Solution 5

Solution 5

c

```
#include <stdlib.h>
#include <stdio.h>
#include <mpi.h>

#define N 20

int main(int argc, char* argv[]){

  int my_id, num_procs;
  int i,j;
  int rem, num_local_row;
  int *matrix;
  int proc_up, proc_down;
  MPI_Status  status1, status2;

  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&my_id);
  MPI_Comm_size(MPI_COMM_WORLD,&num_procs);

  /*  numer of rows for each mpi task */

  rem= N % num_procs;
  num_local_row = (N - rem)/num_procs;

  if(my_id < rem) num_local_row = num_local_row+1;

  matrix = (int*) malloc(((num_local_row+2)*N)*sizeof(int));

  /* inizialization of the local matrix */
  for(i=0; i<num_local_row+2 ;i++){
    for(j=0; j<N; j++){
      matrix[i*N+j] = my_id ;
    }
```

```
  }
  proc_up = my_id+1;
  proc_down = my_id-1;
  if(proc_up==num_procs) proc_up=0;
  if(proc_down < 0) proc_down=num_procs-1;

  /* check printings */
  /* printf("rank %d, proc up %d, proc down %d\n",
  my_id, proc_up, proc_down); */

  /* printf("my_id %d, num_local_row %d\n", my_id, num_local_row); */

  /* printf("my_id %d,matrix[0] %d, matrix[N] %d,
  matrix[(N)*(num_local_row+2)] %d \n", my_id, matrix[0],
  matrix[N], matrix[N*(num_local_row+2)]); */

  /* send receive of the ghost reagions */
  MPI_Sendrecv(&matrix[N],N,MPI_INTEGER,proc_down,10,
    &matrix[N*((num_local_row+1))],N,MPI_INTEGER,proc_up,10,
    MPI_COMM_WORLD,&status1);

  MPI_Sendrecv(&matrix[N*num_local_row],N,MPI_INTEGER,proc_up,
    11,&matrix[0],N,MPI_INTEGER,proc_down,11,MPI_COMM_WORLD,&status2);


  /* check printings */
  printf("\n ");

  printf("my_id, %d, riga arrivata da sopra: ", my_id);
  for(i=0;i<N;i++){
    printf("%d \t", matrix[i]);
  }
  printf("\n ");

  printf("my_id, %d, riga arrivata da sotto: ", my_id);
  for(i=N*(num_local_row+1);i<N*(num_local_row+1)+N;i++){
    printf("%d \t", matrix[i]);
  }
  printf("\n ");

  free(matrix);
  MPI_Finalize();
  return 0;
}
```

FORTRAN

```
program ghost_cells
    use mpi
    implicit none

    integer, parameter :: N=20
    integer :: my_id, num_procs, ierr
    integer :: i,j
    integer :: rem, num_local_col
    integer :: proc_right, proc_left
    integer, allocatable :: matrix(:,:)
    integer status1(MPI_Status_size), status2(MPI_Status_size)

    call mpi_init(ierr)
    call mpi_comm_rank(MPI_COMM_WORLD,my_id,ierr)
    call mpi_comm_size(MPI_COMM_WORLD,num_procs,ierr)

    !  number of columns for each mpi task

    rem= mod(N,num_procs)
    num_local_col = (N - rem)/num_procs

    if(my_id < rem num_local_col = num_local_col+1

    allocate(matrix(N,num_local_col+2))

    ! inizialization of the local matrix
    do j=1,num_local_col+2
       do i=1,N
          matrix(i,j) = my_id
       enddo
    enddo

    proc_right = my_id+1
    proc_left = my_id-1
    if(proc_right .eq. num_procs) proc_right = 0
    if(proc_left < 0) proc_left = num_procs-1

    ! check printings
    write(*,*) "my_id, proc right, proc left ", my_id, proc_right, proc_left
    write(*,*) "my_id, num_local_col ", my_id, num_local_col
    write(*,*) "my_id, matrix(1,1), matrix(1,num_local_col+2), matrix(N,num_local_col+2)", &
                my_id, matrix(1,1), matrix(1,num_local_col+2), &
                matrix(N,num_local_col+2)

    ! send receive of the ghost regions
    call mpi_sendrecv(matrix(:,2),N,MPI_INTEGER,proc_left,10, &
         matrix(:,(num_local_col+2)),N,MPI_INTEGER,proc_right,10, &
         MPI_COMM_WORLD,status1,ierr)

    call mpi_sendrecv(matrix(:,(num_local_col+1)),N,MPI_INTEGER,proc_right, &
         11,matrix(:,1),N,MPI_INTEGER,proc_left,11,MPI_COMM_WORLD,status2,ierr)
```

```
   ! check printings
   write(*,*) "my_id ", my_id, " colonna arrivata da sinistra: ", matrix(:,1)
   write(*,*) "my_id ", my_id, " colonna arrivata da destra: ", &
               matrix(:,num_local_col+2)

   deallocate(matrix)

   call mpi_finalize(ierr)

end program
```

‹ Exercise 5 up Exercise 6 ›