Topic: MPI Programming

Objective

- MPI point-to-point communication
- Important points to remember while implementing MPI
- Hands-on

Topic: MPI Programming

Blocking and non-blocking

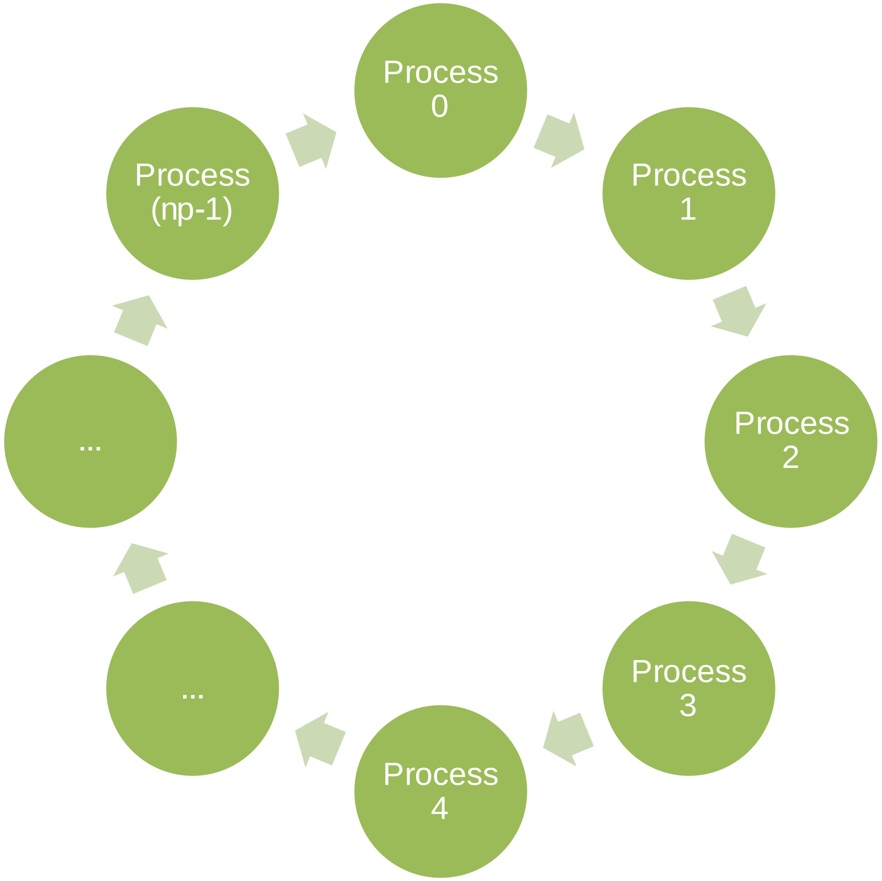
Blocking	Non-blocking
MPI_Send	MPI_Isend
MPI_Recv	MPI_Irecv

Blocking: the process does not return until data transmitted is started from the buffer

