Solution 14

C

```
/*
 * This program demonstrates the use of MPI_Alltoall when
 * transposing a square matrix.
 * For simplicity, the number of processes is 4 and the dimension
 * of the matrix is fixed to NROW
 */

#include <stdio.h>
#include <mpi.h>
#define NROW 131072
#define NP 1024
#define NBLK NROW/NP
#define ONEML 99999999999
void
trans (double *a, int n)
/* transpose square matrix a, dimension nxn
 * Consider this as a black box for the MPI course
 */

{
  int i, j;
  int ij, ji, l;
  double tmp;
  ij = 0;
  l = -1;
  for (i = 0; i < n; i++)
    {
      l += n + 1;
      ji = l;
      ij += i + 1;
      for (j = i+1; j < n; j++)
        {
```

```
          tmp = a[ij];
          a[ij] = a[ji];
          a[ji] = tmp;
          ij++;
          ji += n;
        }
    }
}

int main (int argc, char *argv[])
{
  double a[NROW][NBLK];
  double b[NROW][NBLK];

  int i, j, nprocs, rank,provided,itemp;
  double r0;

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

  if(rank == 0)
  {
    printf("Transposing a %d x %d matrix, divided among %d processors\n",NROW,NROW,NP);
  }
  if (nprocs != NP)
    {
      if (rank == 0)
        printf ("Error, number of processes must be %d\n",NP);
      MPI_Finalize ();
      return 1;
    }
 r0 = MPI_Wtime();
  for (i = 0; i < NROW; i++)
    for (j = 0; j < NBLK; j++)
      a[i][j] = ONEML * i + j + NBLK * rank; /* give every element a unique value */
 r0 = MPI_Wtime()-r0;
  if(rank == 0)
  {
    printf("Building matrix time (sec) %f\n",r0);
  }
  /* do the MPI part of the transpose */

  /* Tricky here is the number of items to send and receive.
   * Not NROWxNBLK as one may guess, but the amount to send to one process
   * and the amount to receive from any process */

 r0 = MPI_Wtime();
  /* MPI_Alltoall does not a transpose of the data received, we have to
   * do this ourself: */
  MPI_Alltoall (&a[0][0],       /* address of data to send  */
```

```
                NBLK * NBLK,    /* number of items to send to one process  */
                MPI_DOUBLE,     /* type of data  */
                &b[0][0],       /* address for receiving the data  */
                /* NOTE: send data and receive data may NOT overlap */
                NBLK * NBLK,    /* number of items to receive
                                   from any process  */
                MPI_DOUBLE,     /* type of receive data  */
                MPI_COMM_WORLD);

 r0 = MPI_Wtime()-r0;
  if(rank == 0)
  {
    printf("MPI_Alltoall time (sec) %f\n",r0);
  }

  /* transpose NP square matrices, order NBLKxNBLK: */
r0 = MPI_Wtime();
  for (i = 0; i < NP; i++)
      trans (&b[i * NBLK][0], NBLK);
r0 = MPI_Wtime()-r0;
  if(rank == 0)
  {
    printf("Transpose block matrices time (sec) %f\n",r0);
  }

   /* now check the result */

  for (i = 0; i < NROW; i++)
    for (j = 0; j < NBLK; j++)
      {
        if (b[i][j] != ONEML * (j + NBLK * rank) + i )
          {
            printf ("process %d found b[%d][%d] = %f, but %f was expected\n",
                    rank, i, j, b[i][j], (double) (ONEML * (j + NBLK * rank) + i));
            MPI_Abort (MPI_COMM_WORLD,1);
            return 1;
          }
      }
  if (rank == 0)
    printf ("Transpose seems ok\n");
  MPI_Finalize ();
  return 0;
}
```

FORTRAN

```
subroutine trans (a, n)
!* transpose square matrix a, dimension nxn
!* Consider this as a black box for the MPI course
!*/
  integer i, j, ij, ji, l,n
  double precision tmp
  double precision a(*)

  ij = 0
  l = -1
  do i = 0, n-1
      l = l+ n + 1
      ji = l
      ij = ij+i + 1
      do j=i+1,n-1
          tmp = a(ij+1)
          a(ij+1) = a(ji+1)
          a(ji+1) = tmp
          ij = ij+1
          ji = ji+ n
      enddo
  enddo
  return
end subroutine trans
!*
!* This program demonstrates the use of MPI_Alltoall when
!* transposing a square matrix.
!* For simplicity, the number of processes is 4 and the dimension
!* of the matrix is fixed to NROW
!*/
 program main
  implicit none
  include 'mpif.h'
  integer :: nprocs
  integer :: proc_me,proc_up,proc_down
    integer ierr
    integer,parameter::NROW=131072
    integer,parameter::NP=1024
    integer,parameter::NBLK=NROW/NP
    double precision,parameter::ONEML=99999999999.0
    double precision,allocatable:: a(:,:)
    double precision,allocatable:: b(:,:)

  integer i, j, rank,provided,itemp
  double precision r0
  call MPI_INIT(ierr)
  call MPI_COMM_SIZE(MPI_COMM_WORLD,nprocs,ierr)
  call MPI_COMM_RANK(MPI_COMM_WORLD,proc_me,ierr)
  allocate( a(NBLK,NROW),b(NBLK,NROW))

  if(proc_me.eq.0) write(*,*) 'Transposing a',NROW,'x',NROW,' matrix, divided among',NP,'processors'
```

```
  if (nprocs.ne.NP) then
      if (proc_me.eq.0) then
        write(*,*) 'Error, number of processes must be',NP
      end if
      call MPI_Finalize (ierr)
        stop
   endif
 r0 = MPI_Wtime()
  do j=1,NROW
     do i=1,NBLK
      a(i,j) = ONEML * j + i + NBLK *proc_me
     enddo
  enddo
 r0 = MPI_Wtime()-r0
 if(proc_me.eq.0) write(*,*) 'Building matrix time (sec)',r0

  !* do the MPI part of the transpose */
  !/* Tricky here is the number of items to send and receive.
  ! * Not NROWxNBLK as one may guess, but the amount to send to one process
  ! * and the amount to receive from any process */

 r0 = MPI_Wtime()
  !/* MPI_Alltoall does not a transpose of the data received, we have to
  ! * do this ourself: */
  call MPI_Alltoall (a(1,1), NBLK * NBLK, MPI_DOUBLE_PRECISION, b(1,1), NBLK * NBLK, MPI_DOUBLE_PRECISION, MPI_COMM_WORLD,ierr)

 r0 = MPI_Wtime()-r0
  if(proc_me.eq.0) write(*,*) 'MPI_Alltoall time (sec)',r0

  !* transpose NP square matrices, order NBLKxNBLK: */
r0 = MPI_Wtime()
  do i=1,NP
     call trans(b(1,(i-1) * NBLK+1),NBLK)
  enddo
r0 = MPI_Wtime()-r0;
  if(proc_me.eq.0) write(*,*) 'Transpose block matrices time (sec)',r0

   !* now check the result */
  do j=1,NROW
     do i=1,NBLK
        if (b(i,j).ne. ONEML * (i + NBLK * proc_me) + j ) then
            write(*,*) 'process',proc_me,'b',b(i,j),'expected', ONEML * (i + NBLK * proc_me) + j
            call MPI_Abort (MPI_COMM_WORLD,1)
        endif
     enddo
  enddo
  if (proc_me.eq.0) write(*,*) 'Transpose seems ok!'
  call MPI_Finalize (ierr)
end program main
```

Q\A exercise 14 answer

- Q/A Exercise 14

‹ Exercise 14 up Q/A Exercise 14 ›