

High Performance Computing (Master) in WS24

Exercise 3: shared memory parallelization using OpenMP

Deadline: Jan 7, 2025, 16:00

Status:

Wednesday 20th November, 2024, 07:56

Supervisor: Prof.Dr. G. Haase,

gundolf.haase@uni-graz.at

We propose to read the OpenMP summary¹ as well as the OpenMP tutorial from LLNL². See also this guide³ by Joel Yliluoma and slides, p.25⁴ by Annika Hagemeyer.

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 download⁶ and docu⁷.

1. Download⁸ the template for the inner product of vectors (example II-A). (4 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. Hagemeyer.
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 policies⁹ in C++17. (4 Pkt.)
3. Parallelize example (F) (Goldbach: count [, pairs]) from Exercise 1. (4 Pkt.)
4. Parallelize examples (B)-(D) from Exercise 2. (4 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) (code¹⁰, docu¹¹) or (F) (code¹², docu¹³) (8 Pkt.) from Exercise 2 and parallelize it.

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>

⁵https://en.cppreference.com/w/cpp/chrono/system_clock

⁶<http://imsc.uni-graz.at/haasegu/Lectures/Math2CPP/Examples/utils.zip>

⁷http://imsc.uni-graz.at/haasegu/Lectures/Math2CPP/Examples/utils/html/timing_8h.html

⁸http://imsc.uni-graz.at/haasegu/Lectures/Math2CPP/Codes/shm/demo_skalar.zip

⁹<https://www.bfilipek.com/2018/06/parstl-tests.html>

¹⁰http://imsc.uni-graz.at/haasegu/Lectures/Math2CPP/Codes/shm/./seq/jacobi_oo_stl.zip

¹¹http://imsc.uni-graz.at/haasegu/Lectures/Math2CPP/Codes/shm/./seq/jacobi_oo_stl/html

¹²http://imsc.uni-graz.at/haasegu/Progs/gh_hack.zip

¹³http://imsc.uni-graz.at/haasegu/Progs/gh_hack/html

5*. Write an alternative matrix accumulation that avoids the `atomic` statements in the element loop. There are at least two opportunities to realize this accumulation: (8 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 $K_{i,j}^T$ have to be calculated a priori.

(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 Library¹⁴ 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. _____

Wednesday 20th November, 2024

¹⁴https://www.boost.org/doc/libs/1_86_0/libs/graph/doc/quick_tour.html