Non-blocking: the process return immediately after the op

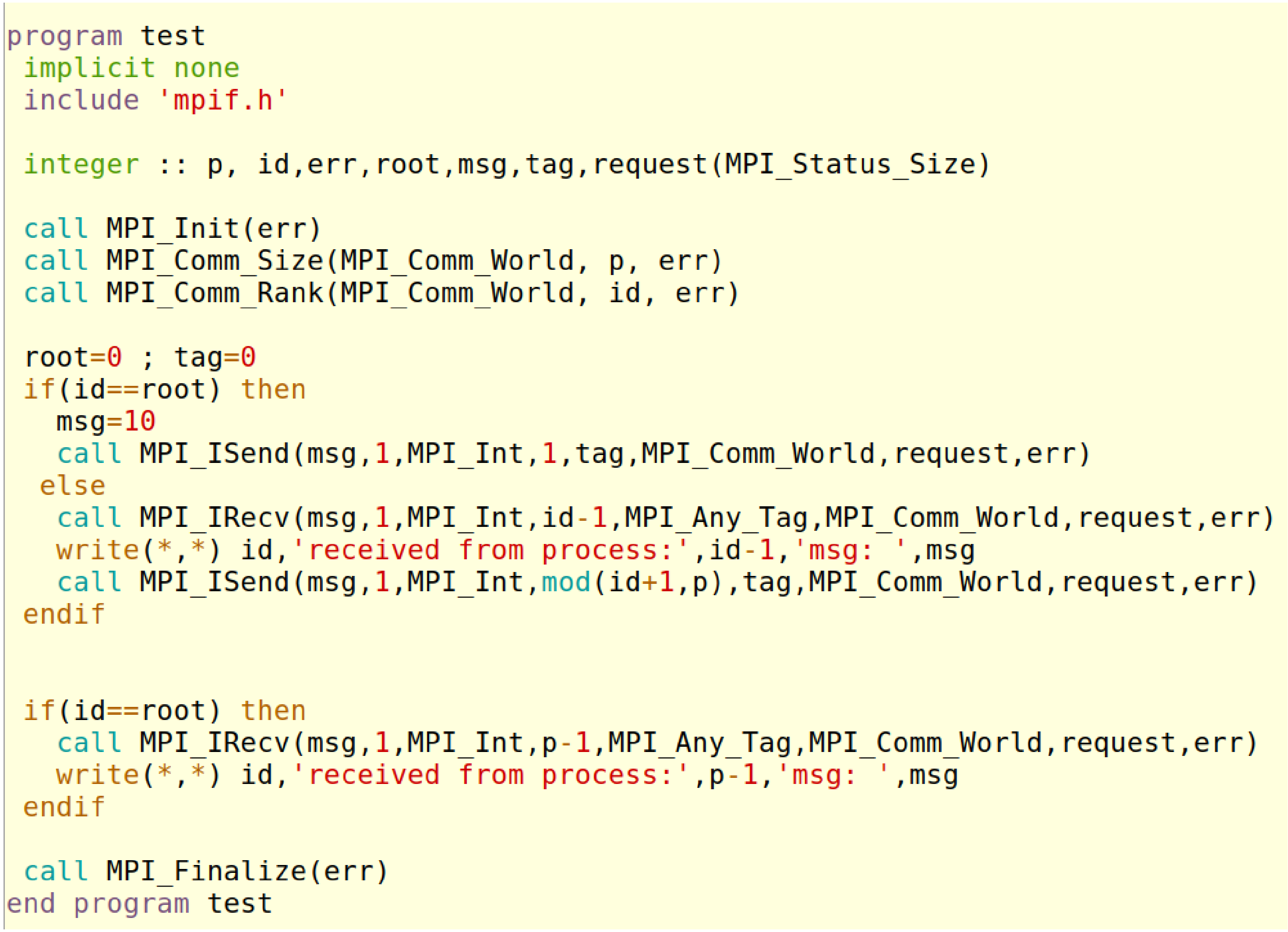
Topic: MPI Programming

Sending data in a ring-like pattern/topology



Topic: MPI Programming

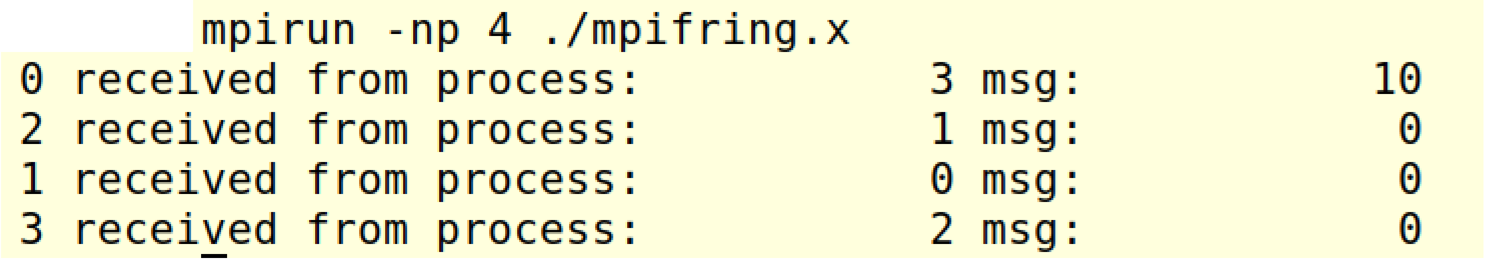
MPI_ISend/MPI_IRecv



```fortran
program test
 implicit none
 include 'mpif.h'

 integer :: p, id,err,root,msg,tag,request(MPI_Status_Size)

 call MPI_Init(err)
 call MPI_Comm_Size(MPI_Comm_World, p, err)
 call MPI_Comm_Rank(MPI_Comm_World, id, err)

 root=0 ; tag=0
 if(id==root) then
   msg=10
   call MPI_ISend(msg,1,MPI_Int,1,tag,MPI_Comm_World,request,err)
  else
   call MPI_IRecv(msg,1,MPI_Int,id-1,MPI_Any_Tag,MPI_Comm_World,request,err)
   write(*,*) id,'received from process:',id-1,'msg:  ',msg
   call MPI_ISend(msg,1,MPI_Int,mod(id+1,p),tag,MPI_Comm_World,request,err)
 endif


 if(id==root) then
   call MPI_IRecv(msg,1,MPI_Int,p-1,MPI_Any_Tag,MPI_Comm_World,request,err)
   write(*,*) id,'received from process:',p-1,'msg:  ',msg
 endif

 call MPI_Finalize(err)
end program test
```

