Parallel computing 23.2.2018
Exercise 7

Submit the X2 task by Fri 23.2. 09:00 using a separate www-form as instructed in a separate email. Submit each of the other solutions separately to Moodle by 23.2. 09:00 (1 hour before exercises). Take skeletons from course www-page.

31. Parallelize the prime finding algorithm of X1 using MPI. The input (interval) is given for the first process, and at the end, the full result (array) should be at the first process.

32. Let us consider a voting problem where each of P processes have k integer values of $[0..P-1]$. Goal is to find which integer value has most occurrences and deliver that to all processes. Write an MPI program using the skeleton at www-page. You can assume that there is a single winning value. Hint: collective communication.

33. Modify your previous program for a case where a draw between two or more values is decided so that the smallest value wins. Hint: new communication group.

The following X2 exercise replaces standard exercises 34 and 35.

The answers to X-exercises have to be unique for every student. No copies of the same answer are allowed. The answer has to be sent via email by Friday 09:00. You will receive an acknowledgment upon successful processing. Answers will be graded. The answer must also contain a short self-evaluation in which you describe whether the program works, nearly works, or does not probably work; how efficient it is, what speedups you got, etc. A correct and proper self-evaluation is worth one point (in case of a proper answer). As the answer is a C program, the self-evaluation must be included in the comments of the program.

Send your solution using a www-form, address and credentials of which you got by email. The solution should be a compilable C source code file If you do not receive an acknowledgment, or you receive compilation errors in the acknowledgement, send it again correctly.

Take a skeleton from course www-page. Do not change the header (name, parameters) of the X-task method(s). Please make sure that the program is compilable as such, i.e., have the whole answer in the same file.

X2. Write a parallel C implementation of the longest common substring -task using MPI. Take the sequential implementation from www-page, parallelize it using MPI, test it using a multiprocessor/core system, and record the speedup you got. The input is two strings A and B and result is the starting index in both string and the length of the common substring. If there are several common substrings of the same length, returning any of the is enough.