SciComp and FEM in WS25

Exercise 5: shared memory parallelization using OpenMP

Deadline: Dec 2, 2025, 20:00

Status: Wednesday 12th November, 2025, 12:42

Supervisor: Prof.Dr. G. Haase, gundolf.haase@uni-graz.at

We propose to read the OpenMP summary[1] as well as the OpenMP tutorial from LLNL[2]. See also this guide[3] by Joel Yliluoma and slides, p.25[4] by Annika Hagemeier.

Timing: Take care to use a wall clock timer, e.g., OMP timing routine `omp_get_wtime()` or system_clock[5] or use `tic()` and `toc()` from the header file *timing.h*, see download[6] and docu[7].

1. Download[8] the template for the inner product of vectors (example II-A). (3 Pkt.)
   - Compile and run it.
   - Try several schedule types and junk sizes in *mylib.cpp:13*, see §4.1 and §2.7.1 in the OpenMP specifications.
   - Calculate the speedup for different number of cores (incl. hyperthreading) Use function `omp_set_num_threads(tn)` in your main function or call `export OMP_NUM_THREADS=tn` from the shell in order to run the code on `tn` parallel threads.
   - Try `omp_get_wtime()`, `omp_get_num_procs()` and `omp_in_parallel()`.
   - Write a second function `scalar` using a parallel environment `#pragma omp parallel` without `for` in the `pragma` directive.
   - Write a function similar to function `reduction_vec(int n)` that appends the private vectors instead of adding them, see p. 74ff in slides by A. Hagemeier.
2. Parallelize task (B) (Data-IO; means and max/min of vector elements) from Exercise 1. Compare the run time of the OpenMP approach with the execution policies[9] in C++17. (3 Pkt.)
3. Parallelize your solution from example (F) (Goldbach: count [, pairs]) from Exercise 1. (3 Pkt.)
   Check for correctness!
4. Parallelize examples (B)-(D) from Exercise 2. (3 Pkt.)
   - Write in (A) also a parallel function that realizes the summation $s = \sum_{k=0}^{n-1} x_k$.
   - Compare the speedup of the sum as well as the inner product for various $n = 10^k$, $k \in [3, 8]$.
5. Copy your sequential Code for example (E) (code[10], docu[11]) from Exercise 3 and parallelize it (Linear algebra as well as FEM matrix computation/accumulation). (6 Pkt.)

[1] http://www.openmp.org/wp-content/uploads/OpenMP-4.0-C.pdf
[2] https://computing.llnl.gov/tutorials/openMP/
[3] https://bisqwit.iki.fi/story/howto/openmp/#IntroductionToOpenmpInC
[4] https://docplayer.org/19676777-Einfuehrung-in-openmp.html
[5] https://en.cppreference.com/w/cpp/chrono/system_clock
[6] http://imsc.uni-graz.at/haasegu/Lectures/Math2CPP/Examples/utils.zip
[7] http://imsc.uni-graz.at/haasegu/Lectures/Math2CPP/Examples/utils/html/timing_8h.html
[8] http://imsc.uni-graz.at/haasegu/Lectures/Math2CPP/Codes/shm/demo_skalar.zip
[9] https://www.bfilipek.com/2018/06/parstl-tests.html
[10] http://imsc.uni-graz.at/haasegu/Lectures/Math2CPP/Codes/shm/../seq/jacobi_oo_stl.zip
[11] http://imsc.uni-graz.at/haasegu/Lectures/Math2CPP/Codes/shm/../seq/jacobi_oo_stl/html

5*. Write an alternative matrix accumulation that avoids the `atomic` statements in the finite element loop.There are at least two opportunities to realize thsi accumulation: (6 Pkt.)

(a) Store the entries of all element matrices in a large vector such that all entries of $K_{i,j}$ are stored successively. The storage scheme will be somehow similar to an CRS storage. Afterwards, the accumulation for each $K_{i,j}$ will be done in parallel without locking.
The placements for the element matrix entries $K^{\tau}_{i,j}$ have to be calculated a priorily.

(b) Use a graph coloring such that all elements with the same coloring have no matrix entry $K_{i,j}$ (and no $K_{i,i}$) in common. Maybe the Boost Graph Library[12] is of use.
Then the old routines for matrix accumulation can be used with an outer loop on the colors.

Do a timing for all parts of your assembling approach, compare it with the timing for the parallel accumulation in 5.

This document might be extended by further advices, links, etc.

[12] `https://www.boost.org/doc/libs/latest/libs/graph/doc/quick_tour.html`