - &
     &                  Centre(1) ) / R1 )**2 - 1.0d0 ) / R1**2 )
```

```
                    V0( I, J, K ) = - Ma * R0**2 * 2.0d0 * ( MeshX(IFlag)-Centre(1) ) * &
    &                        ( MeshY(JFlag)-Centre(2) ) / R1**4
                  ELSE
                    U0( I, J, K ) = Ma * ( 1.0d0 - R1**2 * ( 2.0d0 * ( (MeshX(IFlag) - &
    &                         Centre(1) ) / R1 )**2 - 1.0d0 ) / R0**2 )
                    V0( I, J, K ) = - Ma * R1**2 * 2.0d0 * ( MeshX(IFlag)-Centre(1) ) * &
    &                        ( MeshY(JFlag)-Centre(2) ) / R0**4
                  END IF
                END IF
              END DO
            END DO
          END DO
    ELSEIF ( Flag == 3 ) THEN

    ELSEIF ( Flag == 4 ) THEN

    END IF
  END SUBROUTINE GetMeanFlow
  ! ----------------------------------
  ! >>>>>>>>>>>>>>>
  SUBROUTINE InitializeAcousticField( U, V, W, P, ROU, Au11, Au12, Au13, Au14, &
    &     Au21, Au22, Au23, Au24, Au31, Au32, Au33, Au34, NA, XN1, YN2, &
    &         ZN3, MeshX, MeshY, MeshZ, C0, NPML, N1, N2, N3, PX, PY, PZ, &
    &             NPX, NPY, NPZ, ModelFlag )
    USE AcousticInitialCondition
    IMPLICIT NONE
    INTEGER:: I, J, K
    INTEGER, INTENT(IN):: NA, XN1, YN2, ZN3
    INTEGER, INTENT(IN):: NPML, N1, N2, N3
    INTEGER, INTENT(IN):: PX, PY, PZ, ModelFlag
    INTEGER, INTENT(IN):: NPX, NPY, NPZ
    REAL(KIND=8), INTENT(OUT):: U( -(NA-1):(XN1+NA),-(NA-1):(YN2+NA),-(NA-1):(ZN3+NA) )
    REAL(KIND=8), INTENT(OUT):: V( -(NA-1):(XN1+NA),-(NA-1):(YN2+NA),-(NA-1):(ZN3+NA) )
    REAL(KIND=8), INTENT(OUT):: W( -(NA-1):(XN1+NA),-(NA-1):(YN2+NA),-(NA-1):(ZN3+NA) )
    REAL(KIND=8), INTENT(OUT):: P( -(NA-1):(XN1+NA),-(NA-1):(YN2+NA),-(NA-1):(ZN3+NA) )
    REAL(KIND=8), INTENT(OUT):: ROU( -(NA-1):(XN1+NA),-(NA-1):(YN2+NA),-(NA-1):(ZN3+NA) )
    REAL(KIND=8), INTENT(OUT):: Au11( 1 : XN1, 1 : YN2, 1 : ZN3 )
    REAL(KIND=8), INTENT(OUT):: Au12( 1 : XN1, 1 : YN2, 1 : ZN3 )
    REAL(KIND=8), INTENT(OUT):: Au13( 1 : XN1, 1 : YN2, 1 : ZN3 )
    REAL(KIND=8), INTENT(OUT):: Au14( 1 : XN1, 1 : YN2, 1 : ZN3 )
    REAL(KIND=8), INTENT(OUT):: Au21( 1 : XN1, 1 : YN2, 1 : ZN3 )
    REAL(KIND=8), INTENT(OUT):: Au22( 1 : XN1, 1 : YN2, 1 : ZN3 )
    REAL(KIND=8), INTENT(OUT):: Au23( 1 : XN1, 1 : YN2, 1 : ZN3 )
    REAL(KIND=8), INTENT(OUT):: Au24( 1 : XN1, 1 : YN2, 1 : ZN3 )
    REAL(KIND=8), INTENT(OUT):: Au31( 1 : XN1, 1 : YN2, 1 : ZN3 )
    REAL(KIND=8), INTENT(OUT):: Au32( 1 : XN1, 1 : YN2, 1 : ZN3 )
    REAL(KIND=8), INTENT(OUT):: Au33( 1 : XN1, 1 : YN2, 1 : ZN3 )
    REAL(KIND=8), INTENT(OUT):: Au34( 1 : XN1, 1 : YN2, 1 : ZN3 )
    REAL(KIND=8), INTENT(IN):: C0( -(NA-1):(XN1+NA),-(NA-1):(YN2+NA),-(NA-1):(ZN3+NA) )
    REAL(KIND=8), INTENT(IN):: MeshX( -(NPML-1):(N1+NPML) )
    REAL(KIND=8), INTENT(IN):: MeshY( -(NPML-1):(N2+NPML) )
    REAL(KIND=8), INTENT(IN):: MeshZ( -(NPML-1):(N3+NPML) )
    REAL(KIND=8):: MeshX1( -(NA+NPML-1):(N1+NPML+NA) )
```

```
REAL(KIND=8):: MeshY1( -(NA+NPML-1):(N2+NPML+NA) )
REAL(KIND=8):: MeshZ1( -(NA+NPML-1):(N3+NPML+NA) )
INTEGER:: I0, J0, K0, SIZE0(2), SIZE1(2), SIZE2(2)
!
I0 = PX * XN1
J0 = PY * YN2
K0 = PZ * ZN3
!
U = 0.0d0
V = 0.0d0
W = 0.0d0
P = 0.0d0
ROU = 0.0d0
Au11 = 0.0d0
Au12 = 0.0d0
Au13 = 0.0d0
Au14 = 0.0d0
Au21 = 0.0d0
Au22 = 0.0d0
Au23 = 0.0d0
Au24 = 0.0d0
Au31 = 0.0d0
Au32 = 0.0d0
Au33 = 0.0d0
Au34 = 0.0d0
!
! model dimension judgement: 2-D or 3-D?
IF ( ModelFlag == 0 ) THEN
   SIZE0(1) = - ( NA - 1 )
   SIZE0(2) = ( XN1 + NA )
   SIZE1(1) = - ( NA - 1 )
   SIZE1(2) = ( YN2 + NA )
   SIZE2(1) = 1
   SIZE2(2) = 1
ELSE
   SIZE0(1) = - ( NA - 1 )
   SIZE0(2) = ( XN1 + NA )
   SIZE1(1) = - ( NA - 1 )
   SIZE1(2) = ( YN2 + NA )
   SIZE2(1) = - ( NA - 1 )
   SIZE2(2) = ( ZN3 + NA )
END IF
!
! MPI boundary judgement
! forward and rearward side
IF ( PX == 0 ) THEN
   SIZE0(1) = 1
ELSEIF( PX == NPX - 1 ) THEN
   SIZE0(2) = XN1
END IF
!
! left and right side
IF ( PY == 0 ) THEN
```

```
        SIZE1(1) = 1
     ELSEIF( PY == NPY - 1 ) THEN
        SIZE1(2) = YN2
     END IF
     !
     ! upper and lower side
     IF ( PZ == 0 ) THEN
        SIZE2(1) = 1
     ELSEIF( PZ == NPZ - 1 ) THEN
        SIZE2(2) = ZN3
     END IF
     DO K = SIZE2(1) , SIZE2(2)
        DO J = SIZE1(1), SIZE1(2)
           DO I = SIZE0(1), SIZE0(2)
              P( I, J, K ) = A * DEXP( B * ( ( MeshX( I+I0-NPML ) - X0 )**2 + &
     &              ( MeshY( J+J0-NPML )-Y0 )**2 + &
     &                 ( MeshZ( K+ModelFlag*(K0-NPML) )-Z0 )**2 ) ) &
     &                    * DSIN( Omega * 0.0d0 )
              ROU( I, J, K ) = P( I, J, K ) / C0( I, J, K )**2
           END DO
        END DO
     END DO
  END SUBROUTINE InitializeAcousticField
  ! -----------------------------------
  ! >>>>>>>>>>>>>>>>.
  SUBROUTINE GetConservativeVariables( Q1, Q2, Q3, Q4, F1, F2, F3, F4, G1, G2, G3, G4, &
     &            H1, H2, H3, H4, Au11, Au12, Au13, Au14, Au21, Au22, Au23, Au24, &
     &               Au31, Au32, Au33, Au34, S1, S2, S3, S4, SAu11, SAu12, SAu13, &
     &               SAu14, SAu21, SAu22, SAu23, SAu24, SAu31, SAu32, SAu33, SAu34, &
     &               U0, V0, W0, P0, ROU0, C0, U, V, W, P, ROU, NA, XN1, YN2, ZN3, Jacobi, &
     &               KexiX, EitaY, TaoZ, MeshX, MeshY, MeshZ, Beita, SigmaX1, SigmaX2, &
     &               SigmaY1, SigmaY2, SigmaZ1, SigmaZ2, NPML, N1, N2, N3, PX, PY, PZ, &
     &               NPX, NPY, NPZ, Loops, DeltaT, ModelFlag )
     IMPLICIT NONE
     INTEGER:: I, J, K
     INTEGER, INTENT(IN):: ModelFlag
     INTEGER, INTENT(IN):: NA, XN1, YN2, ZN3
     INTEGER, INTENT(IN):: N1, N2, N3
     INTEGER, INTENT(IN):: PX, PY, PZ, NPX, NPY, NPZ
     INTEGER, INTENT(IN):: NPML
     INTEGER, INTENT(IN):: Loops
     REAL(KIND=8), INTENT(IN):: DeltaT
     REAL(KIND=8), INTENT(IN):: Beita
     REAL(KIND=8), INTENT(IN):: U0( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
     REAL(KIND=8), INTENT(IN):: V0( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
     REAL(KIND=8), INTENT(IN):: W0( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
     REAL(KIND=8), INTENT(IN):: P0( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
     REAL(KIND=8), INTENT(IN):: ROU0( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
     REAL(KIND=8), INTENT(IN):: C0( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
     REAL(KIND=8), INTENT(IN):: U( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
     REAL(KIND=8), INTENT(IN):: V( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
     REAL(KIND=8), INTENT(IN):: W( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
     REAL(KIND=8), INTENT(IN):: P( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
```

```
REAL(KIND=8), INTENT(IN):: ROU( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(IN):: SigmaX1( -(NPML-1) : 0 )
REAL(KIND=8), INTENT(IN):: SigmaX2( 1 : NPML )
REAL(KIND=8), INTENT(IN):: SigmaY1( -(NPML-1) : 0 )
REAL(KIND=8), INTENT(IN):: SigmaY2( 1 : NPML )
REAL(KIND=8), INTENT(IN):: SigmaZ1( -(NPML-1) : 0 )
REAL(KIND=8), INTENT(IN):: SigmaZ2( 1 : NPML )
REAL(KIND=8), INTENT(IN):: Jacobi( -(NPML-1):(N1+NPML),-(NPML-1):(N2+NPML),-(NPML-1):(N…
REAL(KIND=8), INTENT(IN):: KexiX( -(NPML-1) : (N1 + NPML) )
REAL(KIND=8), INTENT(IN):: EitaY( -(NPML-1) : (N2 + NPML) )
REAL(KIND=8), INTENT(IN):: TaoZ( -(NPML-1) : (N3 + NPML) )
REAL(KIND=8), INTENT(IN):: MeshX( -(NPML-1) : ( N1 + NPML ) )
REAL(KIND=8), INTENT(IN):: MeshY( -(NPML-1) : ( N2 + NPML ) )
REAL(KIND=8), INTENT(IN):: MeshZ( -(NPML-1) : ( N3 + NPML ) )
REAL(KIND=8), INTENT(OUT):: Q1( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(OUT):: Q2( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(OUT):: Q3( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(OUT):: Q4( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(OUT):: F1( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(OUT):: F2( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(OUT):: F3( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(OUT):: F4( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(OUT):: G1( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(OUT):: G2( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(OUT):: G3( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(OUT):: G4( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(OUT):: H1( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(OUT):: H2( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(OUT):: H3( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(OUT):: H4( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(INOUT):: Au11( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: Au12( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: Au13( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: Au14( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: Au21( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: Au22( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: Au23( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: Au24( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: Au31( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: Au32( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: Au33( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: Au34( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: S1( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: S2( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: S3( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: S4( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu11( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu12( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu13( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu14( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu21( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu22( 1 : XN1, 1 : YN2, 1 : ZN3 )
```

```
REAL(KIND=8), INTENT(OUT):: SAu23( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu24( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu31( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu32( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu33( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu34( 1 : XN1, 1 : YN2, 1 : ZN3 )
INTEGER:: I0, J0, K0
REAL(KIND=8):: Jacobi1( -(NPML-1+NA) : (N1+NPML+NA), -(NPML-1+NA) : (N2+NPML+NA), &
&                                        -(NPML-1+NA) : (N3+NPML+NA) )
REAL(KIND=8):: KexiX1( - ( NPML - 1 + NA ) : ( N1 + NPML + NA ) )
REAL(KIND=8):: EitaY1( - ( NPML - 1 + NA ) : ( N2 + NPML + NA ) )
REAL(KIND=8):: TaoZ1( - ( NPML - 1 + NA ) : ( N3 + NPML + NA ) )
INTEGER:: ZFlag, Z0(2)
REAL(KIND=8):: T
!!
T = Loops * DeltaT
Jacobi1 = 1.0d0
KexiX1 = 1.0d0
EitaY1 = 1.0d0
TaoZ1 = 1.0d0
Jacobi1( -(NPML-1):(N1+NPML),-(NPML-1):(N2+NPML),-(NPML-1):(N3+NPML) ) = Jacobi( &
&   -(NPML-1):(N1+NPML),-(NPML-1):(N2+NPML),-(NPML-1):(N3+NPML) )
KexiX1( -(NPML-1) : (N1 + NPML) ) = KexiX( -(NPML-1) : (N1 + NPML) )
EitaY1( -(NPML-1) : (N2 + NPML) ) = EitaY( -(NPML-1) : (N2 + NPML) )
TaoZ1( -(NPML-1) : (N3 + NPML) ) = TaoZ( -(NPML-1) : (N3 + NPML) )
!
!get the start point coordinates for MPI slice
I0 = XN1 * PX
J0 = YN2 * PY
K0 = ZN3 * PZ
ZFlag = ModelFlag * ( K0 - NPML )
IF ( ModelFlag == 0 ) THEN
   ! 2-D Model
   Z0(1) = ZN3
   Z0(2) = ZN3
ELSE
   ! 3-D Model
   Z0(1) = - ( NA - 1 )
   Z0(2) = ZN3 + NA
END IF
!
!!
DO K = Z0(1), Z0(2)
   DO J = - ( NA - 1 ), ( YN2 + NA )
      DO I = - ( NA - 1 ), ( XN1 + NA )
         !conservative variables: Q1, Q2, Q3, Q4
         Q1( I, J, K ) = Jacobi1( I0+I-NPML, J0+J-NPML, ZFlag + K ) * ROU( I, J, K )
         Q2( I, J, K ) = Jacobi1( I0+I-NPML, J0+J-NPML, ZFlag + K ) * ( ROU0( I, J, K ) * &
&            U( I, J, K ) + U0( I, J, K ) * ROU( I, J, K ) )
         Q3( I, J, K ) = Jacobi1( I0+I-NPML, J0+J-NPML, ZFlag + K ) * ( ROU0( I, J, K ) * &
&            V( I, J, K ) + V0( I, J, K ) * ROU( I, J, K ) )
         Q4( I, J, K ) = Jacobi1( I0+I-NPML, J0+J-NPML, ZFlag + K ) * ( ROU0( I, J, K ) * &
&            W( I, J, K ) + W0( I, J, K ) * ROU( I, J, K ) )
```

