Assignment 5: OpenMP - advanced

The purpose of this assignment is for you to learn more about

- the tasks construct of OpenMP,
- how parallel loops can be implemented with recursive tasks in OpenMP,
- how easy some parallel recursive algorithms are to write with OpenMP tasks.
- see complex parallelism structures in OpenMP

As usual all time measurements are to be performed on the cluster.

Activate OpenMP in GCC by passing `-fopenmp` to the compiler and linker. (Note that if you omit this parameter, the code will probably still compile. But its execution will be sequential.)

You can control the number of threads in OpenMP in two ways: the environment variable `OMP_NUM_THREADS` or by calling the `omp_set_num_threads` function.

When recursively decomposing the workload, the granularity of the decomposition is set by the size of the smallest task that will not be further decomposed. Traditionally, a threshold is used under which all tasks are processed sequentially.

1 Reduce

Question: Implement computing the sum of an array using the OpenMP task decomposition.
Question: What speedup do you achieve with 16 threads? (grade will depend on achieved speedup.)

2 Merge Sort

Question: Implement a parallel function using OpenMP tasks to perform merge sort on an array of integer.
Question: What speedup do you achieve with 16 threads? (grade will depend on achieved speedup.)

3 Longest Common Subsequence

Question: Implement a parallel version of the Longest Common Subsequence algorithm. Use the construct, granularity, parameters, etc. that you deem appropriate.
Question: What speedup do you achieve with 16 threads? (grade will depend on achieved speedup.)

4 Bubble Sort

Question: Implement a parallel version of BubbleSort algorithm. Use the construct, granularity, parameters, etc. that you deem appropriate.
Question: What speedup do you achieve with 16 threads? (grade will depend on achieved speedup.)