Solution 6

Solution 6

c

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

#define N 10

typedef int row[N];

int main(int argc, char *argv[])
{
  int np, rank, n, r;
  int I;
  int i,j;
  int k;
  row *mat;
  MPI_Status status;

  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&np);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);

  /* Number of rows that are assigned to each processor, taking care of the remainder */
  n = N / np;
  r = N % np;
  if (rank < r)
    n++;
  /* Allocate local workspace */
  mat = (row *) malloc(n * sizeof(row));
  /* Column of the first "one" entry */
  I = n*rank;
  if (rank >= r)
    I += r;
```

```
  /* Initilize local matrix */
  for (i=0; i < n; i++)
  {
    for (j=0; j<N; j++)
      if (j == I)
        mat[i][j] = 1;
      else
        mat[i][j] = 0;
    I++;
  }

  /* Print matrix */
  if (rank == 0)
    {
      /* Rank 0: print local buffer */
      for (i=0; i<n; i++)
    {
      for (j=0; j<N; j++)
        printf("%d ", mat[i][j]);
      printf("\n");
    }
      /* Receive new data from other processes
     in an ordered fashion and print the buffer */
      for (k=1; k < np; k++)
    {
      if(k==r){
        n = n -1;
      }
      MPI_Recv(mat, n*N, MPI_INT, k, 0, MPI_COMM_WORLD, &status);
      for (i=0; i < n; i++)
        {
          for (j=0; j<N; j++)
        printf("%d ", mat[i][j]);
          printf("\n");
        }

    }
    }
  else
    {
      /* Send local data to Rank 0 */
      MPI_Send(mat, n*N, MPI_INT, 0, 0, MPI_COMM_WORLD);
    }

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

FORTRAN

```
program distribuite

  use mpi

  implicit none

  character(LEN=50) :: stringa

  integer, parameter :: N = 10

  integer ierr, error
  integer status(MPI_STATUS_SIZE)

  integer k, i, j, rem, iglob, Ncol, Nrow, sup
  integer me, nprocs

  integer, allocatable, dimension(:,:) :: a

  call MPI_INIT(ierr)
  call MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
  call MPI_COMM_RANK(MPI_COMM_WORLD, me, ierr)

  !$ Number of rows that are assigned to each processor, taking care of the remainder
  Ncol = N
  Nrow = N / nprocs

  rem = MOD(N, nprocs)
  if (me < rem) then
     Nrow = Nrow + 1
  endif

  !$ Allocate local workspace (and notice that the array is distributed by columns)
  ALLOCATE( a(Ncol, Nrow) )
  !$ Logical column (Local row) of the first "one" entry
  iglob = (Nrow * me) + 1
  if (me >= rem) then
     iglob = iglob + rem
  endif
  !$ Initilize local matrix
  do j=1,Nrow
     do i=1,Ncol
        if (i == iglob) then
           A(i,j) = 1.0
        else
           A(i,j) = 0.0
        endif
     enddo
     iglob = iglob + 1;
  enddo
```

```
  write(stringa, *) Ncol
  !$ Print matrix
  if (me == 0) then
     !$ Rank 0: print local buffer
     do j=1,Nrow
        print '('//trim(stringa)//'(I2))', A(:,j)
     enddo
     !$ Receive new data from other processes
     !$ in an ordered fashion and print the buffer
     do k=1, nprocs-1
        if (k==rem) then
           Nrow = Nrow - 1
        endif
        call MPI_RECV(A, Nrow*Ncol, MPI_INTEGER, k, 0, MPI_COMM_WORLD, status, ierr)
        do j=1,Nrow
           print '('//trim(stringa)//'(I2))', A(:,j)
        enddo
     enddo
  else
     !$ Send local data to Rank 0
     call MPI_SEND(A, Nrow*Ncol, MPI_INTEGER, 0, 0, MPI_COMM_WORLD, ierr)
  endif
  DEALLOCATE(a)
  call MPI_FINALIZE(ierr)

end program distribuite
```

‹ Exercise 6 up Exercise 7 ›

© Copyright 2012 SCAI - SuperComputing Applications and Innovation - CINECA