Solution 13

C

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

#include "mpi.h"

#define PARALLEL_IO_PATH_TO_FILE "./output_c.dat"
#define ind(i,j) (i*lsizes[1]+j)

int main( int argc, char *argv[] )
{
//  Declare variables (or do it later)
    const int m = 10; // rows of global matrix
    const int n = 10; // cols of global matrix

//  Start MPI
    MPI_Init( &argc, &argv );

//  Set cartesian topology
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    int dims[2]; dims[0] = 0; dims[1] = 0;
    MPI_Dims_create(world_size, 2, dims);

    int periods[2]; periods[0] = 0; periods[1] = 0;

    MPI_Comm     comm;
    MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, 1, &comm);

    int rank, coords[2];
    MPI_Comm_rank(comm, &rank);
    MPI_Cart_coords(comm, rank, 2, coords);
    if (rank == 0) {
      printf("\nUsing a grid of [%d][%d] processes\n",
          dims[0], dims[1]);
    }
```

```
    // set subarray info
    int rem;
    int gsizes[2], psizes[2];
    int lsizes[2], start_indices[2];

    gsizes[0] = m;  /* no. of rows in global array */
    gsizes[1] = n;  /* no. of columns in global array*/

    psizes[0] = dims[0];  /* no. of processes in vertical dimension  of process grid */
    psizes[1] = dims[1];  /* no. of processes in horizontal dimension  of process grid */

    lsizes[0] = m/psizes[0];   /* no. of rows in local array */
    rem = m%psizes[0];
    if (rem && coords[0] < rem) {
      lsizes[0]++;
      start_indices[0] = coords[0] * lsizes[0];
    } else {
      start_indices[0] = rem + coords[0] * lsizes[0];
    }

    lsizes[1] = n/psizes[1];   /* no. of columns in local array */
    rem = n%psizes[1];
    if (rem && coords[1] < rem) {
      lsizes[1]++;
      start_indices[1] = coords[1] * lsizes[1];
    } else {
      start_indices[1] = rem + coords[1] * lsizes[1];
    }

    // initialize local matrix (array)
    int local_array_size = lsizes[0] * lsizes[1];
    int *local_array = (int *) malloc( local_array_size * sizeof(int) );
    for (int i=0; i<lsizes[0]; i++)
        for (int j=0; j<lsizes[1]; j++)
            local_array[ind(i,j)] = n*(i+start_indices[0]+1)+(j+start_indices[1]+1);

    /* create subarray filetype for MPI File view */
    MPI_File     fh;
    MPI_Datatype filetype;
    MPI_Status   f_status;

//  Create subarray, open and write file in parallel
    MPI_Type_create_subarray(2, gsizes, lsizes, start_indices,
                         MPI_ORDER_C, MPI_INT, &filetype);
    MPI_Type_commit(&filetype);

    MPI_File_delete(PARALLEL_IO_PATH_TO_FILE, MPI_INFO_NULL);
    MPI_File_open(MPI_COMM_WORLD, PARALLEL_IO_PATH_TO_FILE,
      MPI_MODE_CREATE | MPI_MODE_WRONLY,  MPI_INFO_NULL, &fh);
    MPI_Offset displ = 0;
```

```
MPI_File_set_view(fh, displ, MPI_INT, filetype, "native",  MPI_INFO_NULL);

MPI_File_write_all(fh, local_array, local_array_size,
       MPI_INT, &f_status);

MPI_File_close(&fh);

// Parallel verify
int errcount = 0;
int *verify_array = (int *) calloc( local_array_size , sizeof(int) );

MPI_File_open(MPI_COMM_WORLD, PARALLEL_IO_PATH_TO_FILE,
  MPI_MODE_RDONLY,  MPI_INFO_NULL, &fh);
MPI_File_set_view(fh, displ, MPI_INT, filetype, "native",  MPI_INFO_NULL);
MPI_File_read_all(fh, verify_array, local_array_size,
       MPI_INT, &f_status);
for (int i=0; i<lsizes[0]; i++)
    for (int j=0; j<lsizes[1]; j++)
        if (verify_array[(i,j)] != local_array[(i,j)]) {
           fprintf(stderr,"Parallel Verify ERROR: at indexes %d,%d read=%d, written=%d \n",
                   i, j, verify_array[ind(i,j)], local_array[ind(i,j)]);
           errcount++;
        }
if ( errcount == 0 ) {
  printf("Parallel Verify: test passed on proc(%d)\n", rank);
}

MPI_File_close(&fh);

MPI_Type_free(&filetype);
free(local_array);
free(verify_array);

// Serial verify
errcount = 0;
if(rank == 0) {
   FILE* fh;
   fh = fopen(PARALLEL_IO_PATH_TO_FILE,"rb");
   int *serial_verify_array = (int *) calloc(n , sizeof(int) );
   for (int i=0; i<m; i++) {
       fread(serial_verify_array,sizeof(int),n,fh);
       for (int j=0; j<n; j++) {
           if (serial_verify_array[j] != n*(i+1)+(j+1)) {
               fprintf(stderr,"Serial Verify ERROR: at index %d,%d read=%d, written=%d \n",
                      i, j, serial_verify_array[j], n*(i+1)+(j+1));
              errcount++;
           }
       }
   }

   if (!errcount) {
```

```
          printf("Serial Verify: test passed\n");
       }

       free(serial_verify_array);
       fclose(fh);
    }


//  MPI Finalize
    MPI_Finalize();

    return 0;
}
```

Warning for C programmers: depending on the compiler of your choice, you may need to add the flag -std=c99 because of some variable definitions that require C99 standard.

FORTRAN

