Exercise Sheet 6 for Lecture Parallel Programming

Deadline: 2025-06-16, 23:59

Prof. Dr. Michael Kuhn (michael.kuhn@ovgu.de)
Michael Blesel (michael.blesel@ovgu.de)
Parallel Computing and I/O • Institute for Intelligent Cooperating Systems
Faculty of Computer Science • Otto von Guericke University Magdeburg
https://parcio.ovgu.de

This exercise sheet will give you the opportunity to take your first steps in programming with MPI. You will use this knowledge for the coming exercise sheets to solve more complex problems with MPI.

Include the `mpi.h` header and use the `mpicc` compiler. To get access to the MPI compiler on the cluster, you need to load the software stack via the following command.

```
. /opt/spack/pp-2025/env.sh
```

You can find a tutorial for programming with MPI at:

https://hpc-tutorials.llnl.gov/mpi/

1. The First MPI Program (180 Points)

Create an MPI program `timempi` in C which creates the following output:

```
HOSTNAME: TIMESTAMP
```

HOSTNAME: Short hostname of the system that is executing the program.

TIMESTAMP: Timestamp of the time when the program is executed with a precision of at least microseconds.

The program shall adhere to the following requirements:

- The processes with ranks 1 to n shall create the string `HOSTNAME: TIMESTAMP` and send it as a string via MPI to the rank 0 process, which will take care of all the output of the program. Rank 0 shall **not** create and output its own string.
  (Hint: Take a look at the `gethostname()` and `gettimeofday()` functions. In the materials, you can find `timestamp.c` with an example for the timestamp creation.)
- The output shall be ordered by process rank.
- Directly after the output of the received string, the process with rank 0 shall additionally output the smallest microsecond amount out of all processes (including rank 0).
  (Hint: Take a look at `MPI_Reduce`.)
- None of the processes shall terminate before the output has happened. The program is incorrect if a process could terminate too early!

- Directly before termination every process shall output a text similar to: “Rank X terminating now!”
  (Hint: Take a look at `MPI_Barrier`.)
- The program must work with an arbitrary amount of processes.

The output could look like the following:

```
ant13: 2021-11-14 13:15:57.968558
ant14: 2021-11-14 13:15:57.968557
968557
Rank 2 terminating now!
Rank 0 terminating now!
Rank 1 terminating now!
```

2. Ring Communication with MPI (120 Points)

In the materials, you can find the serial program `circle.c`. Modify it with the help of MPI in a way that the parallel version will behave in the following ways:

A one-dimensional array of the length `N` is distributed as evenly as possible over `nprocs` and is initialized with random values (in the range 0–24). `N` shall be given as the only command line argument of the program. The values shall be output according to their ordering in the array. The processes shall create a ring, where every process communicates with its predecessor and successor (for example, rank n with the ranks $n-1$ and $n+1$). The last process will communicate with rank `0` and vice versa. In the `cirlce` function, the processes will exchange their data. In each loop iteration, each process will send its data to its successor and will receive the data from its predecessor. This shall happen until the last process has the first array element of the first process before the loop was entered in the beginning of its array (see Figure 1). If this is the case, all sub-arrays will be output in order of process ranks and the program terminates.

Take into account the following requirements:

- The program shall work with an arbitrary amount of processes and an arbitrary array length. The input must be checked for potential errors and the program shall react appropriately.
- At **no point** is one process allowed to store the complete array in its memory.
- The termination is not allowed to be dependent on a specific number of loop iterations, but must be checking for the correct termination condition described above. Take into account that this condition can happen earlier than after `nprocs - 1` iterations, since the same value can occur more than once (see Figure 1).
- Make sure output is ordered correctly.
- Replace uses of `MPI_Send` with `MPI_Ssend` to make sure that the program will work with arbitrary message sizes. Other operations such as `MPI_Isend` or `MPI_Sendrecv` are of course also allowed.

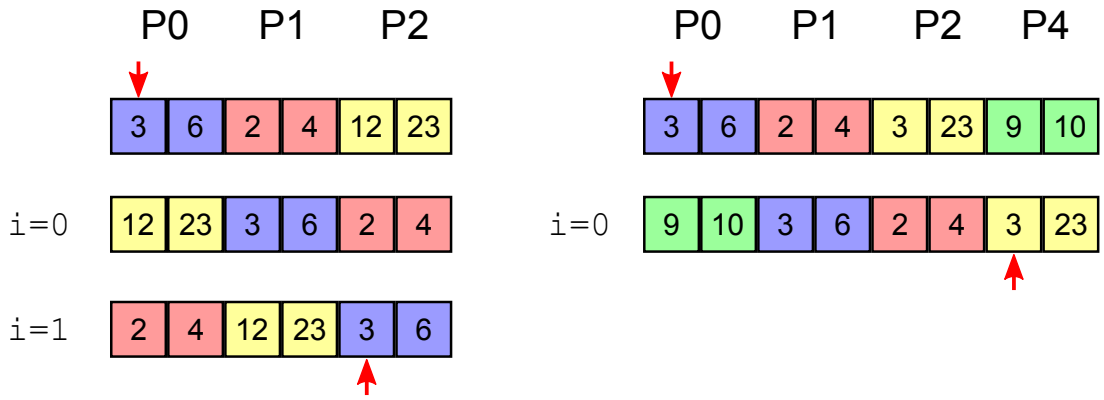

Figure 1: Examples of possible program executions, from initialization until termination (PX stand for process X, i=n for the nth iteration)

Submission

We will count your last commit on the `main` branch of your repository before the exercise deadline as your submission. In the root directory of the repository, we expect a `PP-2025-Exercise-06-Materials` directory with the following contents:

- A file `group.md` with your group members (one per line) in the following format:

  ```
  Erika Musterfrau <erika.musterfrau@example.com>

  Max Mustermann <max.mustermann@example.com>
  ```

- The code of the `timempi` program in the sub-directory `timempi` (Task 1)
  - All code that makes your program (`timempi.c`); well documented
  - A `Makefile` where `make timempi`, `make clean` and `make` act as expected.
- The modified code of the `circle` program in the `circle` sub-directory (Task 2)
  - All code that makes your program (`circle.c`); well documented
  - A `Makefile` where `make circle`, `make circle` and `make` act as expected.