University of Amsterdam
BSc Informatica

**Distributed and Parallel Programming**

# Assignment 3: MPI

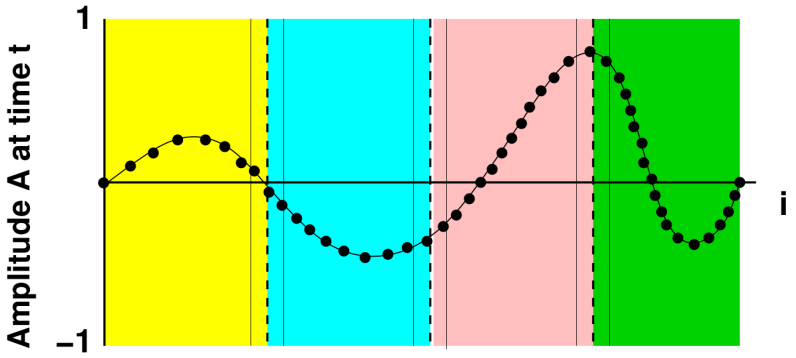
**Parallel Programming for Distributed Memory Systems**

Originally proposed by: dr. Clemens Grelck

November 2025

## Assignment 3.1: Wave equation simulation with MPI

Reconsider the 1-dimensional wave equation studied in assignments 1.1 and 1.2. Write a distributed MPI program in C that uses multiple compute nodes to simulate the wave equation following the domain decomposition approach illustrated below (similar to what was explained during the lecture).




Each MPI process *shall only store the relevant parts of the three amplitude buffers in its local memory*. Add halo cells as necessary, and exchange their values between time steps as needed. Aim at an efficient implementation w.r.t. communication and memory, and use blocking MPI calls to implement the communication between processes.

Analyse the performance of your application by running experiments with different problem sizes, i.e. number of amplitude points being $10^3$, $10^4$, $10^5$, $10^6$, $10^7$. Report your results as speedup graphs: sequential execution time divided by parallel execution time using *at least* three different system configurations: up to 8 MPI processes on a single node, up to 8 nodes with a single MPI process per node, and up to 8 nodes with 8 MPI processes per node.

Please note that your code has to run correctly (i.e., it needs to produce correct results) for any reasonable number of processes/nodes, including $1$. Therefore, design and implement code for any number of processes (that is, do not design, implement, or test your code only targeting the number of processes given above).

Compare the results of the new experiments with those of assignments 1.1 and 1.2. To ensure a meaningful comparison, please *be careful to compare performance with similar execution parameters.*

## Assignment 3.2: Overlapping communication and computation

Overlapping communication (overhead) with computation (productive processing) is essential to achieve good performance. Design two more versions of the 1-d wave equation that overlap the communication with computation more effectively by:

1. Using non-blocking send / blocking receive.

2. Using only non-blocking communication.

Explain your design and your implementation (e.g., using pseudo-code), and make sure you emphasize the differences with the approach in 3.1, as well as the differences between the two implementations - both in terms of the actual implementation, and the expected performance.

**Optionally**, you can implement these new solutions (one or both of them - in case you expect the performance to be significantly different), and compare their measured performance against the previous version.

## Assignment 3.3: Collective communication

Collective communication is a form of structured communication, where, instead of a dedicated sender and a dedicated receiver, all MPI processes of a given communicator participate. A simple example is broadcast communication where a given MPI process sends one message to all other MPI processes in the communicator. Write an MPI function in C:

```
int MYMPI_Bcast (
    void *buffer,          // INOUT : buffer address
    int count,             // IN    : buffer size
    MPI_Datatype datatype, // IN    : datatype of entry
    int root,              // IN    : root process (sender)
    MPI_Comm communicator  // IN    : commuicator
)
```

The function should implement broadcast communication by means of point-to-point communication. Each MPI process of the given communicator is assumed to execute the broadcast function with the same message parameters and root process argument. The root process is supposed to send the content of its own buffer to all other processes of the communicator. Any non-root process uses the given buffer for receiving the corresponding data from the root process.

A particular advantage of collective communication operations is that their implementation can take advantage of the underlying physical network topology without making an MPI-based application program specific to any such topology. For your implementation of broadcast assume a 1-dimensional ring topology, where each node only has communication links with its two direct neighbours with

(circularly) increasing and decreasing MPI process ids. While any MPI process may, nonetheless, send messages to any other MPI process, messages need to be routed through a number of intermediate nodes. Communication cost can be assumed to be linear in the number of nodes involved. Aim for an efficient implementation of your function on an (imaginary) ring network topology.

No performance-analysis experiments are required for this assignment. However, you should empirically check that your solution actually works, and describe your solution in detail in your report: how it is adapted to the given network topology and how many atomic communication events (sending a message from one node to a neighbouring node) are required to complete one broadcast.

## Additional Information

### Final notes on the framework architecture and the expected behavior of the code

- For grading purposes, we separate the code between application initialization (i.e., boiler-plate code) and MPI. All MPI code must be hosted in the simulate.c (we acknowledge this is not best practice, but it is accepted and deterministic for the OpenMPI implementation we use). The MPI process is considered to be the code in simulate.c, in the simulate function. For this assignment, you can assume the application works in a fork-join model, similar to OpenMP (i.e., MPI_Init and MPI_Finalize determine the fork and join, respectively).

- Also due to the implementation of the framework, all processes end up having full access to the full arrays. This is rarely feasible in a real large-scale application, where data is assumed not to fit in full in the memory of any node. Therefore, we ask you to assume limited space for the arrays used by MPI workers. Thus, to clarify, the statement *"Each MPI process shall only store the relevant parts of the three amplitude buffers in its local memory"* specifically refers to having each MPI worker only access, for calculation, sending, and receiving purposes, "local" arrays of approximately localSize+2 elements. The initialization of these local arrays can be done (a) by receiving data from the master, or (b) by correctly aliasing/copying the right chunk of the large arrays. Either way, the master is assumed to have sufficient memory, and must concatenate the results in its "full-size" current array, which is then the result of the simulation.

- The data distribution during initialization can be done by: (a) sending and receiving messages with the actual data, (b) sending and receiving the start and end/length of the portion allocated to each worker, or (c) having an agreement for data partitioning that is applied by every node independently. In practice, option (a) is used when the actual data is not shared, option (b) is used when the workload is dynamically balancend, and option (c) is used when a trivial, static partitioning is to be performed. You can choose whichever option you want, but please explain why, ideally by discussing the pro's and con's of the different options

- We assume the master thread also participates in the computation. Specifically, (potentially after it distributes the data to the others,) it actively participates in the calculation, just like any other worker. Again, when all work is completed, by all processes, it is the master process who collects the data and composes the result array

## Grade weights

Table 1: Weights of different parts of the assignments used in calculating the final grades.

| Assignment | Name | Code | Report |
|---|---|---|---|
| 3.1 | Wave equation simulation with MPI | 35% | 20% |
| 3.2 | Overlapping communication and computation | 0% | 20% |
| 3.3 | Collective communication | 15% | 10% |
| | | 50% | 50% |

## Instructions for submission

For this assignment we expect two deliverables:

1. Source code in the form of a tar-archive of all relevant files that make up the solution of a programming exercise with an adequate amount of comments ready to be compiled.

2. A report in the form of a pdf file that explains the developed solution at a higher level of abstraction, illustrates and discusses the outcomes of experiments and draws conclusions from the observations made. For indications on how to write the report, check the report writing guidelines on Canvas[1].

Please submit one archive containing both your code and your report (one report for the entire assignment).

[1] These are the same as the indications from assignment 1