```fortran
            !
            !flux: F1, F2, F3, F4, G1, G2, G3, G4, H1, H2, H3, H4
            F1( I, J, K ) = Jacobi1( I0+I-NPML, J0+J-NPML, ZFlag + K ) * KexiX1( I0+I-NPML ) &
      &        * ( ROU0( I, J, K ) * U( I, J, K ) + U0( I, J, K ) * ROU( I, J, K ) )
            F2( I, J, K ) = Jacobi1( I0+I-NPML, J0+J-NPML, ZFlag + K ) * KexiX1( I0+I-NPML ) &
      &        * ( 2.0d0 * ROU0( I, J, K ) * U0( I, J, K ) * U( I, J, K ) &
      &          + U0( I, J, K )**2 * ROU( I, J, K ) + P( I, J, K ) )
            F3( I, J, K ) = Jacobi1( I0+I-NPML, J0+J-NPML, ZFlag + K ) * KexiX1( I0+I-NPML ) &
      &        * ( ROU0( I, J, K ) * V0( I, J, K ) * U( I, J, K ) + ROU0( I, J, K ) * &
      &        U0( I, J, K ) * V( I, J, K ) + U0( I, J, K ) * V0( I, J, K ) * ROU( I, J, K ) )
            F4( I, J, K ) = Jacobi1( I0+I-NPML, J0+J-NPML, ZFlag + K ) * KexiX1( I0+I-NPML ) &
      &        * ( ROU0( I, J, K ) * W0( I, J, K ) * U( I, J, K ) + ROU0( I, J, K ) * &
      &        U0( I, J, K ) * W( I, J, K ) + W0( I, J, K ) * U0( I, J, K ) * ROU( I, J, K ) )
!           F1( I, J, K ) = ( ROU0( I, J, K ) * U( I, J, K ) + U0( I, J, K ) * ROU( I, J, K ) )
!           F2( I, J, K ) = ( 2.0d0 * ROU0( I, J, K ) * U0( I, J, K ) * U( I, J, K ) + &
!     &        U0( I, J, K )**2 * ROU( I, J, K ) + P( I, J, K ) )
!           F3( I, J, K ) = ( ROU0( I, J, K ) * V0( I, J, K ) * U( I, J, K ) + &
!     &        ROU0( I, J, K ) * U0( I, J, K ) * V( I, J, K ) + U0( I, J, K ) * &
!     &          V0( I, J, K ) * ROU( I, J, K ) )
!           F4( I, J, K ) = ( ROU0( I, J, K ) * W0( I, J, K ) * U( I, J, K ) + &
!     &        ROU0( I, J, K ) * U0( I, J, K ) * W( I, J, K ) + W0( I, J, K ) * &
!     &          U0( I, J, K ) * ROU( I, J, K ) )
            !
            !Y- direction flux: G
            G1( I, J, K ) = Jacobi1( I0+I-NPML, J0+J-NPML, ZFlag+K ) * EitaY1( J0+J-NPML ) * &
      &        ( ROU0( I, J, K ) * V( I, J, K ) + V0( I, J, K ) * ROU( I, J, K ) )
            G2( I, J, K ) = Jacobi1( I0+I-NPML, J0+J-NPML, ZFlag+K ) * EitaY1( J0+J-NPML ) * &
      &        ( ROU0( I, J, K ) * U0( I, J, K ) * V( I, J, K ) + ROU0( I, J, K ) * &
      &        V0( I, J, K ) * U( I, J, K ) + U0( I, J, K ) * V0( I, J, K ) * ROU( I, J, K ) )
            G3( I, J, K ) = Jacobi1( I0+I-NPML, J0+J-NPML, ZFlag+K ) * EitaY1( J0+J-NPML ) * &
      &        ( 2.0d0 * ROU0( I, J, K ) * V0( I, J, K ) * V( I, J, K ) + V0( I, J, K )**2 &
      &          * ROU( I, J, K ) + P( I, J, K ) )
            G4( I, J, K ) = Jacobi1( I0+I-NPML, J0+J-NPML, ZFlag+K ) * EitaY1( J0+J-NPML ) * &
      &        ( ROU0( I, J, K ) * W0( I, J, K ) * V( I, J, K ) + ROU0( I, J, K ) * &
      &        V0( I, J, K ) * W( I, J, K ) + W0( I, J, K ) * V0( I, J, K ) * ROU( I, J, K ) )
            !
            ! Z- direction flux: H
            H1( I, J, K ) = Jacobi1( I0+I-NPML, J0+J-NPML, ZFlag+K ) * TaoZ1( ZFlag+K ) * &
      &        ( ROU0( I, J, K ) * W( I, J, K ) + W0( I, J, K ) * ROU( I, J, K ) )
            H2( I, J, K ) = Jacobi1( I0+I-NPML, J0+J-NPML, ZFlag+K ) * TaoZ1( ZFlag+K ) * &
      &        ( ROU0( I, J, K ) * U0( I, J, K ) * W( I, J, K ) + ROU0( I, J, K ) * &
      &        W0( I, J, K ) * U( I, J, K ) + U0( I, J, K ) * W0( I, J, K ) * ROU( I, J, K ) )
            H3( I, J, K ) = Jacobi1( I0+I-NPML, J0+J-NPML, ZFlag+K ) * TaoZ1( ZFlag+K ) * &
      &        ( ROU0( I, J, K ) * V0( I, J, K ) * W( I, J, K ) + ROU0( I, J, K ) * &
      &        W0( I, J, K ) * V( I, J, K ) + V0( I, J, K ) * W0( I, J, K ) * ROU( I, J, K ) )
            H4( I, J, K ) = Jacobi1( I0+I-NPML, J0+J-NPML, ZFlag+K ) * TaoZ1( ZFlag+K ) * &
      &        ( 2.0d0 * ROU0( I, J, K ) * W0( I, J, K ) * W( I, J, K ) + W0( I, J, K )**2 &
      &          * ROU( I, J, K ) + P( I, J, K ) )
          !
          END DO
        END DO
      END DO
      IF ( Loops == 1 ) THEN
```

```
          DO K = Z0(1), Z0(2)
            DO J = 1, YN2
              DO I = 1, XN1
                !Auxiliary variables: x- direction
                Au11( I, J, K ) = Jacobi1( I0+I-NPML, J0+J-NPML, ZFlag+K ) * Au11( I, J, K )
                Au12( I, J, K ) = Jacobi1( I0+I-NPML, J0+J-NPML, ZFlag+K ) * Au12( I, J, K )
                Au13( I, J, K ) = Jacobi1( I0+I-NPML, J0+J-NPML, ZFlag+K ) * Au13( I, J, K )
                Au14( I, J, K ) = Jacobi1( I0+I-NPML, J0+J-NPML, ZFlag+K ) * Au14( I, J, K )
                !
                ! y- direction
                Au21( I, J, K ) = Jacobi1( I0+I-NPML, J0+J-NPML, ZFlag+K ) * Au21( I, J, K )
                Au22( I, J, K ) = Jacobi1( I0+I-NPML, J0+J-NPML, ZFlag+K ) * Au22( I, J, K )
                Au23( I, J, K ) = Jacobi1( I0+I-NPML, J0+J-NPML, ZFlag+K ) * Au23( I, J, K )
                Au24( I, J, K ) = Jacobi1( I0+I-NPML, J0+J-NPML, ZFlag+K ) * Au24( I, J, K )
                !~
                ! z- direction
                Au31( I, J, K ) = Jacobi1( I0+I-NPML, J0+J-NPML, ZFlag+K ) * Au31( I, J, K )
                Au32( I, J, K ) = Jacobi1( I0+I-NPML, J0+J-NPML, ZFlag+K ) * Au32( I, J, K )
                Au33( I, J, K ) = Jacobi1( I0+I-NPML, J0+J-NPML, ZFlag+K ) * Au33( I, J, K )
                Au34( I, J, K ) = Jacobi1( I0+I-NPML, J0+J-NPML, ZFlag+K ) * Au34( I, J, K )
              END DO
            END DO
          END DO
        END IF
        !
        ! far-field PML zone
        CALL PMLSource( S1, S2, S3, S4, SAu11, SAu12, SAu13, SAu14, SAu21, SAu22, SAu23, SAu24, &
        &      SAu31, SAu32, SAu33, SAu34, F1, F2, F3, F4, Au11, Au12, Au13, Au14, Au21, &
        &        Au22, Au23, Au24, Au31, Au32, Au33, Au34, NA, XN1, YN2, ZN3, Jacobi, &
        &        KexiX, EitaY, TaoZ, MeshX, MeshY, MeshZ, Beita, SigmaX1, SigmaX2, &
        &        SigmaY1, SigmaY2, SigmaZ1, SigmaZ2, N1, N2, N3, PX, PY, PZ, NPX, NPY, &
        &        NPZ, NPML, T, C0, ModelFlag )
        !
      END SUBROUTINE GetConservativeVariables
      !!
      SUBROUTINE PMLSource( S1, S2, S3, S4, SAu11, SAu12, SAu13, SAu14, SAu21, SAu22, SAu23, SAu24, &
        &      SAu31, SAu32, SAu33, SAu34, F1, F2, F3, F4, Au11, Au12, Au13, Au14, Au21, &
        &        Au22, Au23, Au24, Au31, Au32, Au33, Au34, NA, XN1, YN2, ZN3, Jacobi, &
        &        KexiX, EitaY, TaoZ, MeshX, MeshY, MeshZ, Beita, SigmaX1, SigmaX2, &
        &        SigmaY1, SigmaY2, SigmaZ1, SigmaZ2, N1, N2, N3, PX, PY, PZ, NPX, NPY, &
        &        NPZ, NPML, T, C0, ModelFlag )
        USE AcousticInitialCondition
        IMPLICIT NONE
        INTEGER:: I, J, K
        INTEGER, INTENT(IN):: ModelFlag
        INTEGER, INTENT(IN):: NA, XN1, YN2, ZN3
        INTEGER, INTENT(IN):: N1, N2, N3
        INTEGER, INTENT(IN):: PX, PY, PZ, NPX, NPY, NPZ
        INTEGER, INTENT(IN):: NPML
        REAL(KIND=8), INTENT(IN):: T
        REAL(KIND=8), INTENT(IN):: Beita
        REAL(KIND=8), INTENT(IN):: Jacobi( -(NPML-1):(N1+NPML),-(NPML-1):(N2+NPML),-(NPML-1):(N…
        REAL(KIND=8), INTENT(IN):: KexiX( -(NPML-1) : (N1 + NPML) )
```

```
REAL(KIND=8), INTENT(IN):: EitaY( -(NPML-1) : (N2 + NPML) )
REAL(KIND=8), INTENT(IN):: TaoZ( -(NPML-1) : (N3 + NPML) )
REAL(KIND=8), INTENT(IN):: MeshX( -(NPML-1) : ( N1 + NPML ) )
REAL(KIND=8), INTENT(IN):: MeshY( -(NPML-1) : ( N2 + NPML ) )
REAL(KIND=8), INTENT(IN):: MeshZ( -(NPML-1) : ( N3 + NPML ) )
REAL(KIND=8), INTENT(IN):: SigmaX1( -(NPML-1) : 0 )
REAL(KIND=8), INTENT(IN):: SigmaX2( 1 : NPML )
REAL(KIND=8), INTENT(IN):: SigmaY1( -(NPML-1) : 0 )
REAL(KIND=8), INTENT(IN):: SigmaY2( 1 : NPML )
REAL(KIND=8), INTENT(IN):: SigmaZ1( -(NPML-1) : 0 )
REAL(KIND=8), INTENT(IN):: SigmaZ2( 1 : NPML )
REAL(KIND=8), INTENT(IN):: C0( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(IN):: F1( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(IN):: F2( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(IN):: F3( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(IN):: F4( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(IN):: Au11( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(IN):: Au12( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(IN):: Au13( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(IN):: Au14( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(IN):: Au21( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(IN):: Au22( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(IN):: Au23( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(IN):: Au24( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(IN):: Au31( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(IN):: Au32( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(IN):: Au33( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(IN):: Au34( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: S1( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: S2( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: S3( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: S4( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu11( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu12( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu13( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu14( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu21( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu22( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu23( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu24( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu31( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu32( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu33( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu34( 1 : XN1, 1 : YN2, 1 : ZN3 )
INTEGER:: I0, J0, K0
!
!get the start point coordinates for MPI slice
I0 = XN1 * PX
J0 = YN2 * PY
K0 = ZN3 * PZ
!
S1 = 0.0d0
DO K = 1, ZN3
```

```
     DO J = 1, YN2
        DO I = 1, XN1
           S1( I, J, K ) = Jacobi( I+I0-NPML, J+J0-NPML, K+ModelFlag*(K0-NPML) ) * &
  &                A * DEXP( B * ( ( MeshX( I+I0-NPML ) - X0 )**2 + &
  &                   ( MeshY( J+J0-NPML )-Y0 )**2 + &
  &                      ( MeshZ( K+ModelFlag*(K0-NPML) )-Z0 )**2 ) ) &
  &                         * DSIN( Omega * T ) / C0( I, J, K )**2
        END DO
     END DO
  END DO
  ! PML zone caculation
!   S1 = 0.0d0
  S2 = 0.0d0
  S3 = 0.0d0
  S4 = 0.0d0
  SAu11 = 0.0d0
  SAu12 = 0.0d0
  SAu13 = 0.0d0
  SAu14 = 0.0d0
  SAu21 = 0.0d0
  SAu22 = 0.0d0
  SAu23 = 0.0d0
  SAu24 = 0.0d0
  SAu31 = 0.0d0
  SAu32 = 0.0d0
  SAu33 = 0.0d0
  SAu34 = 0.0d0
  !
  ! far-field PML source region( outer zone )
  IF ( (PY == 0) ) THEN
     !forward face
     DO J = 1, NPML
        S1( 1 : XN1, J, 1 : ZN3 ) = Au21( 1 : XN1, J, 1 : ZN3 ) * &
  &              SigmaY1( J - NPML ) + S1( 1 : XN1, J, 1 : ZN3 )
        SAu21( 1 : XN1, J, 1 : ZN3 ) = Au21( 1 : XN1, J, 1 : ZN3 ) * &
  &              SigmaY1( J - NPML )
        !
        S2( 1 : XN1, J, 1 : ZN3 ) = Au22( 1 : XN1, J, 1 : ZN3 ) * &
  &              SigmaY1( J - NPML ) + S2( 1 : XN1, J, 1 : ZN3 )
        SAu22( 1 : XN1, J, 1 : ZN3 ) = Au22( 1 : XN1, J, 1 : ZN3 ) * &
  &              SigmaY1( J - NPML )
        !
        S3( 1 : XN1, J, 1 : ZN3 ) = Au23( 1 : XN1, J, 1 : ZN3 ) * &
  &              SigmaY1( J - NPML ) + S3( 1 : XN1, J, 1 : ZN3 )
        SAu23( 1 : XN1, J, 1 : ZN3 ) = Au23( 1 : XN1, J, 1 : ZN3 ) * &
  &              SigmaY1( J - NPML )
        !
        S4( 1 : XN1, J, 1 : ZN3 ) = Au24( 1 : XN1, J, 1 : ZN3 ) * &
  &              SigmaY1( J - NPML ) + S4( 1 : XN1, J, 1 : ZN3 )
        SAu24( 1 : XN1, J, 1 : ZN3 ) = Au24( 1 : XN1, J, 1 : ZN3 ) * &
  &              SigmaY1( J - NPML )
     END DO
  END IF
```

```
    IF ( PY == NPY - 1 ) THEN
      !rearward face
      DO J = YN2 - ( NPML - 1 ), YN2
        S1( 1 : XN1, J, 1 : ZN3 ) = Au21( 1 : XN1, J, 1 : ZN3 ) * &
    &             SigmaY2( J - YN2 + NPML ) + S1( 1 : XN1, J, 1 : ZN3 )
        SAu21( 1 : XN1, J, 1 : ZN3 ) = Au21( 1 : XN1, J, 1 : ZN3 ) * &
    &             SigmaY2( J - YN2 + NPML )
        !
        S2( 1 : XN1, J, 1 : ZN3 ) = Au22( 1 : XN1, J, 1 : ZN3 ) * &
    &             SigmaY2( J - YN2 + NPML ) + S2( 1 : XN1, J, 1 : ZN3 )
        SAu22( 1 : XN1, J, 1 : ZN3 ) = Au22( 1 : XN1, J, 1 : ZN3 ) * &
    &             SigmaY2( J - YN2 + NPML )
        !
        S3( 1 : XN1, J, 1 : ZN3 ) = Au23( 1 : XN1, J, 1 : ZN3 ) * &
    &             SigmaY2( J - YN2 + NPML ) + S3( 1 : XN1, J, 1 : ZN3 )
        SAu23( 1 : XN1, J, 1 : ZN3 ) = Au23( 1 : XN1, J, 1 : ZN3 ) * &
    &             SigmaY2( J - YN2 + NPML )
        !
        S4( 1 : XN1, J, 1 : ZN3 ) = Au24( 1 : XN1, J, 1 : ZN3 ) * &
    &             SigmaY2( J - YN2 + NPML ) + S4( 1 : XN1, J, 1 : ZN3 )
        SAu24( 1 : XN1, J, 1 : ZN3 ) = Au24( 1 : XN1, J, 1 : ZN3 ) * &
    &             SigmaY2( J - YN2 + NPML )
      END DO
    END IF
    IF ( PX == 0 ) THEN
      !left face
      DO I = 1, NPML
        S1( I, 1 : YN2, 1 : ZN3 ) = Au11( I, 1 : YN2, 1 : ZN3 ) * SigmaX1( I - NPML ) &
    &             + SigmaX1( I - NPML ) * Beita * F1( I, 1 : YN2, 1 : ZN3 ) / &
    &               KexiX( I -NPML ) + S1( I, 1 : YN2, 1 : ZN3 )
        SAu11( I, 1 : YN2, 1 : ZN3 ) = Au11( I, 1 : YN2, 1 : ZN3 ) * &
    &             SigmaX1( I - NPML ) + SigmaX1( I - NPML ) * Beita * &
    &               F1( I, 1 : YN2, 1 : ZN3 ) / KexiX( I -NPML )
        !
        S2( I, 1 : YN2, 1 : ZN3 ) = Au12( I, 1 : YN2, 1 : ZN3 ) * SigmaX1( I - NPML ) &
    &             + SigmaX1( I - NPML ) * Beita * F2( I, 1 : YN2, 1 : ZN3 ) / &
    &               KexiX( I -NPML ) + S2( I, 1 : YN2, 1 : ZN3 )
        SAu12( I, 1 : YN2, 1 : ZN3 ) = Au12( I, 1 : YN2, 1 : ZN3 ) * &
    &             SigmaX1( I - NPML ) + SigmaX1( I - NPML ) * Beita * &
    &               F2( I, 1 : YN2, 1 : ZN3 ) / KexiX( I -NPML )
        !
        S3( I, 1 : YN2, 1 : ZN3 ) = Au13( I, 1 : YN2, 1 : ZN3 ) * SigmaX1( I - NPML ) &
    &             + SigmaX1( I - NPML ) * Beita * F3( I, 1 : YN2, 1 : ZN3 ) / &
    &               KexiX( I -NPML ) + S3( I, 1 : YN2, 1 : ZN3 )
        SAu13( I, 1 : YN2, 1 : ZN3 ) = Au13( I, 1 : YN2, 1 : ZN3 ) * &
    &             SigmaX1( I - NPML ) + SigmaX1( I - NPML ) * Beita * &
    &               F3( I, 1 : YN2, 1 : ZN3 ) / KexiX( I -NPML )
        !
        S4( I, 1 : YN2, 1 : ZN3 ) = Au14( I, 1 : YN2, 1 : ZN3 ) * SigmaX1( I - NPML ) &
    &             + SigmaX1( I - NPML ) * Beita * F4( I, 1 : YN2, 1 : ZN3 ) / &
    &               KexiX( I -NPML ) + S4( I, 1 : YN2, 1 : ZN3 )
        SAu14( I, 1 : YN2, 1 : ZN3 ) = Au14( I, 1 : YN2, 1 : ZN3 ) * &
    &             SigmaX1( I - NPML ) + SigmaX1( I - NPML ) * Beita * &
```