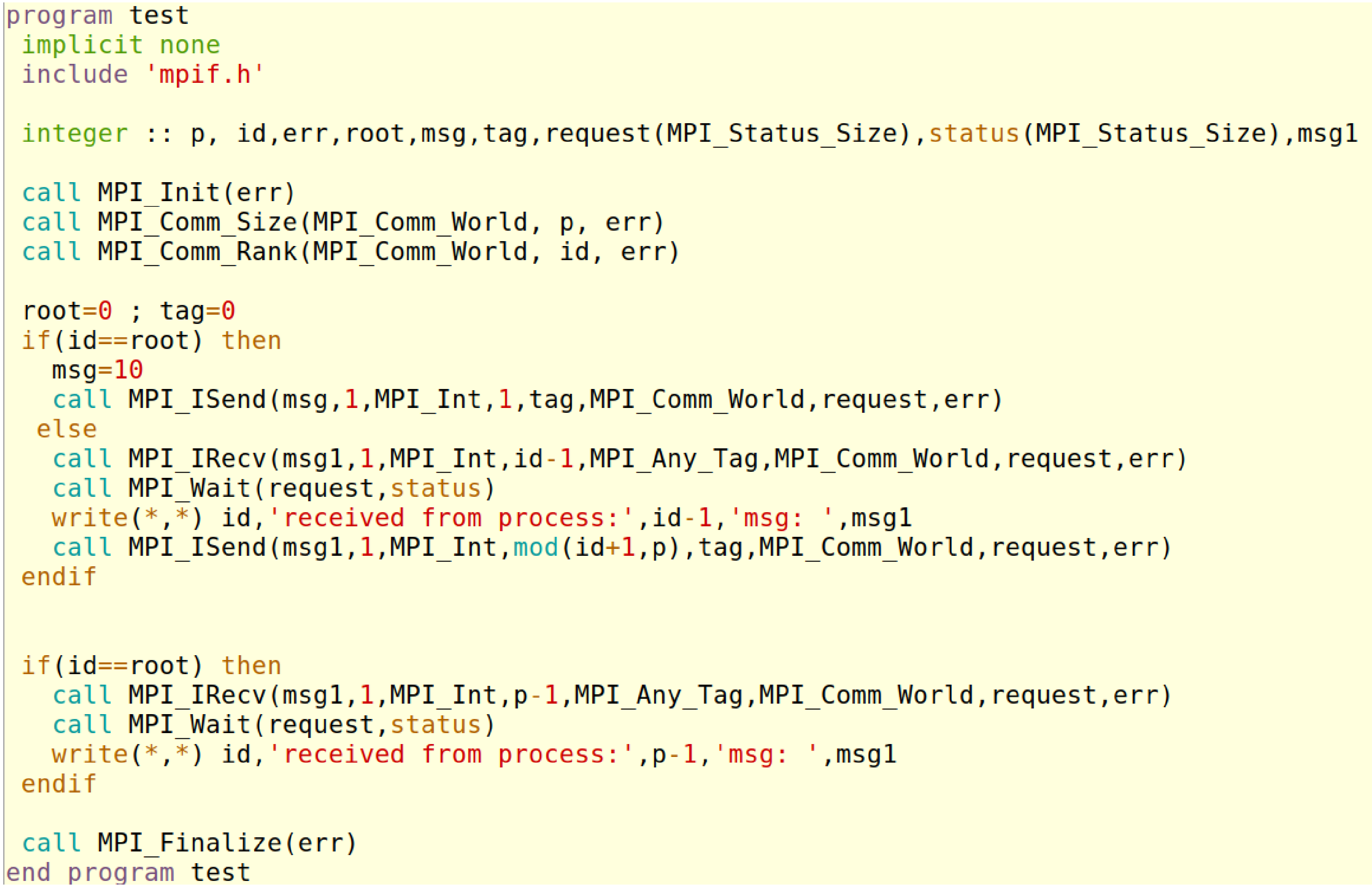
Topic: MPI Programming

Output



```
       mpirun -np 4 ./mpifring.x
0 received from process:           3 msg:          10
2 received from process:           1 msg:           0
1 received from process:           0 msg:           0
3 received from process:           2 msg:           0
```

