```fortran
!********************************************************************************
!
!  matrix_mul_vector_parallel.f90
! 并行矩阵向量乘法，基于一维行划分方法进行并行化
!
!********************************************************************************

program parallel_Mat_mul_Vec

    use mpi
    implicit none

    integer, parameter :: N = 2048
    integer :: my_left, my_right
    integer :: IERR, NPROC, NSTATUS(MPI_STATUS_SIZE)
    integer :: myrank, myleft, myright, myfile, buf_size, cnt
    real(4) :: startwtime, endwtime
    real(4), allocatable :: matrix_buf(:, :), vector_buf(:, :)
    real(4), allocatable :: matrix(:, :), vector(:, :), answer(:, :), answer_buf(:, :)

    call cpu_time(startwtime)

    call mpi_init(IERR)
    call mpi_comm_rank(MPI_COMM_WORLD, myrank, IERR)
    call mpi_comm_size(MPI_COMM_WORLD, NPROC, IERR)

    buf_size = N / NPROC
    myleft = my_left(myrank, NPROC)
    myright = my_right(myrank, NPROC)

    allocate(matrix_buf(N, buf_size)) ! 的矩阵存储方式为列存储，需要Fortran
                                       ! 进行一次转置，因此设置读取缓冲空间
    allocate(vector_buf(buf_size,1)) ! 接收其它进程储存的向量所需要的缓存空间
    allocate(matrix(buf_size, N))
    allocate(vector(buf_size, 1))
    allocate(answer(N, 1))
    allocate(answer_buf(N, 1))

    ! 读取矩阵
    call mpi_file_open(MPI_COMM_WORLD, "matrix", MPI_MODE_RDONLY, &
    &  MPI_INFO_NULL, myfile, IERR)
    call mpi_file_seek(myfile, myrank*N*buf_size*sizeof(MPI_REAL), &
    &  MPI_SEEK_SET, IERR)
    call mpi_file_read(myfile, matrix_buf, N*buf_size, MPI_REAL, &
    &  NSTATUS, IERR)
    call mpi_file_close(myfile, IERR)
    matrix = transpose(matrix_buf)

    ! 读取向量
    call mpi_file_open(MPI_COMM_WORLD, "vector", MPI_MODE_RDONLY, &
    &  MPI_INFO_NULL, myfile, IERR)
    call mpi_file_seek(myfile, myrank*buf_size*sizeof(MPI_REAL), &
    &  MPI_SEEK_SET, IERR)
    call mpi_file_read(myfile, vector, buf_size, MPI_REAL, &
    &  NSTATUS, IERR)
    call mpi_file_close(myfile, IERR)

    answer = 0
    deallocate(matrix_buf) ! 释放矩阵缓存空间用于储存每一次计算时的矩阵块
    allocate(matrix_buf(buf_size, buf_size))
    ! 循环进程中储存的所有矩阵块
    do cnt = 0, NPROC
        ! 计算对应矩阵块与向量的乘积
        matrix_buf = matrix(:, mod(myrank+cnt, NPROC)*buf_size+1:(mod(myrank+cnt, NPROC) &
        &  +1)*buf_size)
        answer(myrank*buf_size+1:(myrank+1)*buf_size, :) = matmul(matrix_buf,vector) &
        &  + answer(myrank*buf_size+1:(myrank+1)*buf_size, :)
        ! 进行一次向量块的传递向上()
        call mpi_send(vector, buf_size, MPI_REAL, myleft, myrank, &
        &  MPI_COMM_WORLD, IERR)
        call mpi_recv(vector_buf, buf_size, MPI_REAL, myright, myright, &
        &  MPI_COMM_WORLD, NSTATUS, IERR)
        vector = vector_buf
    end do

    ! 全规约结果向量，并行输出到文件
    call mpi_allreduce(answer, answer_buf, N, MPI_REAL, MPI_SUM, MPI_COMM_WORLD, IERR)
    call mpi_file_open(MPI_COMM_WORLD, "answer", MPI_MODE_CREATE+MPI_MODE_WRONLY, &
    &  MPI_INFO_NULL, myfile, IERR)
    call mpi_file_seek(myfile, myrank*buf_size*sizeof(MPI_REAL), MPI_SEEK_SET, &
    &  IERR)
    call mpi_file_write(myfile, answer_buf(myrank*buf_size+1, 1), buf_size, MPI_REAL, &
    &  MPI_STATUS_IGNORE, IERR)
    call mpi_file_close(myfile, IERR)

    call cpu_time(endwtime)
    write(*, *) "process", myrank, ":", 1000 * (endwtime - startwtime), "ms"

    deallocate(matrix_buf)
    deallocate(vector_buf)
    deallocate(matrix)
    deallocate(vector)
    deallocate(answer)
    deallocate(answer_buf)
    call mpi_finalize(IERR)

end program parallel_Mat_mul_Vec


!子程序和函数部分--------------------------------------------------------------

integer function my_left(myrank, nproc) result(ans)

    implicit none
    integer, intent(in) :: myrank, nproc

    ans = myrank - 1
    if (0 == myrank) ans = nproc - 1

end function my_left


integer function my_right(myrank, nproc) result(ans)

    implicit none
    integer, intent(in) :: myrank, nproc

    ans = myrank + 1
    if (nproc-1 == myrank) ans = 0

end function my_right
```