Exercise 5

Task 5.1 (All-To-All-Broadcast)**.** An *All-To-All-Broadcast* is defined as a data exchange, where each node i sends its piece of data x_i to all other (participating) nodes. In the end each node has a copy of all data $x_0, x_1, \ldots, x_{p-1}$.

Todo: Write a message-passing pseudo *All-To-All-Broadcast* algorithm that works on a ring of processors, i.e. assume that node i may communicate directly only with nodes $i+1 \bmod p$ and $i-1 \bmod p$; the implementation should be as efficient as possible.

Hint: Starting a plain *Broadcast* on every node of the ring can lead to network congestion. As a result, the algorithm to be written should implement a dedicated strategy tailored to the network layout, using the *Send* and *Receive* primitives explicitly.
Solution:

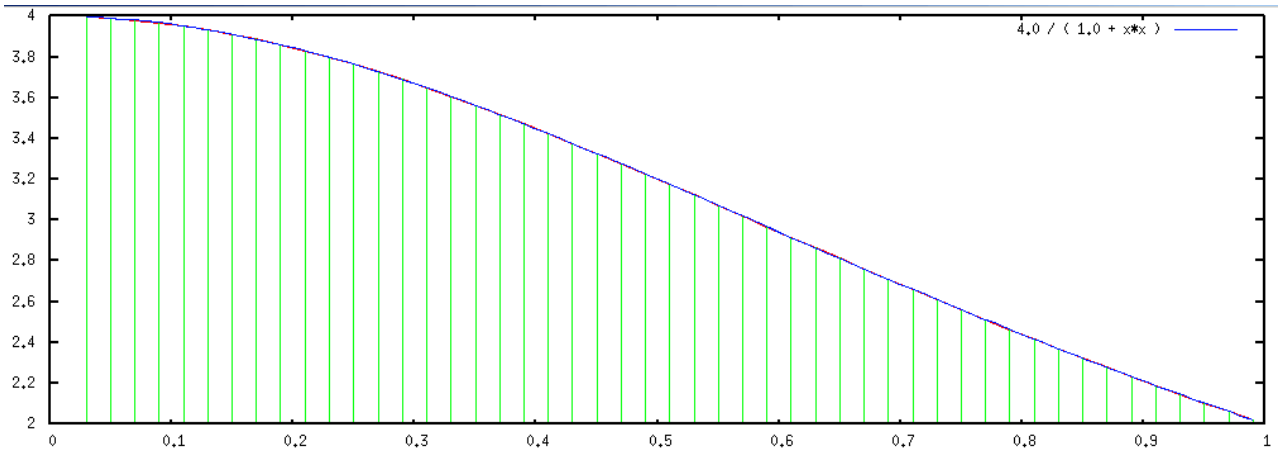
```
p = Comm_size
me = Comm_rank
if me % 2 == 0
    for i=0:p-1
      Send(x[(me -i) % p], (me + 1) % p)
      Recv(x[(me -i - 1) % p], (me - 1) % p)
    end
else
    for i=0:p-1
      Recv(x[(me -i - 1) % p], (me - 1) % p)
      Send(x[(me -i) % p], (me + 1) % p)
    end
end
```

Task 5.2 (Master-Worker)**.** During the lecture we introduced the "Master-Worker" concept, aka "Master-Slave". The **Master** divides the work into packages, which are then distributed among the available MPI processes (**Workers**), so they can perform the computations. In this exercise the familiar numerical integration method is to be parallelized using MPI. Please use the known point-to-point MPI primitives.
Hint: In the **master_worker** directory of the ZIP-Archive **Exercise_04.zip** you may find a skeleton for this exercise.
Hint: The program skeleton implements the following calculation:

$$\int_{x_0}^{x_1} f(x)\mathrm{d}x \approx h\sum_{i=1}^{N} f(x_i);\ x_i = x_0 + h(i-0.5);\ h = \frac{x_1 - x_0}{N}$$

A test for this numerical approximation of integrals is the calculation of π using the following identity:

$$\pi = 4\,\mathrm{atan}(1) = \int_0^1 \frac{4}{1+x^2}\mathrm{d}x$$



a) Compile and execute the program with different process numbers (NPROCS=2 and NPROCS=4). Why is there no noticeable speedup?

b) Modify the program to effectively use the available processors.

c) Please reorganize the program in such a way, that the first process (rank 0) calls a function called **master** and all other processes call the function **worker**. The program must still compute the integral. The work distribution should be performed by the **master**, while the calculation is to be performed by the **workers**.

d) The invocation of both MPI_Send and MPI_Recv for the computation for each individual h-sized interval is not particularly efficient. Implement the Master-Work technique using larger intervals as work items. What is the impact of this modification on the workload balance among the processes?

Solution: You may find a possible solution in the subdirectory **master_worker** of the ZIP archive **Solution_05.zip**

Task 5.3 (Global Communication)**.** Global communication was already approached in class. In these exercises, the associated MPI functions are studied in detail. For this purpose, you will emulate the functions with the primitives for sending and receiving.
Hint: In the **stepping_stone_globals** directory you can find a ZIP-Archive **Exercise_05.zip** containing a skeleton for this exercise.

a) Implement the function **reduce_plus_double(sendbuf, recvbuf, root, comm)**, one of the MPI_Reduce (similar to OpenMP reduction) functions. This function sums up the values provided by each of the MPI processes.

b) Can the associativity of addition be exploited to speed up the process? If yes, implement a solution.

c) The broadcast function MPI_Bcast(...) was already presented in a previous class. Implement the function **bcast_int** to broadcast single integers in the most efficient possible way. You can assume a fully connected network, i.e. each node has a direct connection via the network to every other node.

d) MPI has the function MPI_Barrier(...) which stops a process until every process reaches the barrier. Implement the function **barrier** to do just that.

Solution: You can find the solution to this task in subdirectory **stepping_stone_globals** of the zip-file **Solution_05.zip** .