```
     &                 F4( I, 1 : YN2, 1 : ZN3 ) / KexiX( I -NPML )
        END DO
     END IF
     IF ( PX == NPX - 1 ) THEN
        !right face
        DO I = XN1 - ( NPML - 1 ), XN1
           S1( I, 1 : YN2, 1 : ZN3 ) = Au11( I, 1 : YN2, 1 : ZN3 ) * &
     &              SigmaX2( I - XN1 + NPML ) + SigmaX2( I-XN1+NPML ) * &
     &                 Beita * F1( I, 1 : YN2, 1 : ZN3 ) / KexiX( I-XN1+NPML ) &
     &                    + S1( I, 1 : YN2, 1 : ZN3 )
           SAu11( I, 1 : YN2, 1 : ZN3 ) = Au11( I, 1 : YN2, 1 : ZN3 ) * &
     &              SigmaX2( I - XN1 + NPML ) + SigmaX2( I-XN1+NPML ) * &
     &                 Beita * F1( I, 1 : YN2, 1 : ZN3 ) / KexiX( I-XN1+NPML )
           !
           S2( I, 1 : YN2, 1 : ZN3 ) = Au12( I, 1 : YN2, 1 : ZN3 ) * &
     &              SigmaX2( I - XN1 + NPML ) + SigmaX2( I-XN1+NPML ) * &
     &                 Beita * F2( I, 1 : YN2, 1 : ZN3 ) / KexiX( I-XN1+NPML ) &
     &                    + S2( I, 1 : YN2, 1 : ZN3 )
           SAu12( I, 1 : YN2, 1 : ZN3 ) = Au12( I, 1 : YN2, 1 : ZN3 ) * &
     &              SigmaX2( I - XN1 + NPML ) + SigmaX2( I-XN1+NPML ) * &
     &                 Beita * F2( I, 1 : YN2, 1 : ZN3 ) / KexiX( I-XN1+NPML )
           !
           S3( I, 1 : YN2, 1 : ZN3 ) = Au13( I, 1 : YN2, 1 : ZN3 ) * &
     &              SigmaX2( I - XN1 + NPML ) + SigmaX2( I-XN1+NPML ) * &
     &                 Beita * F3( I, 1 : YN2, 1 : ZN3 ) / KexiX( I-XN1+NPML ) &
     &                    + S3( I, 1 : YN2, 1 : ZN3 )
           SAu13( I, 1 : YN2, 1 : ZN3 ) = Au13( I, 1 : YN2, 1 : ZN3 ) * &
     &              SigmaX2( I - XN1 + NPML ) + SigmaX2( I-XN1+NPML ) * &
     &                 Beita * F3( I, 1 : YN2, 1 : ZN3 ) / KexiX( I-XN1+NPML )
           !
           S4( I, 1 : YN2, 1 : ZN3 ) = Au14( I, 1 : YN2, 1 : ZN3 ) * &
     &              SigmaX2( I - XN1 + NPML ) + SigmaX2( I-XN1+NPML ) * &
     &                 Beita * F4( I, 1 : YN2, 1 : ZN3 ) / KexiX( I-XN1+NPML ) &
     &                    + S4( I, 1 : YN2, 1 : ZN3 )
           SAu14( I, 1 : YN2, 1 : ZN3 ) = Au14( I, 1 : YN2, 1 : ZN3 ) * &
     &              SigmaX2( I - XN1 + NPML ) + SigmaX2( I-XN1+NPML ) * &
     &                 Beita * F4( I, 1 : YN2, 1 : ZN3 ) / KexiX( I-XN1+NPML )
        !
        END DO
     END IF
     !
     IF ( ModelFlag == 1 ) THEN
        ! 3-D Model
        IF ( PZ == 0 ) THEN
           !lower face
           DO K = 1, NPML
              S1( 1 : XN1, 1 : YN2, K ) = Au31( 1 : XN1, 1 : YN2, K ) * &
     &              SigmaZ1( K - NPML ) + S1( 1 : XN1, 1 : YN2, K )
              SAu31( 1 : XN1, 1 : YN2, K ) = Au31( 1 : XN1, 1 : YN2, K ) * &
     &              SigmaZ1( K - NPML )
              !
              S2( 1 : XN1, 1 : YN2, K ) = Au32( 1 : XN1, 1 : YN2, K ) * &
     &              SigmaZ1( K - NPML ) + S2( 1 : XN1, 1 : YN2, K )
```

```
            SAu32( 1 : XN1, 1 : YN2, K ) = Au32( 1 : XN1, 1 : YN2, K ) * &
     &           SigmaZ1( K - NPML )
            !
            S3( 1 : XN1, 1 : YN2, K ) = Au33( 1 : XN1, 1 : YN2, K ) * &
     &           SigmaZ1( K - NPML ) + S3( 1 : XN1, 1 : YN2, K )
            SAu33( 1 : XN1, 1 : YN2, K ) = Au33( 1 : XN1, 1 : YN2, K ) * &
     &           SigmaZ1( K - NPML )
            !
            S4( 1 : XN1, 1 : YN2, K ) = Au34( 1 : XN1, 1 : YN2, K ) * &
     &           SigmaZ1( K - NPML ) + S4( 1 : XN1, 1 : YN2, K )
            SAu34( 1 : XN1, 1 : YN2, K ) = Au34( 1 : XN1, 1 : YN2, K ) * &
     &           SigmaZ1( K - NPML )
          END DO
        END IF
        IF ( PZ == NPZ - 1 ) THEN
          !upper face
          DO K = ZN3 - ( NPML - 1 ), ZN3
            S1( 1 : XN1, 1 : YN2, K ) = Au31( 1 : XN1, 1 : YN2, K ) * &
     &           SigmaZ2( K - ZN3 + NPML ) + S1( 1 : XN1, 1 : YN2, K )
            SAu31( 1 : XN1, 1 : YN2, K ) = Au31( 1 : XN1, 1 : YN2, K ) * &
     &           SigmaZ2( K - ZN3 + NPML )
            !
            S2( 1 : XN1, 1 : YN2, K ) = Au32( 1 : XN1, 1 : YN2, K ) * &
     &           SigmaZ2( K - ZN3 + NPML ) + S2( 1 : XN1, 1 : YN2, K )
            SAu32( 1 : XN1, 1 : YN2, K ) = Au32( 1 : XN1, 1 : YN2, K ) * &
     &           SigmaZ2( K - ZN3 + NPML )
            !
            S3( 1 : XN1, 1 : YN2, K ) = Au33( 1 : XN1, 1 : YN2, K ) * &
     &           SigmaZ2( K - ZN3 + NPML ) + S3( 1 : XN1, 1 : YN2, K )
            SAu33( 1 : XN1, 1 : YN2, K ) = Au33( 1 : XN1, 1 : YN2, K ) * &
     &           SigmaZ2( K - ZN3 + NPML )
            !
            S4( 1 : XN1, 1 : YN2, K ) = Au34( 1 : XN1, 1 : YN2, K) * &
     &           SigmaZ2( K - ZN3 + NPML ) + S4( 1 : XN1, 1 : YN2, K )
            SAu34( 1 : XN1, 1 : YN2, K ) = Au34( 1 : XN1, 1 : YN2, K ) * &
     &           SigmaZ2( K - ZN3 + NPML )
          END DO
        END IF
      END IF
      !
      ! the PML region for the inner of solid

    END SUBROUTINE PMLSource
    ! ------------------------------
    ! >>>>>>>>>>>>>.
    SUBROUTINE SolidPMLSource( S1, S2, S3, S4, SAu11, SAu12, SAu13, SAu14, SAu21, &
     &         SAu22, SAu23, SAu24, SAu31, SAu32, SAu33, SAu34, F1, F2, F3, F4, &
     &         Au11, Au12, Au13, Au14, Au21, Au22, Au23, Au24, Au31, Au32, Au33, &
     &         Au34, NA, XN1, YN2, ZN3, MeshX, MeshY, MeshZ, NPML, N1, N2, N3, &
     &         CutPoints, StartEnd, Centre, DeltaXYZ, R_PML, SolidAfa, SolidSigma0, &
     &         PX, PY, PZ, ModelFlag )
      ! ------------------------introduction--------------------------
      ! This subroutine can get the source of the solid internal for the
```

```
! PML equations.
!-------------------------------------------------------------
IMPLICIT NONE
INTEGER:: I, J, K
INTEGER, INTENT(IN):: ModelFlag
INTEGER, INTENT(IN):: PX, PY, PZ
INTEGER, INTENT(IN):: NA, XN1, YN2, ZN3
INTEGER, INTENT(IN):: NPML, N1, N2, N3
INTEGER, INTENT(IN):: StartEnd( 3, 3 )
REAL(KIND=8), INTENT(IN):: CutPoints( 3, 4 )
REAL(KIND=8), INTENT(IN):: Centre( 3 )
REAL(KIND=8), INTENT(IN):: R_PML, DeltaXYZ
REAL(KIND=8), INTENT(IN):: SolidAfa, SolidSigma0
REAL(KIND=8), INTENT(IN):: Au11( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(IN):: Au12( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(IN):: Au13( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(IN):: Au14( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(IN):: Au21( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(IN):: Au22( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(IN):: Au23( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(IN):: Au24( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(IN):: Au31( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(IN):: Au32( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(IN):: Au33( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(IN):: Au34( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(IN):: F1( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(IN):: F2( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(IN):: F3( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(IN):: F4( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(IN):: MeshX( -(NPML-1) : ( N1 + NPML ) )
REAL(KIND=8), INTENT(IN):: MeshY( -(NPML-1) : ( N2 + NPML ) )
REAL(KIND=8), INTENT(IN):: MeshZ( -(NPML-1) : ( N3 + NPML ) )
REAL(KIND=8), INTENT(OUT):: S1( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: S2( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: S3( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: S4( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu11( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu12( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu13( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu14( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu21( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu22( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu23( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu24( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu31( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu32( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu33( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(OUT):: SAu34( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), ALLOCATABLE:: SolidSigmaX( :, :, : )
REAL(KIND=8), ALLOCATABLE:: SolidSigmaY( :, :, : )
REAL(KIND=8), ALLOCATABLE:: SolidSigmaZ( :, :, : )
REAL(KIND=8):: SolidBeita
```

```
REAL(KIND=8):: Radius
INTEGER:: I0, J0, K0
INTEGER:: IFlag(2), JFlag(2), KFlag(2)
INTEGER:: SolidPMLPosition( 3, 2 )
!
SolidBeita = 0.0d0
I0 = PX * XN1
J0 = PY * YN2
K0 = PZ * ZN3
!
! get the solid PML zone position
DO I = 1, 3
   SolidPMLPosition( I, 1 ) = StartEnd( I, 1 ) + FLOOR( &
&         ( Centre(I)-R_PML-CutPoints( I, 2 ) ) / DeltaXYZ )
   SolidPMLPosition( I, 2 ) = StartEnd( I, 1 ) + FLOOR( &
&         ( Centre(I)+R_PML-CutPoints( I, 2 ) ) / DeltaXYZ )
END DO
IF ( ModelFlag == 0 ) THEN
   ! 2-D Model
   SolidPMLPosition( 3, : ) = 1
END IF
!
ALLOCATE( SolidSigmaX( SolidPMLPosition( 1, 1 ): SolidPMLPosition(1,2), &
&         SolidPMLPosition( 2, 1 ): SolidPMLPosition(2,2), &
&            SolidPMLPosition( 3, 1 ): SolidPMLPosition(3, 2) ) )
ALLOCATE( SolidSigmaY( SolidPMLPosition( 1, 1 ): SolidPMLPosition(1,2), &
&         SolidPMLPosition( 2, 1 ): SolidPMLPosition(2,2), &
&            SolidPMLPosition( 3, 1 ): SolidPMLPosition(3, 2) ) )
ALLOCATE( SolidSigmaZ( SolidPMLPosition( 1, 1 ): SolidPMLPosition(1,2), &
&         SolidPMLPosition( 2, 1 ): SolidPMLPosition(2,2), &
&            SolidPMLPosition( 3, 1 ): SolidPMLPosition(3, 2) ) )
!
IFlag( 1 ) = MAX( 1 + I0 - NPML, SolidPMLPosition( 1, 1 ) )
IFlag( 2 ) = MIN( XN1 + I0 - NPML, SolidPMLPosition( 1, 2 ) )
JFlag( 1 ) = MAX( 1 + J0 - NPML, SolidPMLPosition( 2, 1 ) )
JFlag( 2 ) = MIN( YN2 + J0 - NPML, SolidPMLPosition( 2, 2 ) )
KFlag( 1 ) = MAX( 1 + K0 - NPML, SolidPMLPosition( 3, 1 ) )
KFlag( 2 ) = MIN( ZN3 + K0 - NPML, SolidPMLPosition( 3, 2 ) )
IF ( ModelFlag == 0 ) THEN
   ! 2-D Model
   KFlag = 1
END IF
!
! get the absorbing coefficient for the solid internal
!   and get the source term of the PML equations for
!       the solid internal
SolidSigmaX = 0.0d0
SolidSigmaY = 0.0d0
SolidSigmaZ = 0.0d0
DO K = KFlag(1), KFlag(2)
   DO J = JFlag(1), JFlag(2)
      DO I = IFlag(1), IFlag(2)
         Radius = DSQRT( ( MeshX(I) - Centre(1) )**2 + ( MeshY(J) - Centre(2) )**2 + &
```

```
     &          ( MeshZ(K) - Centre(3) )**2 )
            IF ( Radius .LE. R_PML ) THEN
              SolidSigmaX( I, J, K ) = SolidSigma0 * &
     &          ( ( R_PML - Radius ) / R_PML )**SolidAfa / DeltaXYZ
              SolidSigmaY( I, J, K ) = SolidSigma0 * &
     &          ( ( R_PML - Radius ) / R_PML )**SolidAfa / DeltaXYZ
              SolidSigmaZ( I, J, K ) = SolidSigma0 * &
     &          ( ( R_PML - Radius ) / R_PML )**SolidAfa / DeltaXYZ
              IF ( ModelFlag == 0 ) THEN
                ! 2-D Model
                k0 = 1
              ELSE
                ! 3-D Model
                K0 = K - K0 + NPML
              END IF
              S1( I-I0+NPML, J-J0+NPML, K0 ) = S1( I-I0+NPML, J-J0+NPML, K0 ) + &
     &          SolidSigmaX( I, J, K0 ) * Au11( I-I0+NPML, J-J0+NPML, K0 ) + &
     &          SolidSigmaY( I, J, K0 ) * Au21( I-I0+NPML, J-J0+NPML, K0 ) + &
     &          SolidSigmaZ( I, J, K0 ) * Au31( I-I0+NPML, J-J0+NPML, K0 ) + &
     &          SolidSigmaX( I, J, K0 ) * SolidBeita * F1( I-I0+NPML, J-J0+NPML, K0 )
              S2( I-I0+NPML, J-J0+NPML, K0 ) = S2( I-I0+NPML, J-J0+NPML, K0 ) + &
     &          SolidSigmaX( I, J, K0 ) * Au12( I-I0+NPML, J-J0+NPML, K0 ) + &
     &          SolidSigmaY( I, J, K0 ) * Au22( I-I0+NPML, J-J0+NPML, K0 ) + &
     &          SolidSigmaZ( I, J, K0 ) * Au32( I-I0+NPML, J-J0+NPML, K0 ) + &
     &          SolidSigmaX( I, J, K0 ) * SolidBeita * F2( I-I0+NPML, J-J0+NPML, K0 )
              S3( I-I0+NPML, J-J0+NPML, K0 ) = S3( I-I0+NPML, J-J0+NPML, K0 ) + &
     &          SolidSigmaX( I, J, K0 ) * Au13( I-I0+NPML, J-J0+NPML, K0 ) + &
     &          SolidSigmaY( I, J, K0 ) * Au23( I-I0+NPML, J-J0+NPML, K0 ) + &
     &          SolidSigmaZ( I, J, K0 ) * Au33( I-I0+NPML, J-J0+NPML, K0 ) + &
     &          SolidSigmaX( I, J, K0 ) * SolidBeita * F3( I-I0+NPML, J-J0+NPML, K0 )
              S4( I-I0+NPML, J-J0+NPML, K0 ) = S4( I-I0+NPML, J-J0+NPML, K0 ) + &
     &          SolidSigmaX( I, J, K0 ) * Au14( I-I0+NPML, J-J0+NPML, K0 ) + &
     &          SolidSigmaY( I, J, K0 ) * Au24( I-I0+NPML, J-J0+NPML, K0 ) + &
     &          SolidSigmaZ( I, J, K0 ) * Au34( I-I0+NPML, J-J0+NPML, K0 ) + &
     &          SolidSigmaX( I, J, K0 ) * SolidBeita * F4( I-I0+NPML, J-J0+NPML, K0 )
              SAu11( I-I0+NPML, J-J0+NPML, K0 ) = SAu11( I-I0+NPML, J-J0+NPML, K0 ) + &
     &          SolidSigmaX( I, J, K0 ) * Au11( I-I0+NPML, J-J0+NPML, K0 ) + &
     &          SolidSigmaX( I, J, K0 ) * SolidBeita * F1( I-I0+NPML, J-J0+NPML, K0 )
              SAu12( I-I0+NPML, J-J0+NPML, K0 ) = SAu12( I-I0+NPML, J-J0+NPML, K0 ) + &
     &          SolidSigmaX( I, J, K0 ) * Au12( I-I0+NPML, J-J0+NPML, K0 ) + &
     &          SolidSigmaX( I, J, K0 ) * SolidBeita * F2( I-I0+NPML, J-J0+NPML, K0 )
              SAu13( I-I0+NPML, J-J0+NPML, K0 ) = SAu13( I-I0+NPML, J-J0+NPML, K0 ) + &
     &          SolidSigmaX( I, J, K0 ) * Au13( I-I0+NPML, J-J0+NPML, K0 ) + &
     &          SolidSigmaX( I, J, K0 ) * SolidBeita * F3( I-I0+NPML, J-J0+NPML, K0 )
              SAu14( I-I0+NPML, J-J0+NPML, K0 ) = SAu14( I-I0+NPML, J-J0+NPML, K0 ) + &
     &          SolidSigmaX( I, J, K0 ) * Au14( I-I0+NPML, J-J0+NPML, K0 ) + &
     &          SolidSigmaX( I, J, K0 ) * SolidBeita * F4( I-I0+NPML, J-J0+NPML, K0 )
              SAu21( I-I0+NPML, J-J0+NPML, K0 ) = SAu21( I-I0+NPML, J-J0+NPML, K0 ) + &
     &          SolidSigmaY( I, J, K0 ) * Au21( I-I0+NPML, J-J0+NPML, K0 )
              SAu22( I-I0+NPML, J-J0+NPML, K0 ) = SAu22( I-I0+NPML, J-J0+NPML, K0 ) + &
     &          SolidSigmaY( I, J, K0 ) * Au22( I-I0+NPML, J-J0+NPML, K0 )
              SAu23( I-I0+NPML, J-J0+NPML, K0 ) = SAu23( I-I0+NPML, J-J0+NPML, K0 ) + &
     &          SolidSigmaY( I, J, K0 ) * Au23( I-I0+NPML, J-J0+NPML, K0 )
```

