Solution 15

C

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

#define DIM 8

int printMat(float *mat, int numRow, int numCol, int mype, int nproc){

    float *buf;
    int i, j, k;
    MPI_Status status;

    if (mype == 0)
    {
        for (i=0; i<numRow; i++)
        {
            for (j=0; j<numCol; j++)
                printf("%.2f\t", mat[(i*numCol)+j]);
            printf("\n");
        }

        buf = (float *)malloc(numCol*numRow*sizeof(float));

        for (k=1; k < nproc; k++)
        {
            MPI_Recv(buf, numRow*numCol, MPI_FLOAT, k, 0, MPI_COMM_WORLD, &status);
            for (i=0; i < numRow; i++)
            {
                for (j=0; j<numCol; j++)
                    printf("%.2f\t", buf[(i*numCol)+j]);
                printf("\n");
            }

        }
```

```
        printf("\n\n\n");
    }
    else
        MPI_Send(mat, numRow*numCol, MPI_FLOAT, 0, 0, MPI_COMM_WORLD);

    return 0;
}



int main (int argc, char *argv[])
{
  FILE *fp;
  int count = 0, ind;
  float *A, *B, *C, *bufRecv, *bufSend, somma;
  int numRow, numCol, startInd;
  int nproc, mype;
  int i, j, y, k, sup, r;

  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD, &nproc);
  MPI_Comm_rank(MPI_COMM_WORLD, &mype);

  /* Get local matrix dimensions */
  numRow = DIM / nproc;
  numCol = DIM;
  /* Avoid remainders */
  if (r = DIM % nproc)
    {
      if (mype == 0) printf("The number of process must divide %d exactly.\n",DIM);
      MPI_Finalize();
      return -1;
    }

  /* Allocate workspace */
  A = (float *)malloc(numCol*numRow*sizeof(float));
  B = (float *)malloc(numCol*numRow*sizeof(float));
  C = (float *)malloc(numCol*numRow*sizeof(float));

  bufRecv = (float *)malloc(numCol*numRow*sizeof(float));
  bufSend = (float *)malloc(numRow*numRow*sizeof(float));

  /* Initialize matrices */
  for(i=0;i<numRow;++i)
  {
    for(j=0;j<numCol;++j)
      {
        A[(i*numCol)+j] = (((numRow * mype) + (i+1)) * (j+1));
        B[(i*numCol)+j] = 1 / A[(i*numCol)+j];
        C[(i*numCol)+j] = 0;
```

```
      }
  }
  /* Perform multiplication */
  for(count=0;count<nproc;++count)
    {
      startInd = count * numRow;
      for(i=0;i<numRow;++i)
        for(j=0;j<numRow;j++)
          {
            bufSend[j+(i*numRow)] = B[(i*numCol)+startInd+j];
          }

      MPI_Allgather(bufSend, numRow*numRow, MPI_FLOAT, bufRecv, numRow*numRow, MPI_FLOAT, MPI_COMM_WORLD);
      for(k=0;k<numRow;++k){
        for(i=0, somma=0.0;i<numRow;++i)
          {
            somma=0.0;
            for(j=0;j<numCol;++j)
              {
                somma += A[(k*numCol)+j] * bufRecv[(numRow*j)+i];
              }
            C[(count*numRow)+(k*numCol)+i] = somma;
          }
      }

    }

  /* Print matrices */
  printMat(A, numRow, numCol, mype, nproc);
  printMat(B, numRow, numCol, mype, nproc);
  printMat(C, numRow, numCol, mype, nproc);

  /* Free workspace */
  free(A);
  free(B);
  free(C);
  free(bufSend);
  free(bufRecv);

  MPI_Finalize();
  return 0;

