```fortran
! CGV/solution/parts/cgvp04.F90

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                                              !
! This file has been written as a sample solution to an        !
! exercise in a course given at the High Performance           !
! Computing Centre Stuttgart (HLRS).                           !
! It is made freely available with the understanding that      !
! every copy of this file must include this header and that    !
! HLRS takes no responsibility for the use of the enclosed     !
! teaching material.                                           !
!                                                              !
! Authors: Dr. Rolf Rabenseifner                               !
!          Stephan Scheiderer                                  !
!                                                              !
! Contact: rabenseifner@hlrs.de                                !
!                                                              !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
                                                           !EOH
!>>>>>>>>>>>>>>>>>>>>>>> begin of task 04 <<<<<<<<<<<<<<<<<<<<<<

!****************** domain decomposition *********************

!----------------------- 1-dim decomposition -----------------
!
!   Example: m=6, n=7, numprocs=2
!
!        |       j=0     1     2     3     4     5     6 |
!   ----+------------------------------------------------+-----------
!   i=0 |         0     1     2     3     4     5     6 |
!     1 |         7     8     9    10    11    12    13 | process 1
!     2 |        14    15    16    17    18    19    20 |
!   ----+------------------------------------------------+-----------
!     3 |        21    22    23    24    25    26    27 |
!     4 |        28    29    30    31    32    33    34 | process 2
!     5 |        35    36    37    38    39    40    41 |
!   ----+------------------------------------------------+-----------

!------------------------------
! Init_1dim
!
! TASK:
!   Do the one dimensional domain decomposition as depicted above.
!
! USED GLOBAL DATA:
!   OUT INTEGER start_i    : start row of the physical area this process is to calculate
!   OUT INTEGER end1_i     : end row + 1 of the physical area this process is to calculate
!   OUT INTEGER start_j    : start column of the physical area this process is to calculate
!   OUT INTEGER end1_j     : end column + 1 of the physical area this process is to calculate
!   IN/OUT INTEGER my_rank : the rank of the current process
!   OUT INTEGER comm_cart  : cartesian communicator
!   IN  INTEGER numprocs   : number of used processors
!   IN  INTEGER m          : vertical size of the physical problem
!   IN  INTEGER n          : horizontal size of the physical problem
!
! USED FUNCTIONS:
!   MPI:
!     MPI_CART_CREATE, MPI_COMM_RANK
SUBROUTINE Init_1dim()
    INTEGER :: slice_size
#ifndef serial
    INTEGER :: dims(0:1)
    LOGICAL :: periods(0:1)
    IF ((print_level >= 1) .AND. (my_rank == 0)) THEN
        WRITE(*,'(A, I3)') 'Using 1-dim domain decomposition, numprocs=', numprocs
    END IF
    ! Init the virtual topology (1-dim)
    dims(0) = numprocs
    dims(1) = 1
    periods(0) = .FALSE.
    periods(1) = .FALSE.
    CALL MPI_CART_CREATE(MPI_COMM_WORLD, 2, dims, periods, .FALSE., comm_cart, ierror)
    CALL MPI_COMM_RANK(comm_cart, my_rank, ierror)
#endif

    ! the size of each slice
    ! If m is not a multiple of numprocs, we round up the slice size
    slice_size = ((m - 1) / numprocs) + 1

    ! the rows of the physical area are distributed in slices of equal size
    ! (maybe except for the last slice) to each process
    start_i = (my_rank * slice_size)
    IF (start_i > m) THEN
        start_i = m
    END IF

    end1_i = start_i + slice_size
    IF (end1_i > m) THEN
        end1_i = m
    END IF

    ! we have the full vertical extent (all n columns) for each row
    start_j = 0
    end1_j = n
END SUBROUTINE Init_1dim

!----------------------- 2-dim decomposition -----------------
```

```
875  !
876  !   Example: m=5, n=7, m_procs=2, n_procs=2
877  !
878  !       |           j=0     1     2     3  |  4     5     6            |
879  !   ----+-------------------------------+---------------------------+
880  !   i=0 |             0     1     2     3  |  4     5     6            |
881  !     1 | process 1   7     8     9    10  | 11    12    13  process 2 |
882  !     2 |            14    15    16    17  | 18    19    20            |
883  !   ----+-------------------------------+---------------------------+
884  !     3 | process 3  21    22    23    24  | 25    26    27  process 4 |
885  !     4 |            28    29    30    31  | 32    33    34            |
886  !   ----+-------------------------------+---------------------------+
887
```

```
  888 !------------------------------
  889 ! Init_2dim
  890 !
  891 ! TASK:
  892 !   Do the two dimensional domain decomposition as depicted above.
  893 !