```fortran
                SAu24( I-I0+NPML, J-J0+NPML, K0 ) = SAu24( I-I0+NPML, J-J0+NPML, K0 ) + &
    &               SolidSigmaX( I, J, K0 ) * Au24( I-I0+NPML, J-J0+NPML, K0 )
                SAu31( I-I0+NPML, J-J0+NPML, K0 ) = SAu31( I-I0+NPML, J-J0+NPML, K0 ) + &
    &               SolidSigmaZ( I, J, K0 ) * Au31( I-I0+NPML, J-J0+NPML, K0 )
                SAu32( I-I0+NPML, J-J0+NPML, K0 ) = SAu32( I-I0+NPML, J-J0+NPML, K0 ) + &
    &               SolidSigmaZ( I, J, K0 ) * Au32( I-I0+NPML, J-J0+NPML, K0 )
                SAu33( I-I0+NPML, J-J0+NPML, K0 ) = SAu33( I-I0+NPML, J-J0+NPML, K0 ) + &
    &               SolidSigmaZ( I, J, K0 ) * Au33( I-I0+NPML, J-J0+NPML, K0 )
                SAu34( I-I0+NPML, J-J0+NPML, K0 ) = SAu34( I-I0+NPML, J-J0+NPML, K0 ) + &
    &               SolidSigmaZ( I, J, K0 ) * Au34( I-I0+NPML, J-J0+NPML, K0 )
             END IF
          END DO
       END DO
    END DO

  END SUBROUTINE SolidPMLSource
  ! ---------------------------------------
  ! >>>>>>>>>>>>>>
  SUBROUTINE LDDRK( Q1, Q2, Q3, Q4, F1, F2, F3, F4, G1, G2, G3, G4, &
    &    H1, H2, H3, H4, Au11, Au12, Au13, Au14, Au21, Au22, Au23, &
    &       Au24, Au31, Au32, Au33, Au34, S1, S2, S3, S4, SAu11, &
    &       SAu12, SAu13, SAu14, SAu21, SAu22, SAu23, SAu24, SAu31, &
    &       SAu32, SAu33, SAu34, U0, V0, W0, P0, ROU0, C0, Jacobi, KexiX, &
    &       EitaY, TaoZ, MeshX, MeshY, MeshZ, SigmaX1, SigmaX2, SigmaY1, SigmaY2, &
    &       SigmaZ1, SigmaZ2, Beita, NA, XN1, YN2, ZN3, N1, N2, N3, NPML, DeltaT, &
    &       DeltaXYZ, PX, PY, PZ, NPX, NPY, NPZ, MYLEFT, MYRIGHT, MYFORWARD, &
    &       MYREAR, MYUPPER, MYLOWER, XTYPE, YTYPE, ZTYPE, Flag, Loops, &
    &       ModelFlag, BCFlag )
    !---------------------------------introduction------------------------------------
    !This program can get the next flow variables using LDDRK4-6 time marching algorithm.
    !Here when Flag = 0, using 4- stages LDDRK scheme, and Flag = 1 using 6- stage LDDRK
    !scheme.
    !------------------------------------------------------------------------------
    USE LDDRK46Coefficient
    USE M_Spatial_dis
    IMPLICIT NONE
    INTEGER:: I, J, K, I0, LDDRK46
    INTEGER, INTENT(IN):: BCFlag
    INTEGER, INTENT(IN):: ModelFlag, Loops
    INTEGER, INTENT(IN):: NA, XN1, YN2, ZN3, N1, N2, N3, NPML
    INTEGER, INTENT(IN):: NPX, NPY, NPZ
    INTEGER, INTENT(IN):: PX, PY, PZ
    INTEGER, INTENT(IN):: Flag
    INTEGER, INTENT(IN):: MYLEFT, MYRIGHT, MYFORWARD, MYREAR, MYUPPER, MYLOWER
    INTEGER, INTENT(IN):: XTYPE, YTYPE, ZTYPE
    REAL(KIND=8), INTENT(IN):: DeltaT, DeltaXYZ
    REAL(KIND=8), INTENT(IN):: Beita
    REAL(KIND=8), INTENT(IN):: U0( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
    REAL(KIND=8), INTENT(IN):: V0( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
    REAL(KIND=8), INTENT(IN):: W0( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
    REAL(KIND=8), INTENT(IN):: P0( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
    REAL(KIND=8), INTENT(IN):: ROU0( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
    REAL(KIND=8), INTENT(IN):: C0( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
```

```
REAL(KIND=8), INTENT(IN):: MeshX( -(NPML-1) : ( N1 + NPML ) )
REAL(KIND=8), INTENT(IN):: MeshY( -(NPML-1) : ( N2 + NPML ) )
REAL(KIND=8), INTENT(IN):: MeshZ( -(NPML-1) : ( N3 + NPML ) )
REAL(KIND=8), INTENT(IN):: Jacobi( -(NPML-1):(N1+NPML),-(NPML-1):(N2+NPML),-(NPML-1):(N…
REAL(KIND=8), INTENT(IN):: KexiX( - ( NPML - 1 ) : ( N1 + NPML ) )
REAL(KIND=8), INTENT(IN):: EitaY( - ( NPML - 1 ) : ( N2 + NPML ) )
REAL(KIND=8), INTENT(IN):: TaoZ( - ( NPML - 1 ) : ( N3 + NPML ) )
REAL(KIND=8), INTENT(IN):: SigmaX1( -(NPML-1) : 0 )
REAL(KIND=8), INTENT(IN):: SigmaX2( 1 : NPML )
REAL(KIND=8), INTENT(IN):: SigmaY1( -(NPML-1) : 0 )
REAL(KIND=8), INTENT(IN):: SigmaY2( 1 : NPML )
REAL(KIND=8), INTENT(IN):: SigmaZ1( -(NPML-1) : 0 )
REAL(KIND=8), INTENT(IN):: SigmaZ2( 1 : NPML )
REAL(KIND=8), INTENT(INOUT):: S1( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: S2( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: S3( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: S4( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: SAu11( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: SAu12( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: SAu13( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: SAu14( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: SAu21( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: SAu22( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: SAu23( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: SAu24( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: SAu31( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: SAu32( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: SAu33( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: SAu34( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: F1( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+N…
REAL(KIND=8), INTENT(INOUT):: F2( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+N…
REAL(KIND=8), INTENT(INOUT):: F3( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+N…
REAL(KIND=8), INTENT(INOUT):: F4( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+N…
REAL(KIND=8), INTENT(INOUT):: G1( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+N…
REAL(KIND=8), INTENT(INOUT):: G2( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+N…
REAL(KIND=8), INTENT(INOUT):: G3( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+N…
REAL(KIND=8), INTENT(INOUT):: G4( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+N…
REAL(KIND=8), INTENT(INOUT):: H1( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+N…
REAL(KIND=8), INTENT(INOUT):: H2( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+N…
REAL(KIND=8), INTENT(INOUT):: H3( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+N…
REAL(KIND=8), INTENT(INOUT):: H4( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+N…
REAL(KIND=8), INTENT(INOUT):: Q1( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+N…
REAL(KIND=8), INTENT(INOUT):: Q2( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+N…
REAL(KIND=8), INTENT(INOUT):: Q3( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+N…
REAL(KIND=8), INTENT(INOUT):: Q4( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+N…
REAL(KIND=8), INTENT(INOUT):: Au11( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: Au12( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: Au13( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: Au14( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: Au21( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: Au22( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: Au23( 1 : XN1, 1 : YN2, 1 : ZN3 )
```

```
REAL(KIND=8), INTENT(INOUT):: Au24( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: Au31( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: Au32( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: Au33( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8), INTENT(INOUT):: Au34( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8):: Q01( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8):: Q02( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8):: Q03( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8):: Q04( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8):: Au11_0( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8):: Au12_0( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8):: Au13_0( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8):: Au14_0( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8):: Au21_0( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8):: Au22_0( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8):: Au23_0( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8):: Au24_0( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8):: Au31_0( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8):: Au32_0( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8):: Au33_0( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8):: Au34_0( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8):: K1( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8):: K2( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8):: K3( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8):: K4( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8):: KAu11( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8):: KAu12( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8):: KAu13( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8):: KAu14( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8):: KAu21( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8):: KAu22( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8):: KAu23( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8):: KAu24( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8):: KAu31( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8):: KAu32( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8):: KAu33( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8):: KAu34( 1 : XN1, 1 : YN2, 1 : ZN3 )
REAL(KIND=8):: DeriveX( 1 : XN1 ), DeriveY( 1 : YN2 ), DeriveZ( 1 : ZN3 )
REAL(KIND=8):: AFA( 2:7 ), C( 2:7 )
INTEGER:: Size0(2)
REAL(KIND=8):: T
!!
T = Loops * DeltaT
!
IF ( Flag == 0 ) THEN
   LDDRK46 = 4
   AFA( 2 : 5 ) = AFA1( 2 : 5 )
   C( 2 : 5 ) = C1( 2 : 5 )
ELSE
   LDDRK46 = 6
   AFA( 2 : 7 ) = AFA2( 2 : 7 )
   C( 2 : 7 ) = C2( 2 : 7 )
END IF
```

```
!!
Q01 = Q1
Q02 = Q2
Q03 = Q3
Q04 = Q4
Au11_0 = Au11
Au12_0 = Au12
Au13_0 = Au13
Au14_0 = Au14
Au21_0 = Au21
Au22_0 = Au22
Au23_0 = Au23
Au24_0 = Au24
Au31_0 = Au31
Au32_0 = Au32
Au33_0 = Au33
Au34_0 = Au34
Q1 = 0.0d0
Q2 = 0.0d0
Q3 = 0.0d0
Q4 = 0.0d0
Au11 = 0.0d0
Au12 = 0.0d0
Au13 = 0.0d0
Au14 = 0.0d0
Au21 = 0.0d0
Au22 = 0.0d0
Au23 = 0.0d0
Au24 = 0.0d0
Au31 = 0.0d0
Au32 = 0.0d0
Au33 = 0.0d0
Au34 = 0.0d0
!!
DO I0 = 2, LDDRK46 + 1
  K1( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Q01( 1 : XN1, 1 : YN2, 1 : ZN3 )
  K2( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Q02( 1 : XN1, 1 : YN2, 1 : ZN3 )
  k3( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Q03( 1 : XN1, 1 : YN2, 1 : ZN3 )
  K4( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Q04( 1 : XN1, 1 : YN2, 1 : ZN3 )
  KAu11( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Au11_0( 1 : XN1, 1 : YN2, 1 : ZN3 )
  KAu12( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Au12_0( 1 : XN1, 1 : YN2, 1 : ZN3 )
  kAu13( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Au13_0( 1 : XN1, 1 : YN2, 1 : ZN3 )
  KAu14( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Au14_0( 1 : XN1, 1 : YN2, 1 : ZN3 )
  KAu21( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Au21_0( 1 : XN1, 1 : YN2, 1 : ZN3 )
  KAu22( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Au22_0( 1 : XN1, 1 : YN2, 1 : ZN3 )
  kAu23( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Au23_0( 1 : XN1, 1 : YN2, 1 : ZN3 )
  KAu24( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Au24_0( 1 : XN1, 1 : YN2, 1 : ZN3 )
  KAu31( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Au31_0( 1 : XN1, 1 : YN2, 1 : ZN3 )
  KAu32( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Au32_0( 1 : XN1, 1 : YN2, 1 : ZN3 )
  kAu33( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Au33_0( 1 : XN1, 1 : YN2, 1 : ZN3 )
  KAu34( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Au34_0( 1 : XN1, 1 : YN2, 1 : ZN3 )
  !!
  !x-direction
```

```
        Size0(1) = - ( NA - 1 )
        Size0(2) = XN1 + NA
        DO K = 1, ZN3
          DO J = 1, YN2
            CALL DRP7( DeriveX, F1( :, J, K ), DeltaXYZ, 1, XN1, Size0 )
            K1( 1:XN1, J, K ) = K1( 1:XN1, J, K ) - afa( I0 ) * DeltaT * DeriveX
            KAu11( 1:XN1, J, K ) = KAu11( 1:XN1, J, K ) - afa( I0 ) * DeltaT * DeriveX
            !!
            CALL DRP7( DeriveX, F2( :, J, K ), DeltaXYZ, 1, XN1, Size0 )
            K2( 1:XN1, J, K ) = K2( 1:XN1, J, K ) - afa( I0 ) * DeltaT * DeriveX
            KAu12( 1:XN1, J, K ) = KAu12( 1:XN1, J, K ) - afa( I0 ) * DeltaT * DeriveX
            !!
            CALL DRP7( DeriveX, F3( :, J, K ), DeltaXYZ, 1, XN1, Size0 )
            K3( 1:XN1, J, K ) = K3( 1:XN1, J, K ) - afa( I0 ) * DeltaT * DeriveX
            KAu13( 1:XN1, J, K ) = KAu13( 1:XN1, J, K ) - afa( I0 ) * DeltaT * DeriveX
            !!
            CALL DRP7( DeriveX, F4( :, J, K ), DeltaXYZ, 1, XN1, Size0 )
            K4( 1:XN1, J, K ) = K4( 1:XN1, J, K ) - afa( I0 ) * DeltaT * DeriveX
            KAu14( 1:XN1, J, K ) = KAu14( 1:XN1, J, K ) - afa( I0 ) * DeltaT * DeriveX
          END DO
        END DO
        !!
        !y-direction
        Size0(1) = - ( NA - 1 )
        Size0(2) = YN2 + NA
        DO K = 1, ZN3
          DO I = 1, XN1
            CALL DRP7( DeriveY, G1( I, :, K ), DeltaXYZ, 1, YN2, Size0 )
            K1( I, 1:YN2, K ) = K1( I, 1:YN2, K ) - afa( I0 ) * DeltaT * DeriveY
            KAu21( I, 1:YN2, K ) = KAu21( I, 1:YN2, K ) - afa( I0 ) * DeltaT * DeriveY
            !!
            CALL DRP7( DeriveY, G2( I, :, K ), DeltaXYZ, 1, YN2, Size0 )
            K2( I, 1:YN2, K ) = K2( I, 1:YN2, K ) - afa( I0 ) * DeltaT * DeriveY
            KAu22( I, 1:YN2, K ) = KAu22( I, 1:YN2, K ) - afa( I0 ) * DeltaT * DeriveY
            !!
            CALL DRP7( DeriveY, G3( I, :, K ), DeltaXYZ, 1, YN2, Size0 )
            K3( I, 1:YN2, K ) = K3( I, 1:YN2, K ) - afa( I0 ) * DeltaT * DeriveY
            KAu23( I, 1:YN2, K ) = KAu23( I, 1:YN2, K ) - afa( I0 ) * DeltaT * DeriveY
            !!
            CALL DRP7( DeriveY, G4( I, :, K ), DeltaXYZ, 1, YN2, Size0 )
            K4( I, 1:YN2, K ) = K4( I, 1:YN2, K ) - afa( I0 ) * DeltaT * DeriveY
            KAu24( I, 1:YN2, K ) = KAu24( I, 1:YN2, K ) - afa( I0 ) * DeltaT * DeriveY
          END DO
        END DO
        !!
        !z-direction
        IF ( ModelFlag == 1 ) THEN
          ! 3-D Model
          Size0(1) = - ( NA - 1 )
          Size0(2) = ZN3 + NA
          DO I = 1, XN1
            DO J = 1, YN2
              CALL DRP7( DeriveZ, H1( I, J, : ), DeltaXYZ, 1, ZN3, Size0 )
```

