MPI parallelization

Massage Passing:

- The same program runs on all processes (Single Program Multiple Data, SPMD).
- The program is written in a sequential language. Data exchange, i.e., sending and receiving of messages, is done via calls to an appropriate library.
- All variables in a process are local to this process. There is no concept of shared memory.
- The parallel loop is trivially parallel, with the only difference that it now operates on a fraction of the arrays.

SPMD model:

In SPMD model, each processor executes the same code. The local program has access only to local data.

- If I am processor 0 do nothing, otherwise receive an element from the left.
- If I am the last processor do nothing, otherwise send my element to the right.

See heat_mpi.c example

SPMD model:

blocking communication instructions: a send instruction does not finish until the sent item is actually received, and a receive instruction waits for the corresponding send. This means that sends and receives between processors have to be carefully paired.

This situation, where the program can not progress because every processor is waiting for another, is called *deadlock*.

Alternatively, *non-blocking* communication requires the use of a temporary buffer: a processor could put its data in a buffer, tell the system to make sure that it gets sent at some point, and later checks to see that the buffer is safe to reuse.

A possible solution to deadlock:

A simple solution to this deadlock problem is to interchange the MPI_Send() and MPI_Recv()

- If I am an odd numbered processor, I send first, then receive;
- If I am an even numbered processor, I receive first, then send.

Collective operations: (blocking)

- barrier: synchronizes the members of the communicator, i.e., all processes must call it before they are allowed to return to the user code.
- reduction: each processor has a data item, and these items need to be combined arithmetically with an addition, multiplication, max, or min operation. The result can be left on one processor, or on all, in which case we call this an allreduce operation.
- broadcast: one processor has a data item that all processors need to receive.

Collective operations: (blocking)

- gather : each processor has a data item, and these items need to be collected in an array, without combining them in an operations such as an addition. The result can be left on one processor, or on all, in which case we call this an allgather.
- scatter : one processor has an array of data items, and each processor receives one element of that array.
- all-to-all : each processor has an array of items, to be scattered to all other processors.

Homework

- Use MPI barrier, broadcast, scatter, and gather to do a matrix multiplication.
- Have a look at the attached OpenMP Jacobi iterative solver for a system of linear equations. Use barrier, broadcast, scatter, and gather and write an MPI implementation of the Jacobi solver.