

Excercise Sheet 3

(To be processed until 02.12.)

Lecture Parallel Processing Winter Term 2025/26

Exercise 1 (Compulsory, weight 2): Parallelization of the Jacobi method using OpenMP Submit until Tuesday, December 02nd, 10:00 via moodle

In this exercise we consider the Jacobi method for the iterative solution of a boundary value problem. In Jacobi iteration, a new matrix of values is computed from a given one by assigning to each inner element of the new matrix the mean value of the four neighboring elements of the old matrix. The iteration is repeated until the maximum change of an element is below a predefined limit.

First, have a look at the code in the archive [u03eFiles.zip](#)¹ on the lecture's web page and understand, how it works. The `main()` function parses the command line arguments, allocates and initializes the matrix, calls the solver (in `solver-jacobi.cpp`) and prints some verification and timing values at the end. For performance reasons, the solver does not copy the matrix, but rather swaps the pointers of the main and the auxiliary array. Only at the very end, the auxiliary array is copied back into the main array, if necessary.

The program is invoked like this:

```
./heat <matrix-size> <epsilon>
```

where the first argument is the size of the matrix (range: 3 ... 22000) and the second argument specifies the limit for the maximum change of an element in an iteration (try values between 0.1 and 0.000001).

- a) Parallelize the sequential code for the Jacobi iteration in the file `solver-jacobi.cpp` using OpenMP directives. Use an OpenMP reduction to compute the maximum change in the array (variable `diff`).² Try to optimize your program as far as possible.

Test the program with different matrix sizes and thread counts. At the end, the program prints the values of several matrix elements. Note that in your parallel version these results must be **exactly** the same (that is, in **all** decimal places) as in the sequential program!

For further verification, you can also view the output (in the file `Matrix.txt`) graphically, using the Java program `ViewMatrix`. This output is, however, only created with a matrix size up to 500.

Document to be submitted: modified C++ file `'solver-jacobi.cpp'`.

- b) Acquaint yourself with the hardware and software infrastructure of the computer lab H-A 4111 by having a look at Sect. 1 of the [tutorial slides](#) and the [accompanying screen cast](#).
- c) In the lab H-A 4111, measure the achieved speedup for different matrix sizes and different numbers of threads (c.f. lecture slides 257 ff). Use Scalasca (see Sect. 2.4 of the [tutorial slides](#) and the [accompanying screen cast](#)) to determine the OpenMP overhead for a small matrix size (1000) and a large one (20000). What is the main source of overhead for small matrices? When using 20 threads in the lab H-A 4111, you will see a significant load imbalance for large matrices. What is the reason for this?

Documents to be submitted: PDF file showing

- results of the speedup measurements,
- screen dump of Scalasca showing the reason and location of the main OpenMP overhead for small matrices,
- screen dump of Scalasca showing the reason and location of the main OpenMP overhead for large matrices,
- answers to the questions.

¹<http://www.bs.informatik.uni-siegen.de/web/wismueller/v1/pv/u03eFiles.zip>

²Reductions of this form are directly supported by OpenMP since version 4.0.

Exercise 2: Pipelined parallelization of the Gauss/Seidel method using OpenMP (For Motivated Students)

In order to enable an exact parallelization of the Gauss/Seidel method, the given solver does not iterate until the maximum change is below a threshold, but instead estimates the necessary number of iterations in advance. If the number of iterations is known in advance, a parallelization in a pipeline-like manner is possible (see Sect. 3.4 of the lecture slides). For the parallel execution of the i loop, a synchronization must be provided which ensures the following conditions (induced by the data dependences): before the i -th row is computed in iteration k ,

1. the $(i-1)$ -th row in iteration k and
2. the $(i+1)$ -th row in iteration $k-1$

must have been computed. Since OpenMP version 4.5, the `ordered` directive supports a `depend` clause which enables to specify a synchronisation safeguarding the specified dependencies (see the note at the end of Section 3.3.4 of the lecture slides).

Parallelize the Gauss/Seidel method (see the code in the archive [u03eFiles.zip](#)³ on the lecture's web page) using the above synchronization.

Conduct a performance analysis (if possible, using Scalasca) and try to optimize your program further.

³<http://www.bs.informatik.uni-siegen.de/web/wismueller/v1/pv/u03eFiles.zip>