```
                K1( I, J, 1:ZN3 ) = K1( I, J, 1:ZN3 ) - afa( I0 ) * DeltaT * DeriveZ
                KAu31( I, J, 1:ZN3 ) = KAu31( I, J, 1:ZN3 ) - afa( I0 ) * DeltaT * DeriveZ
                !!
                CALL DRP7( DeriveZ, H2( I, J, : ), DeltaXYZ, 1, ZN3, Size0 )
                K2( I, J, 1:ZN3 ) = K2( I, J, 1:ZN3 ) - afa( I0 ) * DeltaT * DeriveZ
                KAu32( I, J, 1:ZN3 ) = KAu32( I, J, 1:ZN3 ) - afa( I0 ) * DeltaT * DeriveZ
                !!
                CALL DRP7( DeriveZ, H3( I, J, : ), DeltaXYZ, 1, ZN3, Size0 )
                K3( I, J, 1:ZN3 ) = K3( I, J, 1:ZN3 ) - afa( I0 ) * DeltaT * DeriveZ
                KAu33( I, J, 1:ZN3 ) = KAu33( I, J, 1:ZN3 ) - afa( I0 ) * DeltaT * DeriveZ
                !!
                CALL DRP7( DeriveZ, H4( I, J, : ), DeltaXYZ, 1, ZN3, Size0 )
                K4( I, J, 1:ZN3 ) = K4( I, J, 1:ZN3 ) - afa( I0 ) * DeltaT * DeriveZ
                KAu34( I, J, 1:ZN3 ) = KAu34( I, J, 1:ZN3 ) - afa( I0 ) * DeltaT * DeriveZ
              END DO
            END DO
          END IF
          !!
          k1( 1 : XN1, 1 : YN2, 1 : ZN3 ) = k1( 1 : XN1, 1 : YN2, 1 : ZN3 ) - &
        &     S1( 1 : XN1, 1 : YN2, 1 : ZN3 ) * afa( I0 ) * DeltaT
          K2( 1 : XN1, 1 : YN2, 1 : ZN3 ) = K2( 1 : XN1, 1 : YN2, 1 : ZN3 ) - &
        &     S2( 1 : XN1, 1 : YN2, 1 : ZN3 ) * afa( I0 ) * DeltaT
          K3( 1 : XN1, 1 : YN2, 1 : ZN3 ) = K3( 1 : XN1, 1 : YN2, 1 : ZN3 ) - &
        &     S3( 1 : XN1, 1 : YN2, 1 : ZN3 ) * afa( I0 ) * DeltaT
          K4( 1 : XN1, 1 : YN2, 1 : ZN3 ) = K4( 1 : XN1, 1 : YN2, 1 : ZN3 ) - &
        &     S4( 1 : XN1, 1 : YN2, 1 : ZN3 ) * afa( I0 ) * DeltaT
          !
          KAu11( 1 : XN1, 1 : YN2, 1 : ZN3 ) = KAu11( 1 : XN1, 1 : YN2, 1 : ZN3 ) - &
        &     SAu11( 1 : XN1, 1 : YN2, 1 : ZN3 ) * afa( I0 ) * DeltaT
          KAu12( 1 : XN1, 1 : YN2, 1 : ZN3 ) = KAu12( 1 : XN1, 1 : YN2, 1 : ZN3 ) - &
        &     SAu12( 1 : XN1, 1 : YN2, 1 : ZN3 ) * afa( I0 ) * DeltaT
          KAu13( 1 : XN1, 1 : YN2, 1 : ZN3 ) = KAu13( 1 : XN1, 1 : YN2, 1 : ZN3 ) - &
        &     SAu13( 1 : XN1, 1 : YN2, 1 : ZN3 ) * afa( I0 ) * DeltaT
          KAu14( 1 : XN1, 1 : YN2, 1 : ZN3 ) = KAu14( 1 : XN1, 1 : YN2, 1 : ZN3 ) - &
        &     SAu14( 1 : XN1, 1 : YN2, 1 : ZN3 ) * afa( I0 ) * DeltaT
          !
          KAu21( 1 : XN1, 1 : YN2, 1 : ZN3 ) = KAu21( 1 : XN1, 1 : YN2, 1 : ZN3 ) - &
        &     SAu21( 1 : XN1, 1 : YN2, 1 : ZN3 ) * afa( I0 ) * DeltaT
          KAu22( 1 : XN1, 1 : YN2, 1 : ZN3 ) = KAu22( 1 : XN1, 1 : YN2, 1 : ZN3 ) - &
        &     SAu22( 1 : XN1, 1 : YN2, 1 : ZN3 ) * afa( I0 ) * DeltaT
          KAu23( 1 : XN1, 1 : YN2, 1 : ZN3 ) = KAu23( 1 : XN1, 1 : YN2, 1 : ZN3 ) - &
        &     SAu23( 1 : XN1, 1 : YN2, 1 : ZN3 ) * afa( I0 ) * DeltaT
          KAu24( 1 : XN1, 1 : YN2, 1 : ZN3 ) = KAu24( 1 : XN1, 1 : YN2, 1 : ZN3 ) - &
        &     SAu24( 1 : XN1, 1 : YN2, 1 : ZN3 ) * afa( I0 ) * DeltaT
          !
          KAu31( 1 : XN1, 1 : YN2, 1 : ZN3 ) = KAu31( 1 : XN1, 1 : YN2, 1 : ZN3 ) - &
        &     SAu31( 1 : XN1, 1 : YN2, 1 : ZN3 ) * afa( I0 ) * DeltaT
          KAu32( 1 : XN1, 1 : YN2, 1 : ZN3 ) = KAu32( 1 : XN1, 1 : YN2, 1 : ZN3 ) - &
        &     SAu32( 1 : XN1, 1 : YN2, 1 : ZN3 ) * afa( I0 ) * DeltaT
          KAu33( 1 : XN1, 1 : YN2, 1 : ZN3 ) = KAu33( 1 : XN1, 1 : YN2, 1 : ZN3 ) - &
        &     SAu33( 1 : XN1, 1 : YN2, 1 : ZN3 ) * afa( I0 ) * DeltaT
          KAu34( 1 : XN1, 1 : YN2, 1 : ZN3 ) = KAu34( 1 : XN1, 1 : YN2, 1 : ZN3 ) - &
        &     SAu34( 1 : XN1, 1 : YN2, 1 : ZN3 ) * afa( I0 ) * DeltaT
```

```
       !!
       !!
       Q1( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Q1( 1 : XN1, 1 : YN2, 1 : ZN3 ) + C( I0 ) &
     &      * k1( 1 : XN1, 1 : YN2, 1 : ZN3 )
       Q2( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Q2( 1 : XN1, 1 : YN2, 1 : ZN3 ) + C( I0 ) &
     &      * K2( 1 : XN1, 1 : YN2, 1 : ZN3 )
       Q3( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Q3( 1 : XN1, 1 : YN2, 1 : ZN3 ) + C( I0 ) &
     &      * K3( 1 : XN1, 1 : YN2, 1 : ZN3 )
       Q4( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Q4( 1 : XN1, 1 : YN2, 1 : ZN3 ) + C( I0 ) &
     &      * K4( 1 : XN1, 1 : YN2, 1 : ZN3 )
       Au11( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Au11( 1 : XN1, 1 : YN2, 1 : ZN3 ) + &
     &      C( I0 ) * KAu11( 1 : XN1, 1 : YN2, 1 : ZN3 )
       Au12( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Au12( 1 : XN1, 1 : YN2, 1 : ZN3 ) + &
     &      C( I0 ) * KAu12( 1 : XN1, 1 : YN2, 1 : ZN3 )
       Au13( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Au13( 1 : XN1, 1 : YN2, 1 : ZN3 ) + &
     &      C( I0 ) * KAu13( 1 : XN1, 1 : YN2, 1 : ZN3 )
       Au14( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Au14( 1 : XN1, 1 : YN2, 1 : ZN3 ) + &
     &      C( I0 ) * KAu14( 1 : XN1, 1 : YN2, 1 : ZN3 )
       Au21( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Au21( 1 : XN1, 1 : YN2, 1 : ZN3 ) + &
     &      C( I0 ) * KAu21( 1 : XN1, 1 : YN2, 1 : ZN3 )
       Au22( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Au22( 1 : XN1, 1 : YN2, 1 : ZN3 ) + &
     &      C( I0 ) * KAu22( 1 : XN1, 1 : YN2, 1 : ZN3 )
       Au23( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Au23( 1 : XN1, 1 : YN2, 1 : ZN3 ) + &
     &      C( I0 ) * KAu23( 1 : XN1, 1 : YN2, 1 : ZN3 )
       Au24( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Au24( 1 : XN1, 1 : YN2, 1 : ZN3 ) + &
     &      C( I0 ) * KAu24( 1 : XN1, 1 : YN2, 1 : ZN3 )
       Au31( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Au31( 1 : XN1, 1 : YN2, 1 : ZN3 ) + &
     &      C( I0 ) * KAu31( 1 : XN1, 1 : YN2, 1 : ZN3 )
       Au32( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Au32( 1 : XN1, 1 : YN2, 1 : ZN3 ) + &
     &      C( I0 ) * KAu32( 1 : XN1, 1 : YN2, 1 : ZN3 )
       Au33( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Au33( 1 : XN1, 1 : YN2, 1 : ZN3 ) + &
     &      C( I0 ) * KAu33( 1 : XN1, 1 : YN2, 1 : ZN3 )
       Au34( 1 : XN1, 1 : YN2, 1 : ZN3 ) = Au34( 1 : XN1, 1 : YN2, 1 : ZN3 ) + &
     &      C( I0 ) * KAu34( 1 : XN1, 1 : YN2, 1 : ZN3 )
       !IF ( I0 < LDDRK46 + 1 ) THEN
          !
          ! get the next F, G, H from resolved Q and auxiliary variables: Au
     !      CALL ConvertQtoFGH( K1, K2, K3, K4, F1, F2, F3, F4, G1, G2, G3, &
     !&              G4, H1, H2, H3, H4, U0, V0, W0, P0, ROU0, C0, Jacobi, KexiX, &
     !&              EitaY, TaoZ, NA, N1, N2, N3, XN1, YN2, ZN3, PX, PY, PZ, NPML, ModelFlag )
     !      !
     !      ! exchange boundary variables between all processor for F, G, H, Au1~,
     !      ! Au2~, Au3~
     !      CALL ExchangeInterfaceData( F1, F2, F3, F4, NA, XN1, YN2, ZN3, &
     !&              MYLEFT, MYRIGHT, MYFORWARD, MYREAR, MYUPPER, MYLOWER, &
     !&                 PX, PY, PZ, ModelFlag )
     !      !
     !      CALL ExchangeInterfaceData( G1, G2, G3, G4, NA, XN1, YN2, ZN3, &
     !&              MYLEFT, MYRIGHT, MYFORWARD, MYREAR, MYUPPER, MYLOWER, &
     !&                 PX, PY, PZ, ModelFlag )
     !      !
     !      IF ( ModelFlag == 1 ) THEN
     !         ! 3-D Model
```

```fortran
  !        CALL ExchangeInterfaceData( H1, H2, H3, H4, NA, XN1, YN2, ZN3, &
  !&                MYLEFT, MYRIGHT, MYFORWARD, MYREAR, MYUPPER, MYLOWER, &
  !&                   PX, PY, PZ, ModelFlag )
  !      END IF
        !
!       ! exchange boundary variables between all processors for K1, K2, K3, K4
    CALL ExchangeInterfaceDataNew( K1, NA, XN1, YN2, ZN3, NPML, XTYPE, YTYPE, &
  &           ZTYPE, MYLEFT, MYRIGHT, MYFORWARD, MYREAR, MYUPPER, MYLOWER, &
  &              PX, PY, PZ, NPX, NPY, NPZ, ModelFlag, BCFlag )
    !
    CALL ExchangeInterfaceDataNew( K2, NA, XN1, YN2, ZN3, NPML, XTYPE, YTYPE, &
  &           ZTYPE, MYLEFT, MYRIGHT, MYFORWARD, MYREAR, MYUPPER, MYLOWER, &
  &              PX, PY, PZ, NPX, NPY, NPZ, ModelFlag, BCFlag )
    !
    CALL ExchangeInterfaceDataNew( K3, NA, XN1, YN2, ZN3, NPML, XTYPE, YTYPE, &
  &           ZTYPE, MYLEFT, MYRIGHT, MYFORWARD, MYREAR, MYUPPER, MYLOWER, &
  &              PX, PY, PZ, NPX, NPY, NPZ, ModelFlag, BCFlag )
    !
    CALL ExchangeInterfaceDataNew( K4, NA, XN1, YN2, ZN3, NPML, XTYPE, YTYPE, &
  &           ZTYPE, MYLEFT, MYRIGHT, MYFORWARD, MYREAR, MYUPPER, MYLOWER, &
  &              PX, PY, PZ, NPX, NPY, NPZ, ModelFlag, BCFlag )
    !
    ! convert K1, K2, K3, K4 into F~, G~, H~
    CALL ConvertQtoFGH( K1, K2, K3, K4, F1, F2, F3, F4, G1, G2, G3, &
  &           G4, H1, H2, H3, H4, U0, V0, W0, P0, ROU0, C0, Jacobi, KexiX, &
  &           EitaY, TaoZ, NA, N1, N2, N3, XN1, YN2, ZN3, PX, PY, PZ, NPX, &
  &           NPY, NPZ, NPML, ModelFlag )
    !
    ! get the PML source terms for all governing equations
    IF ( I0 < LDDRK46 + 1 ) THEN
       CALL PMLSource( S1, S2, S3, S4, SAu11, SAu12, SAu13, SAu14, SAu21, SAu22, &
  &           SAu23, SAu24, SAu31, SAu32, SAu33, SAu34, F1, F2, F3, F4, Au11, &
  &           Au12, Au13, Au14, Au21, Au22, Au23, Au24, Au31, Au32, Au33, Au34, &
  &           NA, XN1, YN2, ZN3, Jacobi, KexiX, EitaY, TaoZ, MeshX, MeshY, MeshZ, &
  &           Beita, SigmaX1, SigmaX2, SigmaY1, SigmaY2, SigmaZ1, SigmaZ2, N1, N2, &
  &           N3, PX, PY, PZ, NPX, NPY, NPZ, NPML, T, C0, ModelFlag )
    END IF
  END DO
END SUBROUTINE LDDRK
!!
SUBROUTINE ConvertQtoFGH( Q1, Q2, Q3, Q4, F1, F2, F3, F4, G1, G2, G3, G4, &
  &           H1, H2, H3, H4, U0, V0, W0, P0, ROU0, C0, Jacobi, KexiX, EitaY, &
  &              TaoZ, NA, N1, N2, N3, XN1, YN2, ZN3, PX, PY, PZ, NPX, NPY, &
  &                 NPZ, NPML, ModelFlag )
  !-------------------------introduction---------------------------------------------
  !This program can get the flux value ( That is F1, ... , G1, ... ) from conservative
  !variables ( That is Q1, Q2, Q3, Q4 ). And the transformation equation can refer to
  !< Acoustic Scattering in Non-uniform flow > equation (11) derived by ChengLong.
  !--------------------------------------------------------------------------------------
  IMPLICIT NONE
  INTEGER:: I, J, K, KFlag
  INTEGER, INTENT(IN):: ModelFlag
  INTEGER, INTENT(IN):: NPML
```

```
INTEGER, INTENT(IN):: NA, XN1, YN2, ZN3
INTEGER, INTENT(IN):: N1, N2, N3
INTEGER, INTENT(IN):: PX, PY, PZ
INTEGER, INTENT(IN):: NPX, NPY, NPZ
REAL(KIND=8), INTENT(IN):: Q1( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(IN):: Q2( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(IN):: Q3( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(IN):: Q4( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(IN):: U0( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(IN):: V0( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(IN):: W0( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(IN):: P0( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(IN):: ROU0( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(IN):: C0( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(IN):: Jacobi( -(NPML-1):(N1+NPML),-(NPML-1):(N2+NPML),-(NPML-1):(N...
REAL(KIND=8), INTENT(IN):: KexiX( -(NPML-1) : (N1 + NPML) )
REAL(KIND=8), INTENT(IN):: EitaY( -(NPML-1) : (N2 + NPML) )
REAL(KIND=8), INTENT(IN):: TaoZ( -(NPML-1) : (N3 + NPML) )
REAL(KIND=8), INTENT(OUT):: F1( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(OUT):: F2( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(OUT):: F3( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(OUT):: F4( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(OUT):: G1( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(OUT):: G2( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(OUT):: G3( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(OUT):: G4( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(OUT):: H1( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(OUT):: H2( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(OUT):: H3( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(OUT):: H4( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
INTEGER:: I0, J0, K0
INTEGER:: SIZE0(2), SIZE1(2), SIZE2(2)
!!
I0 = XN1 * PX
J0 = YN2 * PY
K0 = ZN3 * PZ
KFlag = ModelFlag * ( K0 - NPML )
!
SIZE0(1) = - ( NA - 1 )
SIZE0(2) = XN1 + NA
SIZE1(1) = - ( NA - 1 )
SIZE1(2) = YN2 + NA
SIZE2(1) = - ( NA - 1 )
SIZE2(2) = ZN3 + NA
!
IF ( ModelFlag == 0 ) THEN
  ! 2-D model
  SIZE2 = 1
END IF
!
IF ( PX == 0 ) THEN
  SIZE0(1) = 1
ELSEIF( PX == NPX - 1 ) THEN
```

```
      SIZE0(2) = XN1
    END IF
    !
    IF ( PY == 0 ) THEN
      SIZE1(1) = 1
    ELSEIF( PY == NPY - 1 ) THEN
      SIZE1(2) = YN2
    END IF
    !
    IF ( PZ == 0 ) THEN
      SIZE2(1) = 1
    ELSEIF( PZ == NPZ - 1 ) THEN
      SIZE2(2) = ZN3
    END IF
    !
    DO K = SIZE2(1), SIZE2(2)
      DO J = SIZE1(1), SIZE1(2)
        DO I = SIZE0(1), SIZE0(2)
          !x-direction flux: F
          F1( I, J, K ) = KexiX( I0+I-NPML ) * Q2( I, J, K )
          F2( I, J, K ) = KexiX( I0+I-NPML ) * ( 2.0d0 * U0( I, J, K ) * &
    &       Q2( I, J, K ) - U0( I, J, K )**2 * Q1( I, J, K ) + &
    &         Q1( I, J, K ) * C0( I, J, K )**2  )
          F3( I, J, K ) = KexiX( I0+I-NPML ) * ( V0( I, J, K ) * Q2( I, J, K ) + &
    &       U0( I, J, K ) * Q3( I, J, K ) - U0( I, J, K ) * V0( I, J, K ) * &
    &         Q1( I, J, K ) )
          F4( I, J, K ) = KexiX( I0+I-NPML ) * ( W0( I, J, K ) * Q2( I, J, K ) + &
    &       U0( I, J, K ) * Q4( I, J, K ) - W0( I, J, K ) * U0( I, J, K ) * &
    &         Q1( I, J, K ) )
          !y-direction flux: G
          G1( I, J, K ) = EitaY( J0+J-NPML ) * Q3( I, J, K )
          G2( I, J, K ) = EitaY( J0+J-NPML ) * ( U0( I, J, K ) * Q3( I, J, K ) + &
    &       V0( I, J, K ) * Q2( I, J, K ) - U0( I, J, K ) * V0( I, J, K ) * &
    &         Q1( I, J, K ) )
          G3( I, J, K ) = EitaY( J0+J-NPML ) * ( 2.0d0 * V0( I, J, K ) * Q3( I, J, K ) - &
    &       V0( I, J, K )**2 * Q1( I, J, K ) + Q1( I, J, K ) * C0( I, J, K )**2 )
          G4( I, J, K ) = EitaY( J0+J-NPML ) * ( W0( I, J, K ) * Q3( I, J, K ) - &
    &       V0( I, J, K ) * Q4( I, J, K ) - W0( I, J, K ) * V0( I, J, K ) * &
    &         Q1( I, J, K ) )
          !z-direction flux: H
          H1( I, J, K ) = TaoZ( K+KFlag ) * Q4( I, J, K )
          H2( I, J, K ) = TaoZ( K+KFlag ) * ( U0( I, J, K ) * Q4( I, J, K ) + &
    &       W0( I, J, K ) * Q2( I, J, K ) - U0( I, J, K ) * W0( I, J, K ) * &
    &         Q1( I, J, K ) )
          H3( I, J, K ) = TaoZ( K+KFlag ) * ( V0( I, J, K ) * Q4( I, J, K ) + &
    &       W0( I, J, K ) * Q3( I, J, K ) -V0( I, J, K ) * W0( I, J, K ) * &
    &         Q1( I, J, K ) )
          H4( I, J, K ) = TaoZ( K+KFlag ) * ( 2.0d0 * W0( I, J, K ) * Q4( I, J, K ) - &
    &       W0( I, J, K )**2 * Q1( I, J, K ) + Q1( I, J, K ) * C0( I, J, K )**2 )
        END DO
      END DO
    END DO
  END SUBROUTINE ConvertQtoFGH
```

