Prof. Dr. Christoph Pflaum
Benjamin Mann

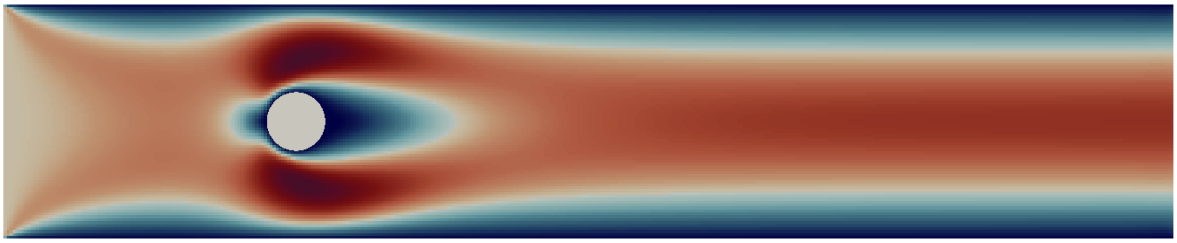
Summer Term
2022

Simulation and Scientific Computing 2
Assignment 4

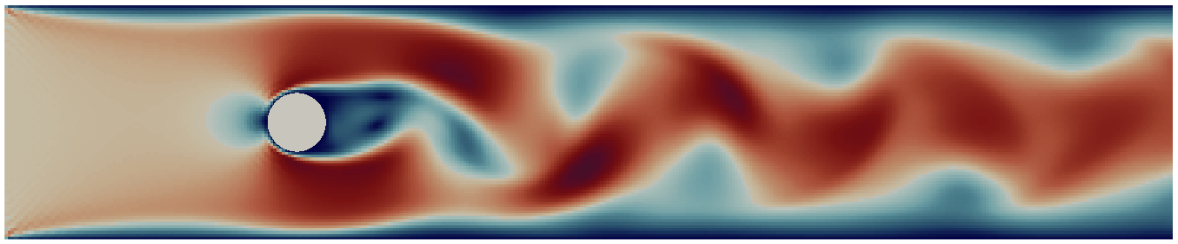
Your very own wind tunnel simulator using the Lattice Boltzmann method

1 Introduction

You have learned in the lecture that the *Lattice Boltzmann method (LBM)* characterizes a class of cellular automata which can be used to solve problems in computational fluid dynamics (CFD). The goal of this programming project is to implement the Lattice Boltzmann method, and to set up a virtual, two-dimensional wind tunnel where you can place obstacles and study the flow around it.



(a) $Re = 40$ after 50000 time steps.



(b) $Re = 500$ after 150000 time steps.

Figure 1: Circular object placed inside the wind tunnel at different Reynolds numbers.

Our virtual wind tunnel will feature a flow between solid walls where it encounters an obstacle. The flow is driven by a constant inflow velocity and exits the tunnel via an appropriate outflow boundary. Depending on the value of the characteristic number of the wind tunnel, the Reynolds number Re, we can observe different flow conditions and structures.

2 The Lattice Boltzmann method

The basic LBM data structure is a two dimensional grid of cells where 9 values, denoted as f_q, have to be stored in each cell. Each of these 9 values is associated with a specific direction, which is labeled according to the points of the compass as shown in Figure 2.

$$q \in \{C, N, S, W, E, NW, NE, SW, SE\}$$

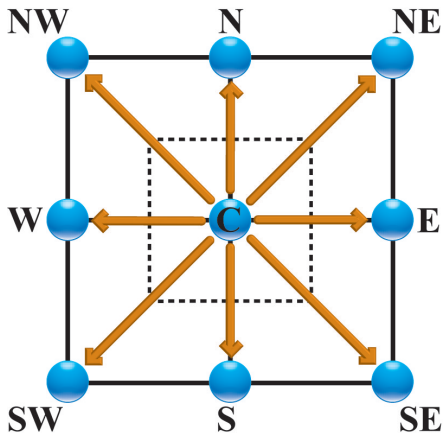


Figure 2: Lattice cell.

Physically, these 9 values f_q correspond to particle distribution functions (PDF) i.e. describe the number of "virtual" particles in that cell moving in direction q.