}
```

FORTRAN

```
program Matmul
```

```
use mpi

implicit none

character(LEN=50) :: stringa

integer, parameter :: N = 8
integer :: nprocs, ierr, ecode, proc_me, status(MPI_STATUS_SIZE)
integer :: count, i, j, iglob, k, z, rem
real, allocatable, dimension(:,:) :: A, B, C, buffer
real, allocatable, dimension(:) :: sup
integer :: Nrow, Ncol

call MPI_INIT(ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD,nprocs,ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD,proc_me,ierr)
!$ Get local matrix dimensions
Nrow = N / nprocs
NCol = N

!$ Allocate workspace
allocate(A(Nrow,Ncol))
allocate(B(Nrow,Ncol))
allocate(C(Nrow,Ncol))
allocate(buffer(Nrow,Ncol))
allocate(sup(Ncol))

!$ Avoid the program to run with a number of processes that does not divide exactly dim
rem = mod(N,nprocs)
if ( rem .ne. 0 ) then
   if (proc_me .eq. 0) print *,"The number of process must divide ",N," exactly."
   call MPI_Abort(MPI_COMM_WORLD,ecode,ierr)
end if
!$ Initialize matrices
do j=1,Ncol
   do i=1,Nrow
      A(i,j) = ((Nrow * proc_me)+i) * j
      B(i,j) = 1.0 / A(i,j)
   enddo
enddo
C = 0.0

!$ Perform multiplication
do count=0,nprocs-1
   call MPI_Allgather(B(:,(count*Nrow)+1:count*Nrow+Nrow), Nrow*Nrow, MPI_REAL, buffer, Nrow*Nrow, MPI_REAL, MPI_COMM_WORLD, ier
   if (proc_me .eq. 0) then
      print *
      print *, buffer(1,:)
      print *, buffer(2,:)
   end if
   do k = (count*Nrow) + 1, (count+1) * Nrow
```

```
      do j = 1, Ncol
         do i = 1, Nrow
            C(i,k)  = C(i,k) + A(i,j) * buffer(mod(j-1,NRow)+1,((j-1)/Nrow)*Nrow + k - (count*Nrow))
         end do
      end do
  end do
end do

write(stringa, *) Ncol

!$ Print matrices

if (proc_me == 0) then
   print *
   do i=1,Nrow
      print '('//trim(stringa)//'(F8.2))', A(i,:)
   enddo

   do k=1, nprocs-1
      call MPI_RECV(A, Nrow*Ncol, MPI_REAL, k, 0, MPI_COMM_WORLD, status, ierr)
      do i=1,Nrow
         print '('//trim(stringa)//'(F8.2))', A(i,:)
      enddo
   enddo
else
   call MPI_SEND(A, Nrow*Ncol, MPI_REAL, 0, 0, MPI_COMM_WORLD, ierr)
endif

if (proc_me == 0) then
   print *
   do i=1,Nrow
      print '('//trim(stringa)//'(F8.2))', B(i,:)
   enddo

   do k=1, nprocs-1
      call MPI_RECV(B, Nrow*Ncol, MPI_REAL, k, 0, MPI_COMM_WORLD, status, ierr)
      do i=1,Nrow
         print '('//trim(stringa)//'(F8.2))', B(i,:)
      enddo
   enddo
else
   call MPI_SEND(B, Nrow*Ncol, MPI_REAL, 0, 0, MPI_COMM_WORLD, ierr)
endif

if (proc_me == 0) then
   print *
   do i=1,Nrow
      print '('//trim(stringa)//'(F8.2))', C(i,:)
   enddo

   do k=1, nprocs-1
```

```
        call MPI_RECV(C, Nrow*Ncol, MPI_REAL, k, 0, MPI_COMM_WORLD, status, ierr)
        do i=1,Nrow
           print '('//trim(stringa)//'(F8.2))', C(i,:)
        enddo
     enddo
  else
     call MPI_SEND(C, Nrow*Ncol, MPI_REAL, 0, 0, MPI_COMM_WORLD, ierr)
  endif

  !$ Free workspace
  deallocate(A,B,C,buffer,sup)
  call MPI_FINALIZE(ierr)

end program Matmul
```

‹ Exercise 15 up