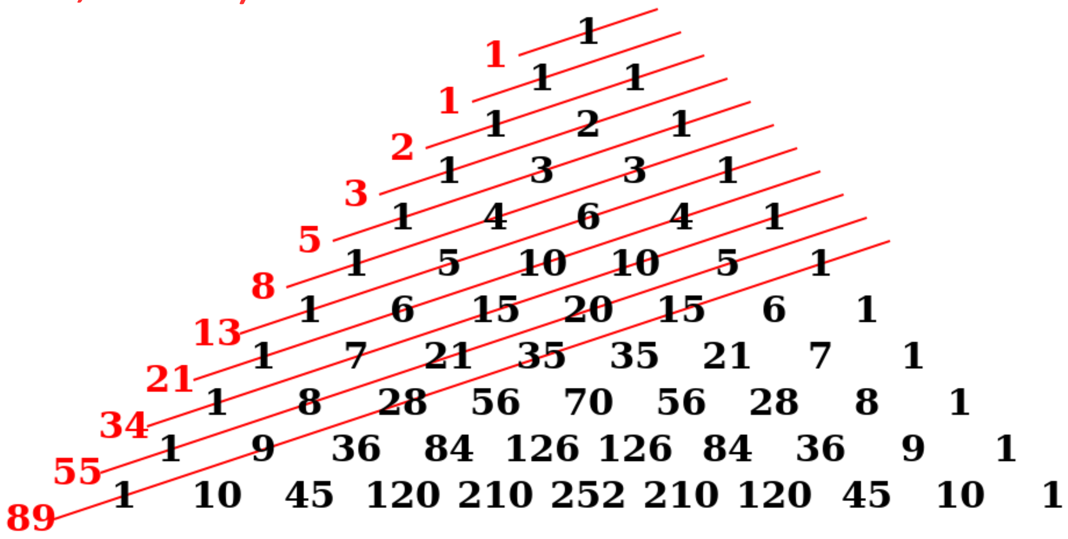
Correct – MPI_ISend/MPI_IRecv



```
program test
 implicit none
 include 'mpif.h'

 integer :: p, id,err,root,msg,tag,request(MPI_Status_Size),status(MPI_Status_Size),msg1

 call MPI_Init(err)
 call MPI_Comm_Size(MPI_Comm_World, p, err)
 call MPI_Comm_Rank(MPI_Comm_World, id, err)

 root=0 ; tag=0
 if(id==root) then
   msg=10
   call MPI_ISend(msg,1,MPI_Int,1,tag,MPI_Comm_World,request,err)
  else
   call MPI_IRecv(msg1,1,MPI_Int,id-1,MPI_Any_Tag,MPI_Comm_World,request,err)
   call MPI_Wait(request,status)
   write(*,*) id,'received from process:',id-1,'msg: ',msg1
   call MPI_ISend(msg1,1,MPI_Int,mod(id+1,p),tag,MPI_Comm_World,request,err)
 endif


 if(id==root) then
   call MPI_IRecv(msg1,1,MPI_Int,p-1,MPI_Any_Tag,MPI_Comm_World,request,err)
   call MPI_Wait(request,status)
   write(*,*) id,'received from process:',p-1,'msg: ',msg1
 endif

 call MPI_Finalize(err)
end program test
```

Can we parallelize all parts of the programs?