The directions can be represented by the vectors $\vec{c}_q$:

$$\vec{c}_C = (0,0), \quad \vec{c}_N = (0,1), \quad \vec{c}_W = (-1,0), \quad \vec{c}_{SE} = (1,-1), \quad ...$$

All local macroscopic flow quantities, like density and velocity, are given as functions (moments) of the local f_q:

$$\text{Density:} \quad \varrho(\vec{x},t) = \sum_q f_q(\vec{x},t) \tag{1}$$

$$\text{Velocity:} \quad \vec{u}(\vec{x},t) = \sum_q f_q(\vec{x},t)\vec{c}_q \tag{2}$$

Note, that we make use of the incompressible formulation of LBM and thus assume $\varrho = 1$ in many cases.

The lattice Boltzmann algorithm itself can be separated into two parts: a *streaming* and a *collide* step.

2.1 Stream Step

In the streaming step, all particle distribution functions are propagated (copied) to the next cell (Fig. 3), e.g. the north value of the current cell is copied to the north value of the northern neighbor cell. This implementation of the stream step is also called *push*. From the exercises, you know that there is

also the *pull* variant which is usually preferred since it will facilitate the boundary treatment (see later on). In the pull variant, you copy the required values from the neighbor cells into the current cell, i.e. pulling it to you instead of pushing it to your neighbor. During the stream operation, values in neighboring cells are overwritten. Thus, two grids are needed, typically referred to as `source` and `destination` grid. During the stream step, the values from the source grid are read and written to the destination grid. Afterwards, the data pointers are swapped.

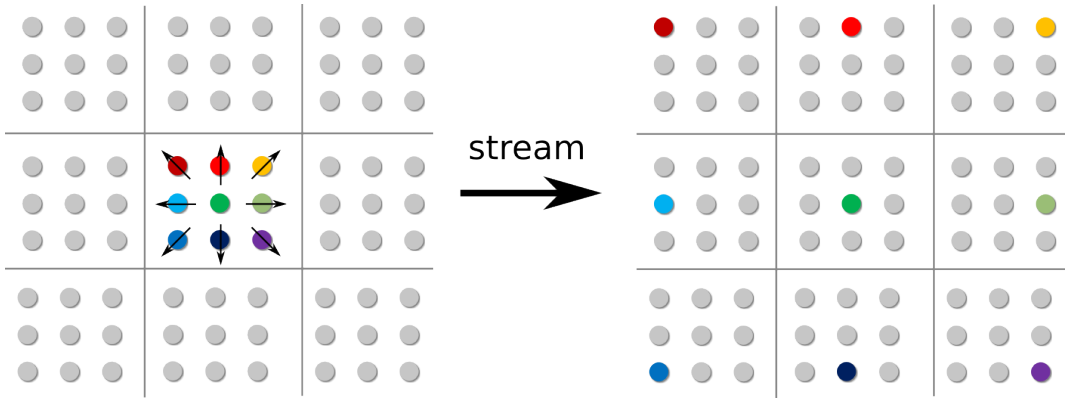


Figure 3: Regular stream (push) step in inner part of domain.

2.2 Collide Step

The collide step is a purely cell-local operation, i.e. each cell can be treated independently. It models the collision of particles by relaxing them to a local thermodynamic equilibrium. First, the density and velocity of the cell are computed according to the equations given above. Then, an equilibrium distribution f_q^{eq} is calculated for each direction:

$$f_q^{eq}(\vec{x},t) = f_q^{eq}(\varrho,\vec{u}) = w_q \left(\varrho + 3\vec{c}_q \cdot \vec{u} + \frac{9}{2}(\vec{c}_q \cdot \vec{u})^2 - \frac{3\vec{u}\cdot\vec{u}}{2} \right), \tag{3}$$

with the weighting constants:

$$\begin{aligned} w_q &:= 4/9 \quad \text{for} \quad q = C \\ w_q &:= 1/9 \quad \text{for} \quad q = N, S, W, E \\ w_q &:= 1/36 \quad \text{for} \quad q = NW, NE, SW, SE. \end{aligned} \tag{4}$$

We use a single relaxation time (SRT) model, where f_q is relaxed towards the equilibrium distribution f_q^{eq} with the relaxation time $\tau \in (\frac{1}{2}, \infty)$. Thus, the following update rule has to be applied in all fluid cells for all f_q:

$$f_q(\vec{x},t) \leftarrow f_q(\vec{x},t) - \tfrac{1}{\tau}(f_q(\vec{x},t) - f_q^{eq}(\vec{x},t)) \tag{5}$$

Note, that the relaxation time defines the kinematic viscosity ν of the simulated fluid via

$$\nu = \tfrac{1}{3}\left(\tau - \tfrac{1}{2}\right). \tag{6}$$

2.3 Boundary Handling

Special care has to be taken at the boundary of the domain and for the obstacle. In the wind tunnel setup, we encounter three different types of boundary conditions: no-slip, velocity, and density boundaries. To simplify the handling of the boundaries, we will have a separate field, called *flag field*, which indicates in each cell the type of the cell with the help of an integer value. We use the convention from Table 1.

cell type	integer value in flag field
fluid cell	0
no-slip boundary cell	1
velocity boundary cell	2
density boundary cell	3

Table 1: Numbering convention for the integer stored in the flag field to indicate the cell type.

