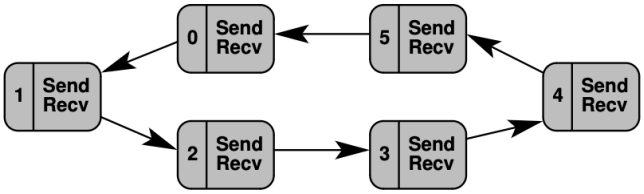
HOCH SCHULE OFFEN BURG

Instruction sheet for lab class 9

Exercise 1: Ring-communication

Write a function that transmits information in a ring-like pattern from process 0 to 1, from 1 to 2, … and finally from the last one back to 0. Use blocking communication functions MPI_Send and MPI_Recv.



The function signature shall be:

```
int ringSendRecv(int sendValue);
```

Each process calls the function with the argument being its rank within MPI_COMM_WORLD. The return value of the function is the value received from the other process. The function shall print a message:

```
"Proc <proc>: send <sendValue>, receive <receivedValue>"
```

For example, running with 4 processes, the sent and received values are:

Process	Sent	Received
0	0	3
1	1	0
2	2	1
3	3	2

a) Start by implementing a program that sends the number from process 0 to process 1. Print out the result on process 1 to verify that the message was correctly received.
b) Now implement all send/receive operations **with the exception of the last one from (n-1) to 0**. Again, print the results to verify that everything works as expected.
c) When implemented directly, communicating the last number would result in a deadlock. Why?
d) Avoid the deadlock by alternating the order of send/receive operations on odd numbered processes and finish the implementation. Test your implementation on the HPC cluster, making sure that you actually run on a distributed memory system.

Insert the output of your program when run with 4 processes, each running on a different compute node:

Blocking

```
Proc 0 sent to 1
Proc 1 received from 0
Proc 1 sent to 2
Process on hpcvl2, 1/4 finished.
Proc 2 received from 1
Proc 2 sent to 3
Process on hpcvl3, 2/4 finished.
Proc 3 received from 2
Proc 3 sent to 0
Process on hpcvl4, 3/4 finished.
Proc 0 received from 3
Process on hpcvl1, 0/4 finished.
```

Async

```
Proc 0 sent to 1
Proc 1 sent to 2
Proc 2 sent to 3
Proc 0 received from 3
Proc 3 sent to 0
Proc 2 received from 1
Proc 1 received from 0
Process on hpcvl1, 0/4 finished.
Process on hpcvl2, 1/4 finished.
Process on hpcvl3, 2/4 finished.
Proc 3 received from 2
Process on hpcvl4, 3/4 finished.
```

o moodle.

e) MPI_Isend and MPI_Irecv are non-blocking alternatives to the standard send and receive functions. The execution continues after a call, but you have to wait for the operation to finish before you may reuse the buffer. Modify the program to use non-blocking send and receive commands – this also avoids deadlocks.

Upload also this program to moodle.

Exercise 2: Computing pi

Consider again the program for computing pi. Implement a parallel version using MPI. The steps for this are quite typical for parallel programs:

1) Each process individually computes the local bounds for the integration (i.e. the first and last index for the for loop).
2) Each process individually computes the part of the integral as a partial sum.
3) The result of each partial sum is communicated to rank 0, where it is printed.
   MPI supports global communication, which you will learn about next week. For this week, you have to use direct calls to MPI_Send and MPI_Recv in order to communicate the result.

You can find a template for the implementation on moodle.

Upload your program to moodle.

Benchmark your program for 1,2,4,6,8,12,16,18,24,32,36 and 48 cores. Show plots of speedup and efficiency.
Hint: if done correctly, the speedup should be almost linear an the efficiency close to 100%.