  894 ! USED GLOBAL DATA:
  895 !   OUT INTEGER start_i   : start row of the physical area this process is to calculate
  896 !   OUT INTEGER end1_i    : end row + 1 of the physical area this process is to calculate
  897 !   OUT INTEGER start_j   : start column of the physical area this process is to calculate
  898 !   OUT INTEGER end1_j    : end column + 1 of the physical area this process is to calculate
  899 !   IN  INTEGER my_rank   : the rank of the current process
  900 !   IN  INTEGER m_procs   : vertical number of processors
  901 !   IN  INTEGER n_procs   : horizontal number of processors
  902 !   OUT INTEGER comm_cart : cartesian communicator
  903 !   IN  INTEGER m         : vertical size of the physical problem
  904 !   IN  INTEGER n         : horizontal size of the physical problem
  905 !
  906 ! USED FUNCTIONS:
  907 !   APPL:
  908 !     abrt
  909 !   MPI:
  910 !     MPI_CART_CREATE, MPI_COMM_RANK
  911 SUBROUTINE Init_2dim()
  912     INTEGER :: blocksize_i, blocksize_j
  913     INTEGER :: startblock_i, startblock_j
  914 #ifndef serial
  915     INTEGER :: dims(0:1)
  916     LOGICAL :: periods(0:1)
  917     IF ((print_level >= 1) .AND. (my_rank == 0)) THEN
  918         WRITE(*,'(A, I3, A, I3, A, I3)') 'Using 2-dim domain decomposition, m_procs=', m_procs, ', &
  919        & n_procs=', n_procs, ', numprocs=', numprocs
  920     END IF
  921     ! Init the virtual topology (2-dims)
  922     dims(0) = m_procs
  923     dims(1) = n_procs
  924     periods(0) = .FALSE.
  925     periods(1) = .FALSE.
  926     CALL MPI_CART_CREATE(MPI_COMM_WORLD, 2, dims, periods, .FALSE., comm_cart, ierror)
  927     CALL MPI_COMM_RANK(comm_cart, my_rank, ierror)
  928 #endif
  929
  930     IF ((my_rank == 0) .AND. ((m_procs * n_procs) /= numprocs)) THEN
  931         WRITE(*,*) 'Number of processors does not fit! Use ', (m_procs * n_procs), ' processors instead. &
  932       & Aborting...'
  933         CALL abrt()
  934     END IF
  935
  936     ! the number of rows for each block (the last block might have
  937     ! less than the first blocks)
  938     blocksize_i = ((m - 1) / m_procs) + 1
  939
  940     ! the number of columns for each block (the last block might have
  941     ! less than the first blocks)
  942     blocksize_j = ((n - 1) / n_procs) + 1
  943
  944     ! calculate the start and end row and column
  945     startblock_i = my_rank / n_procs
  946     startblock_j = MODULO (my_rank, n_procs)
  947     start_i = blocksize_i * startblock_i
  948     end1_i = start_i + blocksize_i
  949     IF (end1_i > m) THEN
  950         end1_i = m
  951     END IF
  952     start_j = blocksize_j * startblock_j
  953     end1_j = start_j + blocksize_j
  954     IF (end1_j > n) THEN
  955         end1_j = n
  956     END IF
  957 END SUBROUTINE Init_2dim
  958
  959 !------------------------------
  960 ! Init_Decomp
  961 !
  962 ! TASK:
  963 !   Do the domain decomposition.
  964 !
  965 ! USED GLOBAL DATA:
  966 !   IN/OUT INTEGER rowsize     : number of data entries in a row of the local chunk
  967 !   IN/OUT INTEGER colsize     : number of data entries in a column of the local chunk
  968 !   IN/OUT INTEGER chunksize   : number of data entries in the local chunk PLUS halo data
  969 !   IN     INTEGER decomp_dims : dimension of the domain decomposition
  970 !   IN     INTEGER my_rank     : the rank of the current process
  971 !
  972 ! USED FUNCTIONS:
  973 !   APPL:
  974 !     abrt, Init_1dim, Init_2dim
  975 SUBROUTINE Init_Decomp()
  976     SELECT CASE (decomp_dims)
  977         CASE (1)
  978             CALL Init_1dim()
  979         CASE (2)
  980             CALL Init_2dim()
  981         CASE DEFAULT
  982             WRITE(*,*) 'Wrong decomp_dims=', decomp_dims
  983             CALL abrt()
  984     END SELECT
```

```
985       ! calculate the size of the rows, cols and the whole chunk
986       rowsize   = end1_j - start_j
987       colsize   = end1_i - start_i
988       chunksize = (rowsize + 2) * (colsize + 2)
989
990       IF ((rowsize < 2) .OR. (colsize < 2)) THEN
991           WRITE(*,*) 'Each process must have at least 2 entries in each dimension.'
992           WRITE(*,*) 'Decrease amount of processes or increase physical area size.'
993           WRITE(*, '(A, I3, A, I, A, I)') 'Rank[', my_rank, ']: rowsize = ', rowsize, ' -- colsize = ', colsize
994           WRITE(*,*) 'Aborting.'
995           CALL abrt()
996       END IF
997   END SUBROUTINE Init_Decomp
998
999   !>>>>>>>>>>>>>>>>>>>>>> -end- of task 04 <<<<<<<<<<<<<<<<<<<<<<
```