Assignment 6

Task: Stereo disparity OpenMP implementation

OpenMP implementation of stereo disparity algorithm (C/C++) and profiling

Task description

The algorithm for "Depth estimation based on Zero-mean Normalized Cross Correlation (**ZNCC**)" should be implemented in C/C++ using **CPU parallelization and multi-threading with OpenMP pragmas**. Profiling of the algorithm needs to be done and then compared with Assignment 5 pthread implementation.

Task Implementation: Stereo disparity OpenMP implementation

This task is dedicated to the implementation of the code in C/C++ in a parallel manner using more than one thread of execution and more than one processor core.Try implementing the code in a way that the most of the parameters (such as e.g. the window size) can be easily changed. This implementation will be used by you in future assignments to understand where are the opportunities for parallelization in the proposed algorithm.

Run the C/C++ code on multiple CPU cores using OpenMP pragma directives. OpenMP is an API supported by multiple vendors programming languages and operating systems that gives the programmer a very simple yet flexible interfaces for the parallelization of applications: https://en.wikipedia.org/wiki/OpenMP

To use OpenMP, the compiler must link against the library and optionally a header file (omp.h) could be included in the code. The simplest way of using OpenMP is to include a #pragma before the sections of the code that can be parallelized (for example a for loop) and let the compiler create the threads for you. The most advanced use of OpenMP requires using some functions exposed by the library -- in other words, some rewriting -- but it is easy to get some results right away by simply decorating your sequential code.

The main idea behind this task is to parallelize the C/C++ code using multi-core CPU and threads and realize their execution times. This is currently explored as an alternative to pthreads/C++ threads (implemented in assignment 5). In addition to multi-threading, vectorization that uses the SIMD units of the processor (for example Intel SSE instructions) can be utilized and will speed up computation further. This can also be explored, if the student is interested but not any compulsory part for this exercise.

You must have completed C/C++ sequential implementation of stereo disparity implementation in assignment 4. The task here is to add this thread functionality to the same C/C++ implementation and realize if the execution times has got any better. Remember to save your sequential C/C++ implementation as well sepearately before adding this thread funcitonality.

For the **profiling** part, measure the total execution time of your implementation (measure each of the important modular functions separately as well, if needed) using some of the tools like QueryPerformanceCounter-function in Windows or gettimeofday-function in Linux. Check the processor load as well. Compare the profiling data of OpenMP implementation with that of pthreads/C++ boost threads implementation and report your observations.

Save the output images in a similar manner to C/C++ implementation and include them under one folder

Expected result:

A working version of the implementation, a brief report (max2-3 pages), saved output images all together in the form of a compressed folder (.zip file)

The report should contain about the task solved, brief description of your implementation, comparison of profiling information for pthread and OpenMP implementations and the final screenshot of your outputs asked to be displayed under the assignment.