UNIVERSITÄT SIEGEN

Fakultät IV
Betriebssysteme und verteilte Systeme
Prof. Dr. rer. nat. Roland Wismüller

Excercise Sheet 7

(To be processed until 21.01.)

Lecture Parallel Processing
Winter Term 2024/25

Exercise 1: Point to point communication with MPI

Modify the code in `point2point.cpp` (in the archive `u07eFiles.zip`[1] on the lecture's web page), such that at the indicated position

- Process 1 sends the contents of variable `myval` to process 3, that receives it in `myval`,
- Process 2 sends the contents of variable `myval` to process 0, that receives it in `myval`.

Thus, with the given initialization, the value printed by process 3 at the end of the program should be 1, the value printed by process 0 should be 2.

Note that this exercise checks for MPI skills in a way comparable to an exercise in the exam!

Exercise 2: Parallelization of a map operation using MPI

Parallelize the code in `map.cpp` in the archive `u07eFiles.zip`[2] on the lecture's web page! Note that only process 0 should initialize the array `x`, and only process 0 should check the result at the end. The application of the function `complex_fct()` on the input array `x` should be distributed to all processes. Note that `complex_fct()` is a pure function that does not have any side effects.

For simplicity, you can assume that the array size can be evenly divided by the number of processes (with the given size of 10000 elements, this is the case for 1, 2, 4, 8, and 16 processes). Do not modify any code in `fcts.cpp`!

Note that this exercise checks for MPI skills in a way comparable to an exercise in the exam!

Exercise 3: Parallelization of a simple optimization code with MPI (Compulsory Exercise, Weight 2! Submit until Tuesday, January 21st, 10:00 via moodle)

In this exercise, we revisit the optimization code of Exercise 3 on Exercise Sheet 3 and Exercise 1 on Exercise Sheet 5. The code is given again in the archive `u07eFiles.zip`[3] on the lecture's web page.

a) Parallelize the code in `optimize1.cpp`. Your program should always use four MPI processes, where each process computes one configuration. At the end, you should compute the final minimum using `MPI_Reduce()`.

b) Parallelize the code in `optimize2.cpp`. Since the execution time of `computeCost()` shows a high variation, use the manager/worker model (see Sect. 2.7.2 of the lecture slides).

Process 0 should be the manager, that sends tasks descriptions to the workers and receives their result. In this code, the task description simply is the number of the configuration to be evaluated (i.e., the argument passed to `computeCost()`). Once a result has been received from a worker, the next task description is sent to that worker. In case the manager is running out of tasks (i.e., all 100 tasks have already been sent), this task description should contain a special value (e.g., a number ≥ 100) that instructs the worker to terminate.

[1] `http://www.bs.informatik.uni-siegen.de/web/wismueller/vl/pv/u07eFiles.zip`
[2] `http://www.bs.informatik.uni-siegen.de/web/wismueller/vl/pv/u07eFiles.zip`
[3] `http://www.bs.informatik.uni-siegen.de/web/wismueller/vl/pv/u07eFiles.zip`

In both files `optimize1.cpp` and `optimize2.cpp`, the code to initialize MPI is already given. Do not modify any code in `functions.cpp`!

Exercise 4: Numerical integration using MPI

Parallelize the code in `integrate.cpp` (in the archive `u07eFiles.zip`[4] on the lecture's web page, identical to that of Exercise 4 on Exercise Sheet 3) with MPI using a reduction (`MPI_Reduce()`). The initialization of MPI is already given. Measure the speedup with different values for the number of intervals and different numbers of processes. Interpret your results.

[4]`http://www.bs.informatik.uni-siegen.de/web/wismueller/vl/pv/u07eFiles.zip`