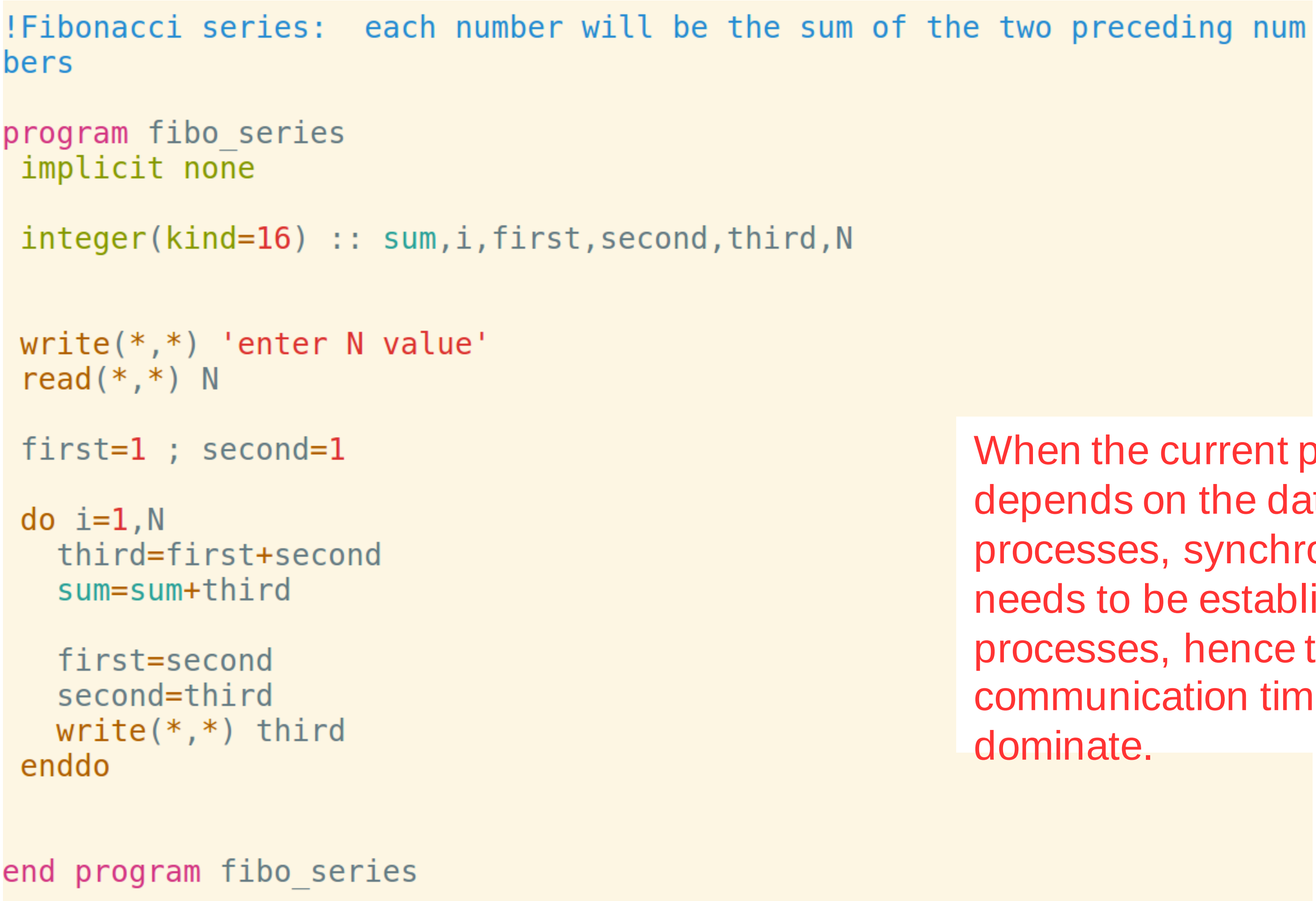
- Fibonacci series: Generate ‘N’ Fibonacci numbers and calculate their average (can we implement MPI, here?)



https://en.wikipedia.org/wiki/Fibonacci_number

Topic: MPI Programming

Can we parallelize all part of the programs? NO



```
!Fibonacci series:  each number will be the sum of the two preceding num
bers

program fibo_series
 implicit none

 integer(kind=16) :: sum,i,first,second,third,N


 write(*,*) 'enter N value'
 read(*,*) N

 first=1 ; second=1

 do i=1,N
   third=first+second
   sum=sum+third

   first=second
   second=third
   write(*,*) third
 enddo


end program fibo_series
```

When the current process depends on the data in other processes, synchronization needs to be established among processes, hence the communication time may dominate.

Topic: MPI Programming

Hands on

- Using the above data ‘random_numbers.dat’, write a program to calculate the average value of N numbers. a) Implement MPI using point-to-point blocking communication protocols b) Implement MPI using point-to-point non-blocking communication protocols. Show that the result in both cases is the same.

- **Optional:** Write a FORTRAN program to read the ‘300x300’ numbers given in ‘random_numbers.dat’ (posted in the general channel) and store them in an array. At each iteration, each number (r(I,j)) (except the numbers in first row, first column, last row and last column) is replaced by the average which is given by,

  r(i,j) = (r(i-1,j)+r(i+1,j)+r(i,j-1)+r(i,j+1))/4

  Do these iterations until the sum of absolute differences between the numbers in current and previous iteration is less than the tolerance 0.001. Can you get speed up in this program? What are you suggestions? (while implementing MPI, show that the number of iterations taken in both serial and parallel executions are the same for crosschecking)