- At the north and south boundaries, we apply reflection (no-slip) boundary conditions which enforce zero velocity at the wall. The update rule for these boundary cells is shown in Fig. 4 and is formally given as:

$$f_{\bar{q}}(\vec{x}, t + \Delta t) = f_q(\vec{x}, t), \tag{7}$$

where $\bar{q}$ is the opposite lattice direction of q, such that $\vec{c}_{\bar{q}} = -\vec{c}_q$. In this formula, the cell $\vec{x}$ is located next to a boundary, and q is a direction from the fluid cell to the boundary, see Fig. 4.

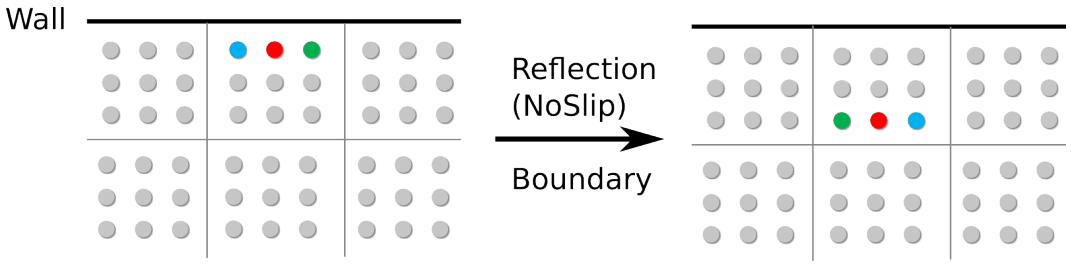


Figure 4: Streaming for reflection (no-slip) boundaries

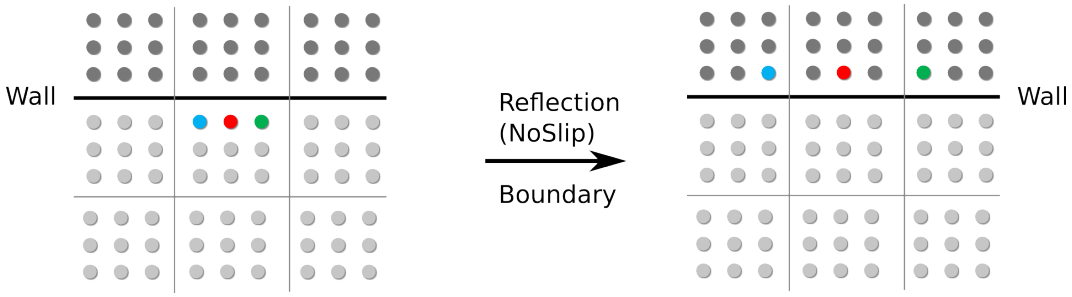


Figure 5: Reflection (no-slip) boundary with helper cells

To simplify the boundary handling, we extend our domain by one row of helper cells in each direction and set up the values in these helper cells before streaming such that the regular (pull)

streaming operation in the inner part of the domain leads to correct no-slip boundaries (Fig. 5). We do the same for all upcoming boundary conditions.

- At the west boundary, i.e. the inflow, we apply velocity boundary conditions, again with the help of an additional layer of cells. They essentially work like the reflecting boundary conditions from before but add a contribution related to the desired inflow velocity $\vec{u}_{in}$:

$$f_{\bar{q}}(\vec{x}, t + \Delta t) = f_q(\vec{x}, t) - 6 w_q \vec{c}_q \cdot \vec{u}_{in}. \tag{8}$$

- At the east boundary, we will have the outflow. Since we do not know the velocity there, we apply an outflow boundary condition that sets the density to a fixed value ϱ_{out} here and allows for varying velocities.

$$f_{\bar{q}}(\vec{x}, t + \Delta t) = -f_q(\vec{x}, t) + 2 w_q \left(\varrho_{out} + \frac{9}{2} (\vec{c}_q \cdot \vec{u})^2 - \frac{3}{2} (\vec{u} \cdot \vec{u}) \right), \tag{9}$$

  with the velocity $\vec{u} = \vec{u}(\vec{x})$. We will always set $\varrho_{out} = 1$ in our simulations.