```
#define PARALLEL_IO_PATH_TO_FILE "./output_fortran.dat"

program mpiio_subarray_matrix

!   Declare variables
    use mpi

    integer, parameter :: m = 10 ! rows of global matrix
    integer, parameter :: n = 10 ! cols of global matrix
    integer :: rank, world_size, ierr
    integer :: dims(2) , coords(2)
    logical :: periods(2), reorder
    integer :: comm, rem
    integer :: gsizes(2), psizes(2), lsizes(2), start_indices(2)
    integer :: local_array_size
    integer, dimension(:,:), allocatable :: local_array, verify_array
    integer, dimension(:), allocatable :: serial_verify_array
    integer :: fh, filetype
    integer, dimension(MPI_STATUS_SIZE) :: f_status
    integer(MPI_OFFSET_KIND) :: displ
    integer :: errcount

!   Start MPI
    call MPI_Init(ierr)

!   Set cartesian topology
    call MPI_Comm_size(MPI_COMM_WORLD, world_size, ierr)

    dims(1) = 0; dims(2) = 0;
```

```
    call MPI_Dims_create(world_size, 2, dims, ierr)

    periods(1) = .false. ; periods(2) = .false.
    reorder = .true.

    call MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, reorder, comm, ierr)

    call MPI_Comm_rank(comm, rank, ierr)
    call MPI_Cart_coords(comm, rank, 2, coords, ierr)
    if (rank == 0) then
       write(*,*) "Using a grid of [",dims(1),"][",dims(2),"] processes"
    endif

!   Set subarray info
    gsizes(1) = m  ! no. of rows in global array
    gsizes(2) = n  ! no. of columns in global array

    psizes(1) = dims(1)  ! no. of processes in vertical dimension  of process grid
    psizes(2) = dims(2)  ! no. of processes in horizontal dimension  of process grid
    lsizes(1) = m/psizes(1)   ! no. of rows in local array
    rem = mod(m,psizes(1))
    if (rem > 0 .and. coords(1) < rem) then
      lsizes(1) = lsizes(1) + 1
      start_indices(1) = coords(1) * lsizes(1)
    else
      start_indices(1) = rem + coords(1) * lsizes(1)
    endif

    lsizes(2) = n/psizes(2)   ! no. of columns in local array
    rem = mod(n,psizes(2))
    if (rem > 0 .and. coords(2) < rem) then
      lsizes(2) = lsizes(2) + 1
      start_indices(2) = coords(2) * lsizes(2)
    else
      start_indices(2) = rem + coords(2) * lsizes(2)
    endif

    local_array_size = lsizes(1) * lsizes(2)

!   Initialize local matrix
    allocate(local_array(lsizes(1),lsizes(2)))
    do j=1,lsizes(2)
    do i=1,lsizes(1)
       local_array(i,j) = m*(i+start_indices(1))+(j+start_indices(2))
    enddo
    enddo

!   Create subarray, open and write file in parallel
    call MPI_Type_create_subarray(2, gsizes, lsizes, start_indices,  &
                        MPI_ORDER_FORTRAN, MPI_INTEGER, filetype,ierr)
    call MPI_Type_commit(filetype,ierr)
```

```
    call MPI_File_delete(PARALLEL_IO_PATH_TO_FILE,MPI_INFO_NULL,ierr)
    call MPI_File_open(MPI_COMM_WORLD, PARALLEL_IO_PATH_TO_FILE,   &
           MPI_MODE_CREATE + MPI_MODE_WRONLY,  MPI_INFO_NULL, fh, ierr)
    displ = 0
    call MPI_File_set_view(fh, displ, MPI_INTEGER, filetype, "native",  MPI_INFO_NULL, ierr)

    call MPI_File_write_all(fh, local_array, local_array_size,   &
           MPI_INTEGER, f_status, ierr)

    call MPI_File_close(fh, ierr)

!   Parallel verify
    allocate(verify_array(lsizes(1),lsizes(2)))
    verify_array = 0

    call MPI_File_open(MPI_COMM_WORLD, PARALLEL_IO_PATH_TO_FILE,   &
                   MPI_MODE_RDONLY,  MPI_INFO_NULL, fh ,ierr)
    call MPI_File_set_view(fh, displ, MPI_INTEGER, filetype, "native",  MPI_INFO_NULL, ierr)
    call MPI_File_read_all(fh, verify_array, local_array_size,     &
           MPI_INTEGER, f_status, ierr)
    errcount = 0
    do j=1,lsizes(2)
    do i=1,lsizes(1)
       if (verify_array(i,j) /= local_array(i,j)) then
          errcount = errcount + 1
          write(*,*) "Parallel Verify ERROR: at index ",i,j," read=",verify_array(i,j)," written=",local_array(i,j)
       endif
    enddo
    enddo
    if(errcount == 0) write(*,*) 'Parallel test passed at proc: ',rank

    call MPI_File_close(fh, ierr)

    call MPI_Type_free(filetype, ierr)
    deallocate(local_array)
    deallocate(verify_array)

!   Serial verify
    if(rank == 0) then
       open(unit=1,file=PARALLEL_IO_PATH_TO_FILE,access='stream')
       allocate(serial_verify_array(m))
       errcount = 0
       do j=1,n
          read(1) serial_verify_array(1:m)
          do i=1,m
             if (serial_verify_array(i) /= m*i+j) then
                errcount = errcount + 1
                write(*,*) "Serial Verify ERROR : at index ",i,j," read=",serial_verify_array(i)," written=",m*i+j
             endif
          enddo
```

```
        enddo
        deallocate(serial_verify_array)
        close(1)
        if(errcount == 0) write(*,*) 'Serial test passed'
    endif

!   Finalize MPI
    call MPI_Finalize(ierr)

end program mpiio_subarray_matrix
```

‹ Exercise 13 up Exercise 14 ›