University of Basel

Lecturer:	Computer Lab. U1.001	Assistants:	http://informatik.unibas.ch/fs2018/lecture-high-performance-computing/
Prof. Dr. Florina M. Ciorba	Spiegelgasse 1, CH-4051 Basel	Aurélien Cavelan, Ph.D. Danilo Guerrera, M.Sc. Ahmed Eleliemy, M.Sc. Ali Mohammed, M.Sc.	

High Performance Computing (17164-02) **Spring Semester 2018**

Assignment 2: OpenMP *(20 Points)*

Starting Date: March 15, 2018
Deadline: March 28, 2018 - 23:59:59

Objectives:

1. Parallelize the execution of the programs using OpenMP.

2. Explore the effect of the scheduling in achieving load balance.

3. Implement parallel programs using OpenMP tasks.

1 Matrix Multiply in OpenMP *(6 Points)*

Starting from the naive implementation of the matrix multiply program provided in *T1.c*, write a parallel version using OpenMP.

a. Optimize and parallelize the source code for the naive multiply. *(2 Points)*

b. Optimize and parallelize the source code for blocked multiply. *(3 Points)*

c. Submit a strong scalability plot (measuring execution time) for the both methods from (a) and (b) with `numThreads = 1, 2, 4, 8, 16`, and `size = 4000`. *(1 Point)*

2 Scheduling in OpenMP *(6 Points)*

Performance of parallel applications can suffer due to load imbalance. In this task, you are required to parallelize and optimize the performance of matrix multiplication provided in *T2.c*. To introduce load imbalance, one of the multiplied matrices will be an upper triangle matrix. Investigate the effect of load imbalance on the performance and how to improve the performance using *schedule* clause in OpenMP.

a. Implement the `fill_matrix_upper` function to fill the upper triangular part of the matrix with double values and the lower triangular part with zeros. *(1 Point)*

b. Parallelize the naive and the blocked versions of the matrix multiplication as in Task 1. Report the performance of both versions using the upper triangular matrices for `numThreads = 2, 4, 8, 16`, and `size = 4000`. *(1 Point)*

c. Using the *schedule* clause, try to achieve load balance to enhance the performance of both versions of the matrix multiplication. *(2 Points)*

d. Submit a strong scalability plot (measuring execution time) for both the methods with and without using load balancing with `numThreads = 2, 4, 8, 16`, and `size = 4000`. *(2 Points)*

3 Task programming in OpenMP *(8 Points)*

In this task, you are required to parallelize the implementation of the matrix multiplication using tasks. The sequential unoptimized code is provided in *T3.c*.

a. Use the `taskloop` construct to parallelize the naive and the blocked matrix multiplication using *OpenMP tasks*. *(4 Points)*

b. Submit a strong scalability plot (measuring execution time) for both the methods with `numThreads = 2, 4, 8, 16`, and `size = 4000`, set `num_tasks` to the number of threads. *(2 Points)*

c. Discuss and show how the parallelization of the matrix multiplication could benefit from task programming, especially in the case of sparse or upper/lower triangular matrices. *(2 Points)*

Please make sure that your optimizations do not affect the correctness of the results.
You should use the miniHPC cluster to obtain all the results.
The delivered solution should be in one tar file.