

Excercise Sheet 8

(To be processed until 03.02.)

Lecture Parallel Processing Winter Term 2025/26

Compulsory Exercise 1 (weight 3): Parallelization of the Jacobi method using MPI Submit until Tuesday, February 03rd, 10:00 via moodle

The sequential code in the files `heat.cpp` and `solver-jacobi.cpp` (in the archive [u08eFiles.zip](#)¹ on the lecture's web page) shall be parallelized using MPI (the code is identical to that of Exercise 1 on [Exercise Sheet 3](#)).

- a) Parallelize the `main()` function and the `solver()` function.

For simplicity, the file output of the matrix (function `write_matrix()`) should be restricted to the local part of the matrix on process 0 (i.e., only the process with rank 0 should call `write_matrix()`)².

However, make sure that the control values are output correctly at the end of `main()`. In the case of correct parallelization, the output values must correspond exactly to those of the sequential version!

You can distribute the matrix in one dimension only (strip-wise, i.e., contiguous blocks of rows, see Sect. 4.8 of the lecture slides) or in both dimensions (see last slide of Sect. 4.1 of the lecture). The strip-wise distribution is (much) simpler, but the block-wise one (with larger process numbers) possibly more efficiently. Ideally, your partitioning should work for all matrix sizes and process numbers (see Sect. 4.8 of the lecture slides and the example code [vecmult3.cpp](#)³).

Try to optimize your program as much as possible, e.g., by using non-blocking receive operations.

Documents to be submitted: modified C++ files 'heat.cpp' and 'solver-jacobi.cpp'.

- b) Measure how much time your program needs. Try different values for the matrix size (e.g.: 1000, 2000, 5000 and 10000) and measure the speedup with different numbers of processes (2 to 16, possibly even more).

Use Scalasca for a more detailed performance analysis. How much time is spent in MPI point-to-point communication, how much time is spent in collective communication?

Document to be submitted: PDF file with speedup results and a screen dump of Scalasca showing the communication time.

Exercise 2: Parallelization of the Gauss/Seidel method using MPI (For Motivated Students)

In this exercise, you shall parallelize the code for the Gauss/Seidel method, as provided in the file `solver-gauss.cpp` (in the archive [u08eFiles.zip](#)⁴ on the lecture's web page) using MPI (the code is identical to that of Exercise 2 on [Exercise Sheet 3](#)).

For simplicity, the function `solver()` in this version performs a fixed number of iterations, calculated in advance from the precision parameter. This allows pipelined parallelization (where in contrast to the OpenMP parallelization using diagonal traversal, the `i` and `j` loops are not rewritten, see Sect. 4.8 of the lecture slides). For example, process 0 sends its last row to process 1 after each iteration, and then waits for the first row of process 1. Process 1 can (and must) send this row immediately after its calculation.

¹<http://www.bs.informatik.uni-siegen.de/web/wismueller/v1/pv/u08eFiles.zip>

²This is sufficient to check the correct working of the program.

³<https://www.bs.informatik.uni-siegen.de/web/wismueller/v1/gen/pv/04Code.zip>

⁴<http://www.bs.informatik.uni-siegen.de/web/wismueller/v1/pv/u08eFiles.zip>

Before you start programming, first consider **exactly** which communications are necessary and how the sequence of the calculations and communications should look like! Also note that you may need to add additional parameters to the interface of the `solver()` function.

Measure how much time your program needs. Try different values for the matrix size (e.g., 1000, 2000, 5000 and 10000) and measure the speedup with different numbers (2 to 19) of processes.

If necessary, do a more detailed performance analysis (if possible, using Scalasca) and try to optimize your program as much as possible, e.g., by using non-blocking receive operations.

Exercise 3: Acquaint yourself with the electronic exam system

The exam for this module will be conducted **electronically** using the Q-Exam system. The **computers (laptops) will be provided by the university**.

To get acquainted with the system, please **have a look at the provided demo exam**⁵ (select the exam named “Demo-Prüfung Parallelverarbeitung”) and the **screen casts provided in the moodle course**⁶! If you have questions or problems, please contact the e-assessment support team (e-klausuren@uni-siegen.de). You can also ask questions immediately before the exam starts.

Please notice that the laptops provide a **German keyboard**. The '#' key is located left of the 'Enter' key, '{', '}', '[', and ']' can be entered by pressing the 'AltGr' key (right of the 'Space' key) together with '7', '8', '9', or '0'. The 'Ctrl' key is named 'Strg'. Please acquaint yourself with that layout! An **accurate image** can be found on [wikipedia](https://en.wikipedia.org/wiki/German_keyboard_layout)⁷. There is also **an interactive simulation**⁸, which, however, differs slightly from the laptop keyboards.

⁵<https://uni-siegen.q-examiner.com/>

⁶<https://moodle.uni-siegen.de/course/view.php?id=23366#section-6>

⁷https://en.wikipedia.org/wiki/German_keyboard_layout

⁸<https://www.branah.com/german>