```fortran
! --------------------------------
! >>>>>>>>>>>>>>>>>>>.
SUBROUTINE ExchangeInterfaceDataNew( Q, NA, XN1, YN2, ZN3, NPML, XTYPE, YTYPE, &
  &         ZTYPE, MYLEFT, MYRIGHT, MYFORWARD, MYREAR, MYUPPER, MYLOWER, &
  &           PX, PY, PZ, NPX, NPY, NPZ, ModelFlag, BCFlag )
  USE MPI
  IMPLICIT NONE
  INTEGER, INTENT(IN):: NA, XN1, YN2, ZN3, NPML
  INTEGER, INTENT(IN):: XTYPE, YTYPE, ZTYPE
  INTEGER, INTENT(IN):: MYLEFT, MYRIGHT, MYFORWARD, MYREAR, MYUPPER, MYLOWER
  INTEGER, INTENT(IN):: PX, PY, PZ, NPX, NPY, NPZ
  INTEGER, INTENT(IN):: BCFlag
  INTEGER, INTENT(IN):: ModelFlag
  REAL(KIND=8), INTENT(INOUT):: Q( -(NA-1):(XN1+NA),-(NA-1):(YN2+NA),-(NA-1):(ZN3+NA) )
  !
  INTEGER:: IERR, TAG1
  INTEGER:: REQ(12), STATUS( MPI_STATUS_SIZE, 12 )
  !
  TAG1 = 100
  ! >>>>>>>>>>>>>>>>sending interferece data
  ! 1- sending forward and rearward face
  IF ( BCFlag == 1 ) THEN
    ! periodic boundary condition: exchange forward and rearward
    !   boundary information
    IF ( NPY == 1 ) THEN
      CALL MPI_ISEND( Q( 1,1+NPML+1,1 ), 1, XTYPE, MYFORWARD, TAG1, &
  &           MPI_COMM_WORLD, REQ(1), IERR )
      !
      CALL MPI_ISEND( Q( 1,YN2-NPML-NA,1 ), 1, XTYPE, MYREAR, TAG1, &
  &           MPI_COMM_WORLD, REQ(2), IERR )
    ELSE
      IF ( PY == 0 ) THEN
        ! forward face
        CALL MPI_ISEND( Q( 1,1+NPML+1,1 ), 1, XTYPE, MYFORWARD, TAG1, &
  &           MPI_COMM_WORLD, REQ(1), IERR )
        !
        CALL MPI_ISEND( Q( 1,YN2-NA+1,1 ), 1, XTYPE, MYREAR, TAG1, &
  &         MPI_COMM_WORLD, REQ(2), IERR )
      ELSEIF( PY == NPY - 1 ) THEN
        ! rearward face
        CALL MPI_ISEND( Q( 1,1,1 ), 1, XTYPE, MYFORWARD, TAG1, &
  &         MPI_COMM_WORLD, REQ(1), IERR )
        !
        CALL MPI_ISEND( Q( 1,YN2-NPML-NA,1 ), 1, XTYPE, MYREAR, TAG1, &
  &           MPI_COMM_WORLD, REQ(2), IERR )
      ELSE
        ! general part among the face
        CALL MPI_ISEND( Q( 1,1,1 ), 1, XTYPE, MYFORWARD, TAG1, &
  &           MPI_COMM_WORLD, REQ(1), IERR )
        !
        CALL MPI_ISEND( Q( 1,YN2-NA+1,1 ), 1, XTYPE, MYREAR, TAG1, &
  &           MPI_COMM_WORLD, REQ(2), IERR )
      END IF
```

```
      END IF
    ELSE
      ! free space boundary condition: No information exchanged on the
      !  forward and rearward boundary of y direction
      CALL MPI_ISEND( Q( 1,1,1 ), 1, XTYPE, MYFORWARD, TAG1, &
    &           MPI_COMM_WORLD, REQ(1), IERR )
      !
      CALL MPI_ISEND( Q( 1,YN2-NA+1,1 ), 1, XTYPE, MYREAR, TAG1, &
    &           MPI_COMM_WORLD, REQ(2), IERR )
    END IF
    !
    ! 2- sending left and right face
    CALL MPI_ISEND( Q( 1,1,1 ), 1, YTYPE, MYLEFT, TAG1, &
    &         MPI_COMM_WORLD, REQ(3), IERR )
    !
    CALL MPI_ISEND( Q( XN1-NA+1,1,1 ), 1, YTYPE, MYRIGHT, TAG1, &
    &         MPI_COMM_WORLD, REQ(4), IERR )
    !
    ! 3- sending lower and upper face
    CALL MPI_ISEND( Q( 1,1,1 ), 1, ZTYPE, MYLOWER, TAG1, &
    &         MPI_COMM_WORLD, REQ(5), IERR )
    !
    CALL MPI_ISEND( Q( 1,1,ZN3-NA+1 ), 1, ZTYPE, MYUPPER, TAG1, &
    &         MPI_COMM_WORLD, REQ(6), IERR )
    !
    ! <<<<<<<<<<<<<<<<<<<<<<<receiving interference data
    ! 1 - receiving forward and readward face
    IF ( BCFlag == 1 ) THEN
      ! periodic boundary condition: exchange forward and rearward
      !  boundary information
      IF ( NPY == 1 ) THEN
        CALL MPI_IRECV( Q( 1,NPML+1-NA,1 ), 1, XTYPE, MYFORWARD, TAG1, &
    &             MPI_COMM_WORLD, REQ(7), IERR )
        !
        CALL MPI_IRECV( Q( 1,YN2-NPML+1,1 ), 1, XTYPE, MYREAR, TAG1, &
    &             MPI_COMM_WORLD, REQ(8), IERR )
      ELSE
        IF ( PY == 0 ) THEN
          ! forward face
          CALL MPI_IRECV( Q( 1,NPML+1-NA,1 ), 1, XTYPE, MYFORWARD, TAG1, &
    &           MPI_COMM_WORLD, REQ(7), IERR )
          !
          CALL MPI_IRECV( Q( 1,YN2+1,1 ), 1, XTYPE, MYREAR, TAG1, &
    &           MPI_COMM_WORLD, REQ(8), IERR )
        ELSEIF( PY == NPY - 1 ) THEN
          ! rearward face
          CALL MPI_IRECV( Q( 1,1-NA,1 ), 1, XTYPE, MYFORWARD, TAG1, &
    &           MPI_COMM_WORLD, REQ(7), IERR )
          !
          CALL MPI_IRECV( Q( 1,YN2-NPML+1,1 ), 1, XTYPE, MYREAR, TAG1, &
    &           MPI_COMM_WORLD, REQ(8), IERR )
        ELSE
```

```
            ! general part among the face
            CALL MPI_IRECV( Q( 1,1-NA,1 ), 1, XTYPE, MYFORWARD, TAG1, &
     &             MPI_COMM_WORLD, REQ(7), IERR )
            !
            CALL MPI_IRECV( Q( 1,YN2+1,1 ), 1, XTYPE, MYREAR, TAG1, &
     &             MPI_COMM_WORLD, REQ(8), IERR )
          END IF
        END IF
      ELSE
        ! free space boundary condition: No information exchanged on the
        !   forward and rearward boundary of y direction
        CALL MPI_IRECV( Q( 1,1-NA,1 ), 1, XTYPE, MYFORWARD, TAG1, &
     &             MPI_COMM_WORLD, REQ(7), IERR )
        !
        CALL MPI_IRECV( Q( 1,YN2+1,1 ), 1, XTYPE, MYREAR, TAG1, &
     &             MPI_COMM_WORLD, REQ(8), IERR )
      END IF
      !
      ! 2- receiving left and right face
      CALL MPI_IRECV( Q( 1-NA,1,1 ), 1, YTYPE, MYLEFT, TAG1, &
     &           MPI_COMM_WORLD, REQ(9), IERR )
      !
      CALL MPI_IRECV( Q( XN1+1,1,1 ), 1, YTYPE, MYRIGHT, TAG1, &
     &           MPI_COMM_WORLD, REQ(10), IERR )
      !
      ! 3- receiving lower and upper face
      CALL MPI_IRECV( Q( 1,1,1-NA ), 1, ZTYPE, MYLOWER, TAG1, &
     &           MPI_COMM_WORLD, REQ(11), IERR )
      !
      CALL MPI_IRECV( Q( 1,1,ZN3+1 ), 1, ZTYPE, MYUPPER, TAG1, &
     &           MPI_COMM_WORLD, REQ(12), IERR )
      !
      CALL MPI_WAITALL( 12, REQ, STATUS, IERR )
    END SUBROUTINE ExchangeInterfaceDataNew
    ! ----------------------------------------
    ! >>>>>>>>>>>>>>.
    SUBROUTINE GetOriginalVariables( U, V, W, P, ROU, U0, V0, W0, P0, ROU0, C0, &
     &             Q1, Q2, Q3, Q4, NA, XN1, YN2, ZN3, Jacobi, KexiX, EitaY, &
     &               TaoZ, N1, N2, N3, PX, PY, PZ, NPML, ModelFlag )
      IMPLICIT NONE
      INTEGER:: I, J, K, I0, J0, K0, KFlag
      INTEGER, INTENT(IN):: ModelFlag
      INTEGER, INTENT(IN):: NA, XN1, YN2, ZN3
      INTEGER, INTENT(IN):: N1, N2, N3, PX, PY, PZ, NPML
      REAL(KIND=8), INTENT(IN):: Q1( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
      REAL(KIND=8), INTENT(IN):: Q2( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
      REAL(KIND=8), INTENT(IN):: Q3( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
      REAL(KIND=8), INTENT(IN):: Q4( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
      REAL(KIND=8), INTENT(IN):: Jacobi( -(NPML-1):(N1+NPML),-(NPML-1):(N2+NPML),-(NPML-1):(N…
      REAL(KIND=8), INTENT(IN):: KexiX( -(NPML-1) : (N1+NPML) )
      REAL(KIND=8), INTENT(IN):: EitaY( -(NPML-1) : (N2+NPML) )
      REAL(KIND=8), INTENT(IN):: TaoZ( -(NPML-1) : (N3+NPML) )
      REAL(KIND=8), INTENT(IN):: C0( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
```

```
REAL(KIND=8), INTENT(IN):: U0( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(IN):: V0( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(IN):: W0( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(IN):: P0( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(IN):: ROU0( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(OUT):: U( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(OUT):: V( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(OUT):: W( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(OUT):: P( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(OUT):: ROU( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+N...
INTEGER:: SIZE0(2)
REAL(KIND=8):: Jacobi1( -(NPML-1+NA) : (N1+NPML+NA), -(NPML-1+NA) : (N2+NPML+NA), &
&                                    -(NPML-1+NA) : (N3+NPML+NA) )
REAL(KIND=8):: KexiX1( - ( NPML - 1 + NA ) : ( N1 + NPML + NA ) )
REAL(KIND=8):: EitaY1( - ( NPML - 1 + NA ) : ( N2 + NPML + NA ) )
REAL(KIND=8):: TaoZ1( - ( NPML - 1 + NA ) : ( N3 + NPML + NA ) )
!!
Jacobi1 = 1.0d0
KexiX1 = 1.0d0
EitaY1 = 1.0d0
TaoZ1 = 1.0d0
Jacobi1( -(NPML-1):(N1+NPML),-(NPML-1):(N2+NPML),-(NPML-1):(N3+NPML) ) = Jacobi( &
&   -(NPML-1):(N1+NPML),-(NPML-1):(N2+NPML),-(NPML-1):(N3+NPML) )
KexiX1( -(NPML-1) : (N1 + NPML) ) = KexiX( -(NPML-1) : (N1 + NPML) )
EitaY1( -(NPML-1) : (N2 + NPML) ) = EitaY( -(NPML-1) : (N2 + NPML) )
TaoZ1( -(NPML-1) : (N3 + NPML) ) = TaoZ( -(NPML-1) : (N3 + NPML) )
!!
I0 = XN1 * PX
J0 = YN2 * PY
K0 = ZN3 * PZ
IF ( ModelFlag == 0 ) THEN
   ! 2-D Model
   SIZE0 = ZN3
ELSE
   ! 3-D Model
   SIZE0(1) = - ( NA - 1 )
   SIZE0(2) = ZN3 + NA
END IF
!
KFlag = ModelFlag * ( K0 - NPML )
!!
DO K = SIZE0(1), SIZE0(2)
   DO J = - ( NA - 1 ), ( YN2 + NA )
      DO I = - ( NA - 1 ), ( XN1 + NA )
         ROU( I, J, K ) = Q1( I, J, K ) / Jacobi1( I0+I-NPML, J0+J-NPML, K+KFlag )
         U( I, J, K ) = ( Q2( I, J, K ) - Q1( I, J, K ) * U0( I, J, K ) ) &
&              / ( Jacobi1( I0+I-NPML, J0+J-NPML, K+KFlag ) * ROU0( I, J, K ) )
         V( I, J, K ) = ( Q3( I, J, K ) - Q1( I, J, K ) * V0( I, J, K ) ) &
&              / ( Jacobi1( I0+I-NPML, J0+J-NPML, K+KFlag ) * ROU0( I, J, K ) )
         W( I, J, K ) = ( Q4( I, J, K ) - Q1( I, J, K ) * W0( I, J, K ) ) &
&              / ( Jacobi1( I0+I-NPML, J0+J-NPML, K+KFlag ) * ROU0( I, J, K ) )
         P( I, J, K ) = ROU( I, J, K ) * C0( I, J, K )**2
      END DO
```

```
      END DO
    END DO
  END SUBROUTINE GetOriginalVariables
  ! -----------------------------------------------
  ! >>>>>>>>>>>.
  SUBROUTINE GetEffectMatrix( AProcessor, PLN, EffectRange, &
    &         Shapes, Cell_S, NormalVector, LN, MeshX, MeshY, &
    &           MeshZ, NPML, N1, N2, N3, StartEnd, CutPoints, &
    &             DeltaXYZ, MYID, ModelFlag )
    IMPLICIT NONE
    INTEGER:: I, J, K, L, M
    INTEGER, INTENT(IN):: ModelFlag
    INTEGER, INTENT(IN):: NPML
    INTEGER, INTENT(IN):: PLN, LN, N1, N2, N3
    INTEGER, INTENT(IN):: MYID
    INTEGER, INTENT(IN):: StartEnd( 3, 3 )
    REAL(KIND=8), INTENT(IN):: DeltaXYZ
    REAL(KIND=8), INTENT(IN):: CutPoints( 3, 4 )
    REAL(KIND=8), INTENT(IN):: Shapes( LN, 3 ), Cell_S( LN )
    REAL(KIND=8), INTENT(IN):: NormalVector( LN, 3 )
    REAL(KIND=8), INTENT(IN):: MeshX( -(NPML-1) : (N1+NPML) )
    REAL(KIND=8), INTENT(IN):: MeshY( -(NPML-1) : (N2+NPML) )
    REAL(KIND=8), INTENT(IN):: MeshZ( -(NPML-1) : (N3+NPML) )
    REAL(KIND=8), INTENT(OUT):: AProcessor( LN, PLN )
    INTEGER, INTENT(OUT):: EffectRange( LN, 6 )
    INTEGER:: I0, FlagXYZ( 3, 4 ), FlagXYZ1( 3, 2 )
    INTEGER:: Flag
    REAL(KIND=8):: TempMesh( 1, 3 )
    REAL(KIND=8), EXTERNAL:: Distribution
    !!
    AProcessor = 0.0d0
    !determine the maximum lagrange point
    IF ( ( MYID + 1 ) * PLN <= LN ) THEN
      I0 = PLN
    ELSE
      I0 = LN - MYID * PLN
    END IF
    !!
    !get the effect matrix AProcessor for processor MYID
    DO J = 1, 3
      DO I = 1, LN
        EffectRange( I, 2*J-1 ) = StartEnd( J, 1 ) + FLOOR( ( Shapes( I , J ) &
    &         - CutPoints( J, 2 ) ) / DeltaXYZ ) - 2
        EffectRange( I, 2*J ) = EffectRange( I, 2*J-1 ) + 5
      END DO
    END DO
    IF ( ( ModelFlag == 0 ) ) THEN
      ! 2-D Model
      J = 3
      EffectRange( :, 2*J-1 ) = 1
      EffectRange( :, 2*J ) = 1
    END IF
  !   OPEN( UNIT=10, FILE='EffectRange.dat' )
```