- Up to now, we have only specified the wind tunnel itself. For the obstacle, we use a circle of diameter D that we place inside the tunnel at position (S_x, S_y). Since it is a stationary obstacle, we consider all lattice cells with cell centers inside the circle also as no-slip cells. This leads to a stair-case approximation of the circular shape in the simulation but allows us to deal with all kinds of geometries easily.

2.4 Summary

Your LBM simulation should be carried out as follows:

Initial Setup

Initially all fluid cells should be set to equilibrium with a density of 1.0 and zero velocity:

$$f_q \leftarrow f_q^{eq}(1, (0, 0)) = w_q \tag{10}$$

Time Stepping

One time step of the lattice Boltzmann method consists of the following steps:

1. collide step (only fluid cells)
2. handle no-slip, velocity and density boundaries
3. pull stream step (only fluid cells)
4. (optional) write density/velocity as VTK

Domain setup

The complete layout of the simulation domain is given in Fig. 6, which is also a visualization of the flag field.

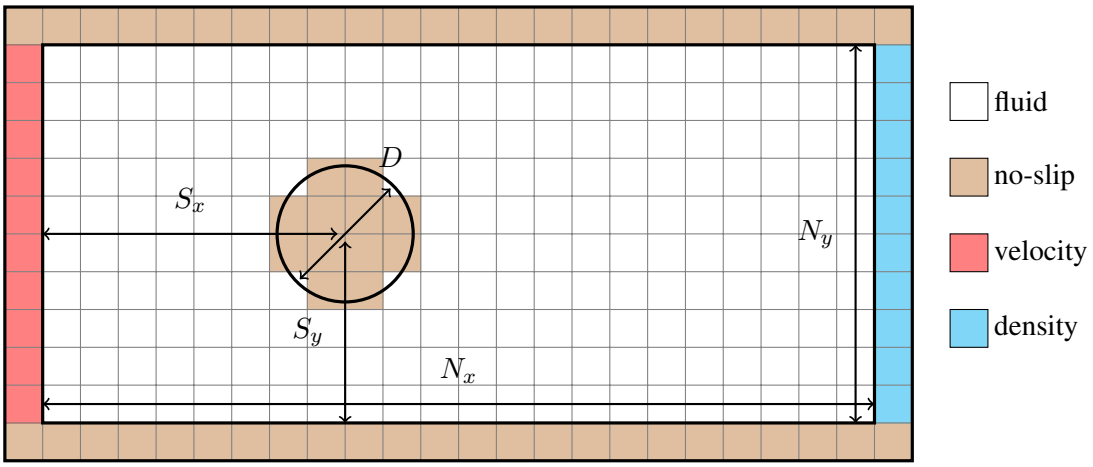


Figure 6: Grid layout: Your domain should consist of (N_x, N_y) cells and an additional boundary layer around it, which results in a grid of $(N_x + 2, N_y + 2)$ cells. The boundary layer consists of no-slip boundary cells (north and south), velocity boundary cells (west) with a velocity $\vec{u}_{in}$, and a density boundary (east) with $\varrho_{out} = 1$. A circle with diameter D is placed at position (S_x, S_y) (notice the position of the origin) and all cells with cell centers inside this circle are marked as no-slip boundaries.

3 Parameterization

3.1 Physical input

In order to specify the flow conditions, you heard in the exercise about the very famous Reynolds number Re. Here, we will define it as

$$Re = \frac{u_{in} N_y}{\nu}. \tag{11}$$

Thus, if you know the domain size (N_y) and the inflow velocity, you can calculate the relaxation time τ via Eq. (6) for a given Reynolds number. For increasing values of Re, you will notice that τ approaches the lower limit of $\frac{1}{2}$, making the simulation appear more turbulent but also more prone to numerical instabilities. Thus, if you observe instabilities in your simulations, try smaller values for Re. Alternatively, you can increase the numerical resolution, i.e. N_y, of the channel.

