ACM40640 Assignment 2

ICHEC

Deadline: 5th April 2020 at 23:00

1 Introduction

Please carry out all sections and document the code with appropriate comments. The assignments should be your own work. Marks will be deducted if there is copying between students or from online sources. Upload the files to BrightSpcae by the deadline.

2 Communication in a Ring

1. Communication in a ring:
   - Write a MPI program where each rank sends a message to its right neighbour.
   - The message is passed around the ring until it reaches the originator rank.
   - At this point the **message** should contain the sum of all the ranks.
   - So at termination all ranks should have the sum of the ranks.
   - Hints: use non-blocking communications, like the practical and the message does not need to a single integer.
   - Write an equivalent operation that does the same using a set MPI_Ireduce calls.

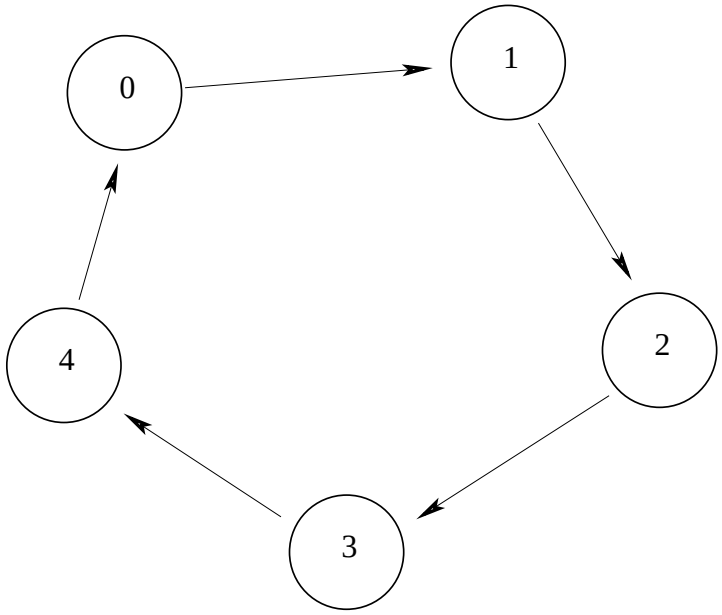

Figure 1. Five processes logically arranged in a ring: the left neighbor of P_0 is P_4 and its right neighbor is P_1.

3 Find Determinant using Cramer's Rule

2. Use Cramer's rule to find the determinant of the 5x5 matrix below, using 5 MPI processes. Calculate and print the result on rank 0. The actual result is $-2678797333.0/88905600000.0$.

- Below is an example of Cramer's rule (`http://en.wikipedia.org/wiki/Cramer's_rule`) for a 4x4 matrix.

$$\det\left(\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix}\right) = a\det\left(\begin{bmatrix} f & g & h \\ j & k & l \\ n & o & p \end{bmatrix}\right) - b\det\left(\begin{bmatrix} e & g & h \\ i & k & l \\ m & o & p \end{bmatrix}\right)$$
$$+ c\det\left(\begin{bmatrix} e & f & h \\ i & j & l \\ m & n & p \end{bmatrix}\right) - d\det\left(\begin{bmatrix} e & f & g \\ i & j & k \\ m & n & o \end{bmatrix}\right)$$

- The 5x5 matrix

$$\begin{pmatrix} 1 & -\frac{1}{2} & -\frac{1}{3} & -\frac{1}{4} & -\frac{1}{5} \\ -\frac{1}{2} & \frac{1}{3} & -\frac{1}{4} & -\frac{1}{5} & -\frac{1}{6} \\ -\frac{1}{3} & -\frac{1}{4} & \frac{1}{5} & -\frac{1}{6} & -\frac{1}{7} \\ -\frac{1}{4} & -\frac{1}{5} & -\frac{1}{6} & \frac{1}{7} & -\frac{1}{8} \\ -\frac{1}{5} & -\frac{1}{6} & -\frac{1}{7} & -\frac{1}{8} & \frac{1}{9} \end{pmatrix}$$

4 Find the Deadlock

3. The find_the_deadlock.f90 (find_the_deadlock.c) code in Section 5 contains a deadlock. Find and describe the deadlock and try to correct it. The expected behavior is:

   - processor 0 reads from a file (cut&paste the file values.dat) 25 random integers;
   - processor 0 broadcasts these integers to all the other processors;
   - every processor saves in a separate files these values.

   NOTE: There is a bug but also some errors in the program. Solving the deadlock does not mean you have reproduced the expected behavior. Take a look at all the program to see if everything is consistent and matches the specs. The expected wall-time of the corrected program is less than 30 second.

5 Source Code

File: *values.dat*:

```
13
23
21
19
20
15
25
3
11
7
8
14
12
5
10
18
2
17
6
16
22
24
9
1
4
```

```
program find_the_deadlock
   implicit none
   include 'mpif.h'
   integer :: rank, nprocs,ierror,fileunit
   integer :: indata(25)
   character(len=10) :: filenam

   call MPI_Init(ierror)
   call MPI_Comm_Size(MPI_COMM_WORLD,nprocs,ierror)
   call MPI_Comm_Rank(MPI_COMM_WORLD,rank,ierror)

   if (rank.eq.0) then
     open(1,FILE='values.dat')
     read(1,*,end=10) indata
     call MPI_Bcast(indata,25,MPI_DOUBLE_PRECISION,1,MPI_COMM_WORLD,ierror)
     close(1)
10   continue
   endif

   write(filenam,'(a,i0)') 'output.',rank

   ! output process rank 0 in file output.0
   ! output process rank 0 in file output.1
   ! output process rank 0 in file output.2
   ! and so on
   fileunit = rank+100
   write(fileunit,*) indata


  call MPI_Finalize(ierror)
end program find_the_deadlock
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
FILE *fr;

int main(int argc, char **argv){
   int i, ierr, rank, nprocs;
   long elapsed_seconds;
   int indata[25];
   char buf[256];

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

   if (rank == 0 ) {
    /* open the file for reading */
      fr = fopen ("values.dat", "rt");

      /* fill indata */
      i = 0;
      while (!feof(fr)) {
        fscanf(fr, "%d", &indata[i]);
        i = i + 1;
     }
      /* close the file */
     fclose(fr);

     ierr = MPI_Bcast( indata, 25, MPI_DOUBLE, 1, MPI_COMM_WORLD);
   }
     // output process rank 0 in file output.0
     // output process rank 1 in file output.1
     // output process rank 2 in file output.2
     // and so on ...
     snprintf(buf, sizeof(buf), "output.%d", rank);


     i = 0;
     while ( i<25 ) {
       fprintf(fr, "%d\n", indata[i]);
       i = i + 1;
     }

   ierr = MPI_Finalize();
   return 0;
}
```