# Simulation and High-Performance Computing

University of Kiel, September 28th to Oktober 9th, 2020.

## Exercises: Day Six

CSC Computational Science Center

## Main Exercises: Applications of Lecture 11

### a) Basic OpenMP Concepts, Work-Sharing

In High-Performance-Computing, it is mandatory that even basic operations on data structures completely exploit the capabilities of the underlying hardware. This especially means that the code has to make use of *multi-threading*. That term refers to taking advantage of multiple processor cores in a machine by organizing a program into several *threads* which can then be worked on by different cores - ideally, but not always, in parallel. Amongst some others, the most prominent multi-threading technique is called *OpenMP*.

We have seen that in order to solve partial differential equations with the finite difference method, we need to employ grid functions as our basic data structures. To achieve a convenient workflow, we had implemented several useful operations with those grid functions.

Your task is to **implement** parallel versions of the functions `clear_gridfunc2d`, `dot_gridfunc2d` and `addeval_5point_stencil_gridfunc2d` in the file `exercise_omp_gridfunc2d.c`.

To obtain a parallel version of `clear_gridfunc2d`, try two different approaches: Firstly, use explicit worksharing with the API-functions `omp_get_thread_num` and `omp_get_num_threads`. Secondly, take the standard approach with the OpenMP directive `#pragma omp for`.

When implementing `dot_gridfunc2d` you need some kind of *reduce* operation. You can either use a special OpenMP clause that gets added to `for`-pragma or you can do it manually. Once again, try both versions.

Of course, the matrix-vector product with the discrete Laplacian is crucial for the overall performance of methods such as time-stepping schemes. Try to parallelize that matrix-vector product.

### b) Monte Carlo Methods and OpenMP

A well-known approach for solving complicated integrals or similar problems consists of the so-called Monte Carlo methods. For a simple example, we want to look at computing the famous constant $\pi$:

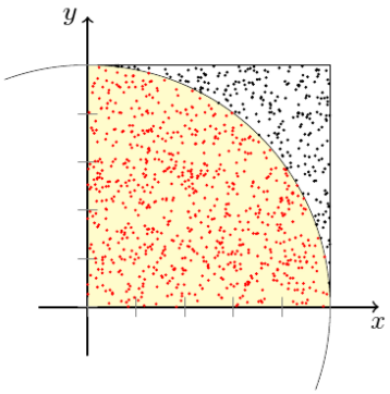


Figure 1: Computing $\pi$ using random numbers. https://de.wikipedia.org/wiki/Monte-Carlo-Simulation

As we can see in the picture, random points $x \in [0,1]^2$ are created in the unit square $[0,1]^2$ and then the function

$$p : [0,1]^2 \to \{0,1\}, \qquad p(x) = \begin{cases} 1 & : \|x\|_2 \leq 1 \\ 0 & : \text{otherwise} \end{cases}$$

tests, whether the points lie within the quarter circle or not.

Repeating this experiment $n$-times gives an approximation of $\pi$ via

$$\pi \approx 4 \frac{\sum_{i=0}^{n} p(x_i)}{n}.$$

In the file `exercise_montecarlo.c` you can find both a sequential version, as well as a parallel version using OpenMP. Find at least 3 problems/mistakes with the parallel implementation, correct them and `implement` and `test` the corrected parallel version.

## Additional Exercises: Applications of Lecture 12

### a) Blocking and Parallelization of UL-Factorization

As we have seen in week one of the block course, we can usually also find a *UL-factorization* of a given matrix $A \in \mathbb{R}^{n \times n}$ , i.e., we have $A = U\,L$, where $U \in \mathbb{R}^{n \times n}$ is an upper triangular and $L \in \mathbb{R}^{n \times n}$ is a lower triangular matrix.

We can also state this algorithm in terms of block matrix operations. This means that a parallelization with *OpenMP tasks* is possible as well.

**Implement** the blocked version as well as the blocked parallel version using tasks. You might need the functions:

- `rldecomp`: Compute the factorization $A = U\,L$ for a matrix block $A$.
- `block_rsolve`: Solve for $X$ in $U\,X = B$, where $U$ is an upper triangular matrix and $X$ and $B$ are general matrices.
- `block_lsolve_trans`: Solve for $X$ in $L^T\,X^T = B^T$, where $L$ is a lower triangular matrix and $X$ and $B$ are general matrices.
- `gemm`: Matrix-matrix-multiplication $C \leftarrow C + \alpha AB$.

**Test** your code for various matrix and block dimensions.

*Please provide feedback about the format/difficulty/length of the exercises so that we can adjust accordingly. Contact us via Email or on the OpenOLAT forum.*