```
!
!  Description: Subroutine implementing the CG method (parallel version)
!
! -----------------------------------------------------------------------

subroutine cg(A,u,b,tol,maxits,its)

! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!     Arguments:
!
!       Input:
!
!         A          local part of system matrix in compressed row format
!         b          right hand side vector in distributed form
!         tol        tolerance for iterative method
!         maxits     maximum number of iterations
!
!       Output:
!
!         u          solution vector in distibuted form
!
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

   use header

   implicit none

   type(Matrix), intent(inout) :: A
   type(Vector), intent(inout) :: u
   type(Vector), intent(inout) :: b
   real(kind=8), intent(in) :: tol
   integer, intent(in)  :: maxits

   real(kind=8) :: one, zero
   parameter (one = 1.0_8, zero = 0.0_8)

   real(kind=8) :: Vec_Dot    ! external function

   type(Vector) :: p
   type(Vector) :: q
   type(Vector) :: r
   real(kind=8) :: rtr,alpha,beta,gamma,delta,norm,norm0
   integer      :: n_loc, its
   n_loc = b%iend - b%ibeg + 1

! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!     Allocate memory for additional vectors p, q and r
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

   allocate(p%xx(b%n))
   allocate(q%xx(b%n))
   allocate(r%xx(b%n))

   p%n    = b%n
   p%ibeg = b%ibeg
   p%iend = b%iend

   q%n    = b%n
   q%ibeg = b%ibeg
   q%iend = b%iend

   r%n    = b%n
   r%ibeg = b%ibeg
   r%iend = b%iend

! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!     Beginning of program - Initialise solution vector and other vectors
```

```
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -


   !print*, 'inside conj u: ',u%xx
   !print*, ' inside conj a: ', A%aa
   u%xx(u%ibeg:u%iend) = zero

   call dcopy(n_loc,b%xx(u%ibeg),1,r%xx(u%ibeg),1)
   call dcopy(n_loc,b%xx(u%ibeg),1,p%xx(u%ibeg),1)

! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!     Calculate initial residual norm and stop if small enough
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

   rtr = Vec_Dot(r,r)

   norm0 = sqrt(rtr)

! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!     Iterate - up to kmax iterations
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   do its=1,maxits
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!     Implementation of one CG iteration
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

      call Mat_Mult(A,p,q)

      gamma = Vec_Dot(p,q)

      alpha = rtr / gamma

      call daxpy(n_loc,alpha,p%xx(p%ibeg),1,u%xx(u%ibeg),1)
      call daxpy(n_loc,-alpha,q%xx(q%ibeg),1,r%xx(r%ibeg),1)

      delta = Vec_Dot(r,r)

      beta = delta / rtr
      rtr  = delta

      norm = sqrt(rtr)
      if (norm/norm0 < tol) exit

      call dscal(n_loc,beta,p%xx(p%ibeg),1)
      call daxpy(n_loc,one,r%xx(r%ibeg),1,p%xx(p%ibeg),1)

   end do

   deallocate(p%xx)
   deallocate(q%xx)
   deallocate(r%xx)

end subroutine cg
```