```
!------------------------------------------------------------------------
!
!  Subroutine to multiply a matrix and a vector in compressed row storage
!  format , i.e to calculate b = A*u (parallel version).
!
!  Assumes that the total length of the vector is an integer multiple of
!  the number of processors
!
!------------------------------------------------------------------------

subroutine mat_mult(A,u,b)

   use header
   include "mpif.h"


   type(Matrix), intent(in) :: A
   type(Vector), intent(in) :: u
   type(Vector), intent(inout) :: b

   integer :: nprocs, i,j,n_loc,ierr,jbeg,jend
   real(kind = 8) :: ddot,d

   integer, allocatable , dimension(:) :: disp,recvcounts


! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!     Gather the entire vector u on each processor
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

   n_loc = u%iend - u%ibeg + 1
   call mpi_comm_size(MPI_COMM_WORLD, nprocs, ierr)
   allocate(disp(nprocs),recvcounts(nprocs))
   if (nprocs > 1) then

      ! does the length of u divide the number of processors?

      if (mod(u%n,nprocs) ==0)then

         ! if so, then use mpi_allgather to save computations i.e. displacement vector an↗
d
         ! recvcounts do not need to be calculated

         ! gather all distributed vectors from all processors

         call mpi_allgather(u%xx(u%ibeg),n_loc,MPI_DOUBLE_PRECISION, &
              u%xx,n_loc,MPI_DOUBLE_PRECISION, MPI_COMM_WORLD,ierr)
      else

         ! vector length of u does not divide the number of processors

         ! disp: integer array specifying the displacement at which to place the incoming↗
 data from processor

         ! recvcounts: integer array containing lengths of number of elements in distribu↗
ted vector

         do j=1,nprocs
            jbeg = int((j-1)*real(u%n)/nprocs)+1
            jend = int(j*real(u%n)/nprocs)
            recvcounts(j) = jend-jbeg+1
            disp(j) = jbeg - 1
         end do

         ! gather all distributed vector components from all processors

         call mpi_allgatherv(u%xx(u%ibeg),n_loc,mpi_double_precision, &
```

```
               u%xx, recvcounts,disp,mpi_double_precision,mpi_comm_world,ierr)
      end if
   end if

! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!     Calculate each component of b by taking the scalar product of the
!     i-th row of A and the vector u.
!
!       Note that in the compressed row storage format the nonzero
!       entries of row i are stored in
!
!         A%aa(A%ii(i)), A%aa(A%ii(i)+1), ..., A%aa(A%ii(i+1)-1)
!
!       the according (global) column numbers are stored in
!
!         A%jj(A%ii(i)), A%jj(A%ii(i)+1), ..., A%jj(A%ii(i+1)-1)
!
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

   !b%xx(b%ibeg:b%iend) = 0.0_8
   !do i = b%ibeg,b%iend
      !do i_j = A%ii(i), A%ii(i+1)-1
         !call daxpy(1,A%aa(i_j),u%xx(A%jj(i_j)),1,b%xx(i),1)
         !b%xx(i) = b%xx(i) + A%aa(i_j)*u%xx(A%jj(i_j))
      !end do
   !end do

   b%xx(b%ibeg:b%iend) = 0.0_8
   do i=b%ibeg,b%iend
      d=ddot(A%ii(i+1)-1-A%ii(i) + 1,A%aa(A%ii(i):(A%ii(i+1)-1)),1,u%xx(A%jj(A%ii(i):(A%i↗
i(i+1)-1))),1)
      call daxpy(1,1.0_8,d,1,b%xx(i),1)
   end do


   deallocate(disp,recvcounts)
end subroutine mat_mult
```