Universiteit van Amsterdam
BSc Informatica
Dr Clemens Grelck

Concurrency en
Parallel Programmeren
2015/2016

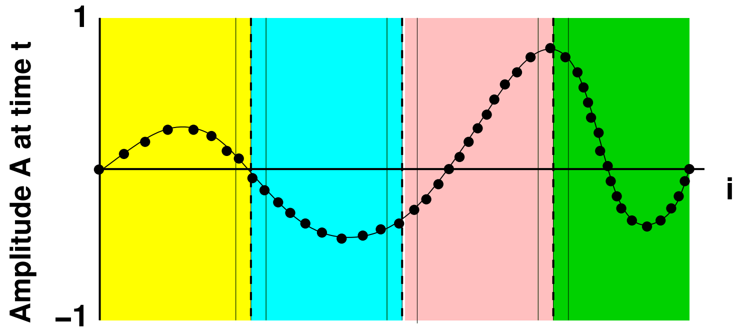
Assignment 2

Parallel Programming for Distributed Memory Systems

Assignment 2.1: Wave equation simulation with MPI

(Code: 35%, Report: 25%)

Reconsider the 1-dimensional wave equation studied in assignment 1.1. Write a distributed C+MPI program that uses multiple compute nodes to simulate the wave equation following the domain decomposition approach as illustrated below (and 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 that overlaps communication with computation as demonstrated during the lecture.

Repeat your experiments of assignment series 1 with 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. Compare the results of the new experiments with those of assignments 1.1 and 1.2.

Assignment 2.2: Overlapping communication and computation

(Code: 0%, Report: 20%)

Overlapping communication (overhead) with computation (productive processing) is key to achieve good performance. Develop two versions of the 1-d wave equation that more effectively overlap the communication with computation than the example shown in the lecture:

a) solely with the standard non-blocking send / blocking receive;

b) with making use of asynchronous communication.

Use pseudo code notation in analogy to the lecture and explain your pseudo code in plain text.

Assignment 2.3: Collective communication

(Code: 10%, Report: 10%)

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 a C+MPI function

```
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    : communicator            */
```

that implements 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 may 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 experiments are required for this assignment, but make sure your solution actually works and describe your code 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.

Instructions for submission:

We expect two kinds of deliverables:

a) 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; and

b) 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.

Please, submit one archive with all code files and separately one pdf-file with your report for the entire assignment.

Code due date: November 18, 2015, noon
Report due date: November 20, 2015, midnight