```
!   DO I = 1, LN
!      WRITE( UNIT=10, FMT=* )EffectRange( I, : )
!   END DO
!   CLOSE( UNIT=10 )
    !!
    IF ( ModelFlag == 0 ) THEN
      ! 2- D model
      Flag = 2
    ELSE
      Flag = 3
    END IF
    !
    DO L = 1, LN
      !!
      DO M = 1, I0
        !!
        !get the minimum interpolation range
        DO I = 1, 3
          FlagXYZ( I, 1 ) = EffectRange( L, 2*I-1 )
          FlagXYZ( I, 3 ) = EffectRange( L, 2*I )
          FlagXYZ( I, 2 ) = EffectRange( M + MYID * PLN, 2*I-1 )
          FlagXYZ( I, 4 ) = EffectRange( M + MYID * PLN, 2*I )
        END DO
        !!
        DO J = 1, 3
          FlagXYZ1( J, 1 ) = MAXVAL( FlagXYZ( J, 1 : 2 ) )
          FlagXYZ1( J, 2 ) = MINVAL( FlagXYZ( J, 3 : 4 ) )
        END DO
        !!
        !get the element of matrix AProcessor
        DO K = FlagXYZ1( 3, 1 ), FlagXYZ1( 3, 2 )
          DO J = FlagXYZ1( 2, 1 ), FlagXYZ1( 2, 2 )
            DO I = FlagXYZ1( 1, 1 ), FlagXYZ1( 1, 2 )
              TempMesh( 1, 1 ) = MeshX( I )
              TempMesh( 1, 2 ) = MeshY( J )
              TempMesh( 1, 3 ) = MeshZ( K )
              AProcessor( L, M ) = AProcessor( L, M ) + &
    &           ( NormalVector( L, 1 ) * NormalVector( M + MYID * PLN, 1 ) + &
    &           NormalVector( L , 2 ) * NormalVector( M + MYID * PLN, 2 ) + &
    &           NormalVector( L , 3 ) * NormalVector( M + MYID * PLN, 3 ) ) * &
    &           Cell_S( M + MYID * PLN ) * DeltaXYZ**Flag * &
    &           Distribution( Shapes( L , : ), TempMesh, DeltaXYZ, ModelFlag ) &
    &            * Distribution( Shapes( M + MYID * PLN, : ), &
    &            TempMesh, DeltaXYZ, ModelFlag )
            END DO
          END DO
        END DO
      END DO
    END DO
  END SUBROUTINE GetEffectMatrix
  ! ----------------------------------------
  ! >>>>>>>>>>>>>>>>>....
  SUBROUTINE GetDuDt( DuDt, DuDtFlag, EffectRange, Shapes, NormalVector, LN, U, V, W, &
```

```
&            NA, XN1, YN2, ZN3, MeshX, MeshY, MeshZ, NPML, N1, N2, N3, &
&              DeltaXYZ, DeltaT, PX, PY, PZ, NPX, NPY, NPZ, MYID, ModelFlag )
IMPLICIT NONE
INTEGER:: I, J, K, L, I0
INTEGER, INTENT(IN):: ModelFlag
INTEGER, INTENT(IN):: NPML, N1, N2, N3
INTEGER, INTENT(IN):: LN
INTEGER, INTENT(IN):: NA, XN1, YN2, ZN3
INTEGER, INTENT(IN):: MYID, PX, PY, PZ, NPX, NPY, NPZ
INTEGER, INTENT(IN):: EffectRange( LN, 6 )
REAL(KIND=8), INTENT(IN):: DeltaT, DeltaXYZ
REAL(KIND=8), INTENT(IN):: Shapes( LN, 3 ), NormalVector( LN, 3 )
REAL(KIND=8), INTENT(IN):: U( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(IN):: V( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(IN):: W( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
REAL(KIND=8), INTENT(IN):: MeshX( -(NPML-1) : (N1+NPML) )
REAL(KIND=8), INTENT(IN):: MeshY( -(NPML-1) : (N2+NPML) )
REAL(KIND=8), INTENT(IN):: MeshZ( -(NPML-1) : (N3+NPML) )
REAL(KIND=8), INTENT(OUT):: DuDt( LN )
INTEGER, INTENT(OUT):: DuDtFlag( LN+1 )
REAL(KIND=8):: Un( LN ), U1, V1, W1, TempMesh( 1, 3 )
REAL(KIND=8):: A(2), B(2), C(2)
INTEGER:: I1(2), J1(2), K1(2)
INTEGER:: X0, Y0, Z0
REAL(KIND=8):: Flag
INTEGER:: Flag0
REAL(KIND=8), EXTERNAL:: Distribution
!!
X0 = PX * XN1
Y0 = PY * YN2
Z0 = PZ * ZN3
IF ( PX == NPX - 1 ) THEN
   A(1) = MeshX( X0 + 1 - NPML )
   A(2) = MeshX( X0 + XN1 - NPML )
ELSE
   A(1) = MeshX( X0 + 1 - NPML )
   A(2) = MeshX( X0 + XN1 - NPML + 1 )
END IF
!
IF ( PY == NPY - 1 ) THEN
   B(1) = MeshY( Y0 + 1 - NPML )
   B(2) = MeshY( Y0 + YN2 - NPML )
ELSE
   B(1) = MeshY( Y0 + 1 - NPML )
   B(2) = MeshY( Y0 + YN2 - NPML + 1  )
END IF
!
IF ( ModelFlag == 0 ) THEN
   ! 2-D Model
   C(1) = MeshZ( ZN3 )
   C(2) = MeshZ( ZN3 )
ELSE
   IF ( PZ == NPZ - 1 ) THEN
```

```
        C(1) = MeshZ( Z0 + 1 - NPML )
        C(2) = MeshZ( Z0 + ZN3 - NPML )
      ELSE
        C(1) = MeshZ( Z0 + 1 - NPML )
        C(2) = MeshZ( Z0 + ZN3 - NPML + 1 )
      END IF
    END IF
    !
    Flag0 = 2
    DO L = 1, LN
      IF ( ( Shapes(L,1) >= A(1) ) ) THEN
      IF ( ( Shapes(L,1) < A(2) ) .OR. ( ( Shapes(L,1) == A(2) ) &
    &                  .AND. ( PX == (NPX-1) ) ) ) THEN
        IF ( ( Shapes(L,2) >= B(1) ) ) THEN
        IF ( ( Shapes(L,2) < B(2) ) .OR. ( ( Shapes(L,2) == B(2) ) &
    &                  .AND. PY == ( NPY-1 ) ) ) THEN
          IF ( ( Shapes(L,3) >= C(1) ) ) THEN
          IF ( ( Shapes(L,3) < C(2) ) .OR. ( ( Shapes(L,3) == C(2) ) &
    &                  .AND. PZ == ( NPZ-1 ) ) ) THEN
            ! Lagrange Point L is in the processor MYID
            DuDtFlag( Flag0 ) = L
            !
            ! get the interpolation range
            K1(1) = EffectRange( L, 5 ) - Z0 + NPML
            K1(2) = EffectRange( L, 6 ) - Z0 + NPML
            J1(1) = EffectRange( L, 3 ) - Y0 + NPML
            J1(2) = EffectRange( L, 4 ) - Y0 + NPML
            I1(1) = EffectRange( L, 1 ) - X0 + NPML
            I1(2) = EffectRange( L, 2 ) - X0 + NPML
            !
            ! get the velocity for lagrange points I
            U1 = 0.0d0
            V1 = 0.0d0
            W1 = 0.0d0
            !
            IF ( ModelFlag == 0 ) THEN
            ! 2- D model
              K1 = ZN3
              Flag = 2
            ELSE
              Flag = 3
            END IF
            !
            DO k = K1(1), K1(2), 1
              DO J = J1(1), J1(2), 1
                DO I = I1(1), I1(2), 1
                  TempMesh( 1, 1 ) = MeshX( I + X0 - NPML )
                  TempMesh( 1, 2 ) = MeshY( J + Y0 - NPML )
                  TempMesh( 1, 3 ) = MeshZ( K + ModelFlag * ( Z0-NPML ) )
                  U1 = U1+Distribution( Shapes( L, : ), TempMesh, DeltaXYZ, &
    &                     ModelFlag ) * U( I, J, K ) * DeltaXYZ**Flag
                  V1 = V1+Distribution( Shapes( L, : ), TempMesh, DeltaXYZ, &
    &                     ModelFlag ) * V( I, J, K ) * DeltaXYZ**Flag
```

```fortran
                              W1 = W1+Distribution( Shapes( L, : ), TempMesh, DeltaXYZ, &
     &                               ModelFlag ) * W( I, J, K ) * DeltaXYZ**Flag
                            END DO
                         END DO
                      END DO
                      !
                      Un( Flag0-1 ) = NormalVector( L, 1 ) * U1 + NormalVector( L, 2 ) * V1 &
     &                                             + NormalVector( L, 3 ) * W1
                      DuDt( Flag0-1 ) = - Un( Flag0 - 1 ) / DeltaT
                      Flag0 = Flag0 + 1
                   END IF
                   END IF
                END IF
                END IF
             END IF
             END IF
          END DO
          DuDtFlag(1) = Flag0 - 2
       END SUBROUTINE GetDuDt
       ! -------------------------------------
       ! >>>>>>>>>>>>>
       SUBROUTINE ForcingSolver( Du, Dv, Dw, AV, AT, DuDt, MeshX, MeshY, MeshZ, Shapes, &
          &          NormalVector, DeltaXYZ, DeltaT, Cell_S, StartEnd, LN, NPML, &
          &             N1, N2, N3, XN1, YN2, ZN3, PX, PY, PZ, Loops, ModelFlag )
          IMPLICIT NONE
          INTEGER:: I, J, K
          INTEGER, INTENT(IN):: Loops
          INTEGER, INTENT(IN):: ModelFlag
          INTEGER, INTENT(IN):: LN
          INTEGER, INTENT(IN):: NPML, N1, N2, N3
          INTEGER, INTENT(IN):: XN1, YN2, ZN3
          INTEGER, INTENT(IN):: PX, PY, PZ
          INTEGER, INTENT(IN):: StartEnd( 3, 3 )
          REAL(KIND=8), INTENT(IN):: Shapes( LN, 3 )
          REAL(KIND=8), INTENT(IN):: DeltaXYZ, DeltaT, Cell_S( LN )
          REAL(KIND=8), INTENT(IN):: NormalVector( LN,3 )
          REAL(KIND=8), INTENT(IN):: MeshX( -(NPML-1) : (N1+NPML) )
          REAL(KIND=8), INTENT(IN):: MeshY( -(NPML-1) : (N2+NPML) )
          REAL(KIND=8), INTENT(IN):: MeshZ( -(NPML-1) : (N3+NPML) )
          REAL(KIND=8), INTENT(IN):: AV( LN, LN )
          REAL(KIND=8), INTENT(IN):: DuDt( LN )
          REAL(KIND=8), INTENT(OUT):: Du( 1:XN1, 1:YN2, 1:ZN3 )
          REAL(KIND=8), INTENT(OUT):: Dv( 1:XN1, 1:YN2, 1:ZN3 )
          REAL(KIND=8), INTENT(OUT):: Dw( 1:XN1, 1:YN2, 1:ZN3 )
          REAL(KIND=8), INTENT(INOUT):: AT( LN, LN )
          REAL(KIND=8):: Fn( LN )
          INTEGER:: LOOPS1(2), LOOPS2(2), LOOPS3(2)
          REAL(KIND=8):: SumFx, SumFy ,SumFz
          REAL(KIND=8):: points( 1, 3 ), Coordinate( 1, 3 )
          INTEGER:: INFO,RANK,LWORK
          REAL(KIND=8):: B( LN, 1 )
          REAL(KIND=8):: RCOND
          REAL(KIND=8), ALLOCATABLE:: S(:), WORK(:)
```

```
INTEGER:: I0, J0, K0, I1, J1, K1, K2
REAL(KIND=8):: Residual
INTEGER:: MaxIteration
!
INTEGER:: IPIV( LN )
INTEGER:: NRHS
REAL(KIND=8), EXTERNAL:: Distribution
!
INTEGER:: X0(2), Y0(2), Z0(2)
REAL(KIND=8):: StartTime, EndTime
!
Du = 0.0d0
Dv = 0.0d0
Dw = 0.0d0
!
! start position
I1 = PX * XN1
J1 = PY * YN2
K1 = PZ * ZN3
!
!LWORK = 64 * LN
!ALLOCATE( S(LN), WORK(LWORK) )
!RCOND = 0.01d0
B( 1 : LN, 1 ) = DuDt( 1 : LN )
!NRHS = 1
!
!CALL CPU_TIME( StartTime )
! solve linear equations using SVD
!CALL DGELSS( LN, LN, 1, AT, LN, B, LN, S, RCOND, RANK, WORK, LWORK, INFO )
!
Residual = 1.0d-6
MaxIteration = 100000
!CALL SOR( AV, B, LN, Residual, MaxIteration )
IF ( Loops == 1 ) THEN
   AT = AV
   CALL LU(AT,LN)
END IF
CALL SOLVE(AT,B,LN)
!CALL CPU_TIME( EndTime )
!WRITE( *, * )EndTime-StartTime
!
! algorithm 2: using linear sovler
! LU decompsition
! CALL DGETRF( LN, LN, AT, LN, IPIV, INFO )
! solve
! CALL DGETRS( 'No transpose', LN, NRHS, AT, LN, IPIV, B, LN, INFO )
!
Fn( 1 : LN ) = B( 1 : LN, 1 )
!
LOOPS3( 1 ) = FLOOR( ( MINVAL( Shapes( :, 3 ) ) - MeshZ( StartEnd( 3, 1 ) ) )/DeltaXYZ ) - 2
LOOPS3( 2 ) = CEILING( ( MAXVAL( Shapes( :, 3 ) ) - MeshZ( StartEnd( 3, 1 ) ) )/DeltaXYZ ) + 2
LOOPS3( 1 ) = LOOPS3( 1 ) + StartEnd( 3, 1 )
```

```
LOOPS3( 2 ) = LOOPS3( 2 ) + StartEnd( 3, 1 )
LOOPS2( 1 ) = FLOOR( ( MINVAL( Shapes( :, 2 ) ) - MeshY( StartEnd( 2, 1 ) ) )/DeltaXYZ ) - 2
LOOPS2( 2 ) = CEILING( ( MAXVAL( Shapes( :, 2 ) ) - MeshY( StartEnd( 2, 1 ) ) )/DeltaXYZ ) + 2
LOOPS2( 1 ) = LOOPS2( 1 ) + StartEnd( 2, 1 )
LOOPS2( 2 ) = LOOPS2( 2 ) + StartEnd( 2, 1 )
LOOPS1( 1 ) = FLOOR( ( MINVAL( Shapes( :, 1 ) ) - MeshX( StartEnd( 1, 1 ) ) )/DeltaXYZ ) - 2
LOOPS1( 2 ) = CEILING( ( MAXVAL( Shapes( :, 1 ) ) - MeshX( StartEnd( 1, 1 ) ) )/DeltaXYZ ) + 2
LOOPS1( 1 ) = LOOPS1( 1 ) + StartEnd( 1, 1 )
LOOPS1( 2 ) = LOOPS1( 2 ) + StartEnd( 1, 1 )
!
! Z-direction
IF ( ModelFlag == 0 ) THEN
  ! 2-D model
  Z0 = ZN3
ELSE
  ! 3-D Model
  IF ( ( LOOPS3( 1 ) + NPML - K1 ) .GE. 1 ) THEN
    Z0(1) = LOOPS3(1) + NPML - K1
  ELSE
    Z0(1) = 1
  END IF
  IF ( ( LOOPS3( 2 ) + NPML - K1 ) .LE. ZN3 ) THEN
    Z0(2) = LOOPS3(2) + NPML - K1
  ELSE
    Z0(2) = ZN3
  END IF
END IF
!
! Y- direction
IF ( ( LOOPS2( 1 ) + NPML - J1 ) .GE. 1 ) THEN
  Y0(1) = LOOPS2(1) + NPML - J1
ELSE
  Y0(1) = 1
END IF
IF ( ( LOOPS2( 2 ) + NPML - J1 ) .LE. YN2 ) THEN
  Y0(2) = LOOPS2(2) + NPML - J1
ELSE
  Y0(2) = YN2
END IF
!
!X -direction
IF ( ( LOOPS1( 1 ) + NPML - I1 ) .GE. 1 ) THEN
  X0(1) = LOOPS1(1) + NPML - I1
ELSE
  X0(1) = 1
END IF
IF ( ( LOOPS1( 2 ) + NPML - I1 ) .LE. XN1 ) THEN
  X0(2) = LOOPS1(2) + NPML - I1
ELSE
  X0(2) = XN1
END IF
!
K2 = ModelFlag * ( K1 - NPML )
```

```fortran
    DO K = Z0(1), Z0(2)
      DO J = Y0(1), Y0(2)
        DO I = X0(1), X0(2)
          SumFx = 0.0d0
          SumFy = 0.0d0
          SumFz = 0.0d0
          points( 1, 1 ) = MeshX( I + I1 - NPML )
          points( 1, 2 ) = MeshY( J + J1 - NPML )
          points( 1, 3 ) = MeshZ( k + K2 )
          DO K0 = 1, LN
            coordinate( 1, 1 ) = Shapes( K0, 1 )
            coordinate( 1, 2 ) = Shapes( K0, 2 )
            coordinate( 1, 3 ) = Shapes( K0, 3 )
            SumFx = SumFx + Fn( K0 ) * NormalVector( K0, 1 ) * Cell_S(K0) * &
    &           Distribution( coordinate, points, DeltaXYZ, ModelFlag )
            SumFy = SumFy + Fn( K0 ) * NormalVector( K0, 2 ) * Cell_S(K0) * &
    &           Distribution( coordinate, points, DeltaXYZ, ModelFlag )
            SumFz = SumFz + Fn( K0 ) * NormalVector( K0, 3 ) * Cell_S(K0) * &
    &           Distribution( coordinate, points, DeltaXYZ, ModelFlag )
          END DO
          Du( I, J, k ) = SumFx * DeltaT
          Dv( I, J, k ) = SumFy * DeltaT
          Dw( I, J, k ) = SumFz * DeltaT
        END DO
      END DO
    END DO
  END SUBROUTINE ForcingSolver
  ! ----------------------------------------
  ! >>>>>>>>>>>>>>>...
  SUBROUTINE CorrectAcousticField( U, V, W, Du, Dv, Dw, MeshX, MeshY, MeshZ, &
    &         NA, XN1, YN2, ZN3, NPML, N1, N2, N3, PX, PY, PZ, &
    &           EffectRange, LN, ModelFlag )
    IMPLICIT NONE
    INTEGER:: I, J, K
    INTEGER, INTENT(IN):: ModelFlag
    INTEGER, INTENT(IN):: LN
    INTEGER, INTENT(IN):: NPML
    INTEGER, INTENT(IN):: XN1, YN2, ZN3, NA, N1, N2, N3
    INTEGER, INTENT(IN):: PX, PY, PZ
    INTEGER, INTENT(IN):: EffectRange( LN, 6 )
    REAL(KIND=8), INTENT(IN):: Du( 1:XN1, 1:YN2, 1:ZN3 )
    REAL(KIND=8), INTENT(IN):: Dv( 1:XN1, 1:YN2, 1:ZN3 )
    REAL(KIND=8), INTENT(IN):: Dw( 1:XN1, 1:YN2, 1:ZN3 )
    REAL(KIND=8), INTENT(IN):: MeshX( -(NPML-1) : (N1+NPML) )
    REAL(KIND=8), INTENT(IN):: MeshY( -(NPML-1) : (N2+NPML) )
    REAL(KIND=8), INTENT(IN):: MeshZ( -(NPML-1) : (N3+NPML) )
    REAL(KIND=8), INTENT(INOUT):: U( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
    REAL(KIND=8), INTENT(INOUT):: V( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+NA) )
    REAL(KIND=8), INTENT(INOUT):: W( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+N...
    INTEGER:: I0(2), J0(2), K0(2), I1(4), J1(4), K1(4)
    INTEGER:: IFlag, JFlag, KFlag, XFlag(2), YFlag(2), ZFlag(2)
    !!
    IFlag = PX * XN1
```

```
    JFlag = PY * YN2
    KFlag = PZ * ZN3
    !
    I0(1) = MINVAL( EffectRange( :, 1 ) )
    I0(2) = MAXVAL( EffectRange( :, 2 ) )
    J0(1) = MINVAL( EffectRange( :, 3 ) )
    J0(2) = MAXVAL( EffectRange( :, 4 ) )
    K0(1) = MINVAL( EffectRange( :, 5 ) )
    K0(2) = MAXVAL( EffectRange( :, 6 ) )
    !!
    I1(1) = I0(1)
    I1(2) = 1 + IFlag - NPML
    I1(3) = I0(2)
    I1(4) = ( PX + 1 ) * XN1 - NPML
    J1(1) = J0(1)
    J1(2) = 1 + JFlag -NPML
    J1(3) = J0(2)
    J1(4) = ( PY + 1 ) * YN2 - NPML
    K1(1) = K0(1)
    K1(2) = 1 + KFlag - NPML
    K1(3) = K0(2)
    K1(4) = ( PZ + 1 ) * ZN3 - NPML
    IF ( ModelFlag == 0 ) THEN
      ! 2-D Model
      K1 = ZN3
    END IF
    !!
    ZFlag(1) = MAXVAL( K1(1:2) ) - ModelFlag * ( KFlag - NPML )
    ZFlag(2) = MINVAL( K1(3:4) ) - ModelFlag * ( KFlag - NPML )
    YFlag(1) = MAXVAL( J1(1:2) ) - ( JFlag - NPML )
    YFlag(2) = MINVAL( J1(3:4) ) - ( JFlag - NPML )
    XFlag(1) = MAXVAL( I1(1:2) ) - ( IFlag - NPML )
    XFlag(2) = MINVAL( I1(3:4) ) - ( IFlag - NPML )
    DO K = ZFlag(1), ZFlag(2)
      DO J = YFlag(1), YFlag(2)
        DO I = XFlag(1), XFlag(2)
          U( I, J, K ) = U( I, J, K ) + Du( I, J, K )
          V( I, J, K ) = V( I, J, K ) + Dv( I, J, K )
          W( I, J, K ) = W( I, J, K ) + Dw( I, J, K )
        END DO
      END DO
    END DO
  END SUBROUTINE CorrectAcousticField
  ! --------------------------------------
  ! >>>>>>>>>>>>>>>>>>>.
  SUBROUTINE OutputToTextFile( TotalX, N1, N2, N3, K0, FLAG, Filename0 )
    ! ------------------introduction----------------------
    ! this subroutine can output the total flow field result
    !   of text form by using series I/0
    ! ----------------------------------------------------
    IMPLICIT NONE
    INTEGER:: I, J, K
    INTEGER, INTENT(IN):: FLAG, K0
```

```
    INTEGER, INTENT(IN):: N1, N2, N3
    REAL(KIND=8), INTENT(IN):: TotalX( 1:N1, 1:N2, 1:N3 )
    CHARACTER(LEN=80), INTENT(IN):: Filename0
    CHARACTER(LEN=80):: FILENAME, NAME0,STR1
    !
    IF ( FLAG == 1 ) THEN
      ! output x-direction velocity: U
      NAME0 = 'U'
    ELSEIF ( FLAG == 2 ) THEN
      ! output y-direction velocity: V
      NAME0 = 'V'
    ELSEIF ( FLAG == 3 ) THEN
      ! output Z-direction velocity: W
      NAME0 = 'W'
    ELSEIF ( FLAG == 4 ) THEN
      ! output pressure: P
      NAME0 = 'p'
    ELSEIF ( FLAG == 5 ) THEN
      ! output density: rou
      NAME0 = 'rou'
    ELSE
      ! output vorticity: omiga
      NAME0 = 'omiga'
    END IF
    IF ( K0 < 10 ) THEN
      WRITE( STR1, '(I1)' ) K0
    ELSEIF ( K0 < 100 ) THEN
      WRITE( STR1, '(I2)' ) K0
    ELSEIF ( K0 < 1000 ) THEN
      WRITE( STR1, '(I3)' ) K0
    ELSEIF ( K0 < 10000 ) THEN
      WRITE( STR1, '(I4)' ) K0
    END IF
    FILENAME = ( TRIM( Filename0 ) // TRIM( NAME0 ) // TRIM( STR1 ) // TRIM( '.dat' ) )
    !
    OPEN( UNIT=10, FILE = TRIM( FILENAME ) )
    DO K = 1, N3
      DO I = 1, N1
        DO J = 1, N2
          IF ( J == N2 ) THEN
            WRITE( UNIT= 10, FMT = '(f20.14)', ADVANCE='YES' ) TotalX( I, J, K )
          ELSE
            WRITE( UNIT=10, FMT= '(f20.14)', ADVANCE = 'NO' ) TotalX( I, J, K )
          END IF
        END DO
      END DO
    END DO
    CLOSE( UNIT=10 )
  END SUBROUTINE OutputToTextFile
  ! ------------------------------
  ! >>>>>>>>>>>>.
  !Author:RainMan
```

```fortran
!2009-12-20
!LU分解法解线性方程组AX=B
!该程序从文件输入，格式为
!矩阵的维数是 3
!A矩阵
!4 2 1
!8 7 4
!16 17 15
!B矩阵
!11 34 95
SUBROUTINE LU(A,DIM)
  IMPLICIT NONE
  INTEGER:: DIM, I, J, K
  REAL(KIND=8):: A(DIM,DIM), m
  !
  DO I=2,DIM
    DO J=I,DIM
      A(J,I-1) = A(J,I-1)*1.0/A(I-1,I-1)
      m = A(J,I-1)
      DO K=I,DIM
        A(J,K) = A(J,K)-m*A(I-1,K)
      END DO
    END DO
  END DO
END SUBROUTINE LU

SUBROUTINE solve(A,B,DIM)
  IMPLICIT NONE
  INTEGER:: DIM, I, J
  REAL(KIND=8):: A(DIM,DIM),B(DIM,1)
!----------LY = B
  DO I=2,DIM
    DO J=1,I-1
      B(I,1) = B(I,1) - A(I,J)*B(J,1)
    END DO
  END DO

  B(DIM,1) = B(DIM,1)/A(DIM,DIM)
!---------UX=Y Y stores in B,same as X
  DO I=DIM-1,1,-1
    DO J=I+1,DIM
      B(I,1) = B(I,1) - A(I,J)*B(J,1)
    END DO
    B(I,1) = B(I,1)/A(I,I)
  END DO
END SUBROUTINE solve
! --------------------------------------------
! >>>>>>>>>>>>>>>
SUBROUTINE SOR( A, X, N, Residual, MaxIteration )
  IMPLICIT NONE
  INTEGER:: I, J
  INTEGER, INTENT(IN):: N
```

```
    INTEGER, INTENT(IN):: MaxIteration
    REAL(KIND=8), INTENT(IN):: Residual
    REAL(KIND=8), INTENT(IN):: A( N, N )
    REAL(KIND=8), INTENT(INOUT):: X( N, 1 )
    REAL(KIND=8):: B( N, 1 ), X1( N, 1 )
    REAL(KIND=8):: Omega
    REAL(KIND=8):: Residual0, Sum0
    INTEGER:: K0
    !
    ! Successive over relaxation coefficient
    Omega = 0.9d0
    !
    ! right side term
    B = X
    !
    X = 0.0d0
    X1 = X
    !
    Residual0 = 1.0d0
    K0 = 0
    DO WHILE ( ( Residual0 .GE. Residual ) .AND. ( K0 .LE. MaxIteration ) )
      DO I = 1, N
        Sum0 = 0.0d0
        Sum0 = Sum0 + Sum( A( I, 1:(I-1) ) * X( 1:(I-1), 1 ) ) / A( I, I )
!        DO J = 1, I - 1
!           Sum0 = Sum0 + A( I, J ) * X( J, 1 ) / A( I, I )
!        END DO
        !
        Sum0 = Sum0 + sum( A( I, (I+1):N ) * X( (I+1):N, 1 ) ) / A( I, I )
!        DO J = I + 1, N
!           Sum0 = Sum0 + A( I, J ) * X( J, 1 ) / A( I, I )
!        END DO
        !
        Sum0 = Sum0 + ( 1.0d0 - 1.0d0/Omega ) * X( I, 1 )
        Sum0 = Sum0 - B( I, 1 ) / A( I, I )
        X( I, 1 ) = - Omega * Sum0
      END DO
      Residual0 = Maxval( DABS( X1 - X ) )
      X1 = X
      K0 = K0 + 1
    END DO
  END SUBROUTINE SOR
  !~ ----------------------------
  ! >>>>>>>>>>>>>>.
  REAL(KIND=8) FUNCTION Distribution( Coordinate, Points, H, ModelFlag )
    IMPLICIT NONE
    INTEGER::I,J
    INTEGER, INTENT(IN):: ModelFlag
    REAL(KIND=8), INTENT(IN):: H, Coordinate(1,3),Points(1,3)
    REAL(KIND=8)::R1, R2, R3, F1, F2, F3
    !!
    R1 = DABS( ( Points( 1, 1 ) - Coordinate( 1, 1 ) ) / H );
    R2 = DABS( ( Points( 1, 2 ) - Coordinate( 1, 2 ) ) / H );
```

```
    R3 = DABS( ( Points( 1, 3 ) - Coordinate( 1, 3 ) ) / H );
    !!
    IF ( R1 < 1.0d0 ) THEN
       F1 = ( 3.0d0 - 2.0d0 * R1 + DSQRT( 1.0d0 + 4.0d0 * R1 - &
    &          4.0d0 * R1**2 ) ) / 8.0d0
    ELSE IF ( ( R1 >= 1.0d0 ) .AND. ( R1 < 2.0d0 ) ) THEN
       R1 = 2.0d0 - R1
       F1 = 0.5d0 - ( 3.0d0 - 2.0d0 * R1 + DSQRT( 1 + 4.0d0 &
    &          * R1 - 4.0d0 * r1**2 ) ) / 8.0d0
    ELSE
       F1 = 0.0d0
    END IF
    !!
    !!
    IF( R2 < 1.0d0 ) THEN
       F2 = ( 3.0d0 - 2.0d0 * R2 + DSQRT( 1.0d0 + 4.0d0 * R2 - &
    &          4.0d0 * R2**2 ) ) / 8.0d0
    ELSE IF ( ( R2 >= 1.0d0 ) .AND. ( R2 < 2.0d0 ) ) THEN
       R2 = 2.0d0 - R2
       F2 = 0.5d0 - ( 3.0d0 - 2.0d0 * R2 + DSQRT( 1.0d0 + 4.0d0 &
    &          * R2 - 4.0d0 * R2**2 ) ) / 8.0d0
    ELSE
       F2 = 0.0d0
    END IF
    !!
    !!
    IF( R3 < 1.0d0 ) THEN
       F3 = ( 3.0d0 - 2.0d0 * R3 + DSQRT( 1.0d0 + 4.0d0 * R3 - &
    &          4.0d0 * R3**2 ) ) / 8.0d0
    ELSE IF ( ( R3 >= 1.0d0 ) .AND. ( R3 < 2.0d0 ) ) THEN
       R3 = 2.0d0 - R3
       F3 = 0.5d0 - ( 3.0d0 - 2.0d0 * R3 + DSQRT( 1.0d0 + 4.0d0 &
    &          * R3 - 4.0d0 * R3**2 ) ) / 8.0d0
    ELSE
       F3 = 0.0d0
    END IF
    !
    IF ( ModelFlag == 0 ) THEN
       Distribution = F1 * F2 / H**2
    ELSE
       Distribution = F1 * F2 * F3 / H**3
    END IF
  END FUNCTION Distribution
  ! -------------------------------
  ! >>>>>>>>>>>.
  SUBROUTINE Filter( Q1, Q2, Q3, Q4, NA, XN1, YN2, ZN3, ModelFlag )
    USE FilterCoefficient
    IMPLICIT NONE
    INTEGER:: I, J, K
    INTEGER, INTENT(IN):: ModelFlag
    INTEGER, INTENT(IN):: NA, XN1, YN2, ZN3
    REAL(KIND=8), INTENT(INOUT):: Q1( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+N...
    REAL(KIND=8), INTENT(INOUT):: Q2( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+N...
```

```fortran
    REAL(KIND=8), INTENT(INOUT):: Q3( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+N...
    REAL(KIND=8), INTENT(INOUT):: Q4( -(NA-1) : (XN1+NA), -(NA-1) : (YN2+NA), -(NA-1) : (ZN3+N...
    REAL(KIND=8):: dQ1( 1:XN1, 1:YN2, 1:ZN3 ), dQ2( 1:XN1, 1:YN2, 1:ZN3 )
    REAL(KIND=8):: dQ3( 1:XN1, 1:YN2, 1:ZN3 ), dQ4( 1:XN1, 1:YN2, 1:ZN3 )
    !
    dQ1 = 0.0d0
    dQ2 = 0.0d0
    dQ3 = 0.0d0
    dQ4 = 0.0d0
    !
    ! for x - direction
    DO K = 1, ZN3
      DO J = 1, YN2
        CALL BasicFilter( dQ1(1:XN1, J, K), Q1(:, J, K), 1, XN1, -(NA-1), XN1+NA )
        CALL BasicFilter( dQ2(1:XN1, J, K), Q2(:, J, K), 1, XN1, -(NA-1), XN1+NA )
        CALL BasicFilter( dQ3(1:XN1, J, K), Q3(:, J, K), 1, XN1, -(NA-1), XN1+NA )
        CALL BasicFilter( dQ4(1:XN1, J, K), Q4(:, J, K), 1, XN1, -(NA-1), XN1+NA )
      END DO
    END DO
    !
    ! for y direction
    DO K = 1, ZN3
      DO I = 1, XN1
        CALL BasicFilter( dQ1(I, 1:YN2, K), Q1(I, :, K), 1, YN2, -(NA-1), YN2+NA )
        CALL BasicFilter( dQ2(I, 1:YN2, K), Q2(I, :, K), 1, YN2, -(NA-1), YN2+NA )
        CALL BasicFilter( dQ3(I, 1:YN2, K), Q3(I, :, K), 1, YN2, -(NA-1), YN2+NA )
        CALL BasicFilter( dQ4(I, 1:YN2, K), Q4(I, :, K), 1, YN2, -(NA-1), YN2+NA )
      END DO
    END DO
    !
    ! for z direction
    IF ( ModelFlag == 1 ) THEN
      ! 3-D model
      DO J = 1, YN2
        DO I = 1, XN1
          CALL BasicFilter( dQ1(I, J, 1:ZN3), Q1(I, J, :), 1, ZN3, -(NA-1), ZN3+NA )
          CALL BasicFilter( dQ2(I, J, 1:ZN3), Q2(I, J, :), 1, ZN3, -(NA-1), ZN3+NA )
          CALL BasicFilter( dQ3(I, J, 1:ZN3), Q3(I, J, :), 1, ZN3, -(NA-1), ZN3+NA )
          CALL BasicFilter( dQ4(I, J, 1:ZN3), Q4(I, J, :), 1, ZN3, -(NA-1), ZN3+NA )
        END DO
      END DO
    END IF
    !
    Q1( 1:XN1, 1:YN2, 1:ZN3 ) = Q1( 1:XN1, 1:YN2, 1:ZN3 ) - SigmaD * dQ1( 1:XN1, 1:YN2, 1:ZN3 )
    Q2( 1:XN1, 1:YN2, 1:ZN3 ) = Q2( 1:XN1, 1:YN2, 1:ZN3 ) - SigmaD * dQ2( 1:XN1, 1:YN2, 1:ZN3 )
    Q3( 1:XN1, 1:YN2, 1:ZN3 ) = Q3( 1:XN1, 1:YN2, 1:ZN3 ) - SigmaD * dQ3( 1:XN1, 1:YN2, 1:ZN3 )
    Q4( 1:XN1, 1:YN2, 1:ZN3 ) = Q4( 1:XN1, 1:YN2, 1:ZN3 ) - SigmaD * dQ4( 1:XN1, 1:YN2, 1:ZN3 )
  END SUBROUTINE Filter
```