3.2 Parameter file

The parameters of the Lattice-Boltzmann simulation are specified via a parameter file of the following form:

```
sizex 400
sizey 80
timesteps 50000
uin 0.02
Re 40
spherex 100
spherey 40
diameter 20
vtk_file example
vtk_step 300
```

This file specifies the number of fluid cells for your simulation as well as the number of time steps. The `uin` value is the x component of the inflow velocity and `Re` is the Reynolds number from Eq. (11). The center of the sphere with the given diameter is at (`spherex`,`spherey`). `vtk_file` gives the name of the visualization files and `vtk_step` specifies the spacing between two visualized time steps (Note: a value of zero for `vtk_step` should turn the visualization off). The first time step to be visualized should be time step `vtk_step`. It is advised to implement a dedicated FileReader class, that can appropriately handle this parameter file by performing the necessary type casts. Also have a look at the provided input file for an example.

4 Visualization

We use *Paraview* and VTK legacy files to visualize the results. Since only the interior of the channel is of interest, it is sufficient to only visualize these inner cells. For every cell, the flag, the fluid velocity, and the density should be written to the VTK files.

As an example consider the following annotated VTK file:

```
# vtk DataFile Version 4.0          # file header
SiWiRVisFile                        # this is only a comment
ASCII
DATASET STRUCTURED_POINTS
DIMENSIONS 50 50 1                  # number of points in all dimensions
ORIGIN 0 0 0
SPACING 1 1 1
POINT_DATA 2500                     # total number of points

SCALARS flags unsigned_int 1        # scalar field with identifier "flags"
LOOKUP_TABLE default                # default color table
1                                   # flag of first point
...

SCALARS density double 1            # scalar field with identifier "flags"
LOOKUP_TABLE default
1
...

VECTORS velocity double             # vector field with identifier "velocity"
-7.00552e-07 1.90304e-08 0
...
```

In order to create time series, the VTK files must be tagged with a number: `example0.vtk`, `example1.vtk`, ... *Paraview* automatically detects these time series and offers the functionality to visualize them in an animation. The tag for the time step and the file extension `.vtk` should be appended to the filename specified in the parameter file.

5 Implementation guideline

The following order of steps is recommended (but not mandatory) to implement the tasks of this assignment sheet:

1. Implement a Lattice class, preferably templated. To store the 2D lattice, a three dimensional array data structure is required where 2 dimensions are used for space and the third dimension holds the data per cell. For the PDF field, this third dimension is for indexing the distribution functions inside one cell. For the flag field, the third dimension is a single value and stores the integer flag.

2. Start with a small empty channel of size $N_x = 3$ and $N_y = 3$, so no obstacle is present. Check that the flag field is set up correctly.

3. Implement the stream step as well as boundary conditions. Initialize all fluid cells of your domain with the lattice weights w_q (corresponding to $f_q^{eq}(1, (0, 0))$ and give non-sense values inside the boundary layer. Then run your boundary handling function and one streaming step. If everything is implemented correctly, the f_q's should have the same value as before when using an inflow velocity of $\vec{u}_{in} = (0, 0)$. Also, no non-sense values should have entered your fluid domain.

4. Implement the collide step. Use the same setup and initialize the domain as in the previous step. Then run only your collide routine with different choices of $\tau \in (\frac{1}{2}, \infty)$. Again, the f_q should remain unchanged regardless of your choice of τ.

5. Implement the VTK writer to visualize your simulation state. This will greatly help with debugging and checking if everything is working correctly.

6. Implement the parameter file reader that allows you to quickly try out different scenarios.

7. Check your simulations of an empty channel with the complete LBM algorithm. Vary the domain sizes as well as the Reynolds number and the input velocity. In case of problems, make sure that you are still inside a stable region for the LBM method. The following rules of thumb should be kept in mind:

   - the maximum velocity of the fluid should not exceed 0.1 anywhere at any time.
   - τ should be larger than around 0.51. At the beginning of the simulation, print its value on the screen to check it!

8. Implement the routine to map the circle given in the input file into the domain. Again, test different input values for position and diameter. It is advised to keep some distance between the sphere and the outflow boundary to not cause instabilities at the outflow. If you encounter any problems, first check stream and collide again separately.

9. Run your simulation with the parameter files provided on StudOn. Those were used to create the two pictures at the beginning of this assignment sheet. Verify your code by comparing your results with these pictures.

6 Requirements

Please hand in your solution until Sunday, **July 22, 2022** at 23:55! Make sure the following requirements are met:

- The program should be compilable with a provided Makefile.
- The program should come with a suitable README file.
- The program should compile without errors or warnings with (at least) the following g++ compiler flags:

  ```
  -O3 -Wall -pedantic -std=c++17
  ```

- The program should be callable as

  ```
  ./lbm params.dat
  ```

  where `params.dat` is the parameter file.
- The program should execute correct simulations of the above described setup of a virtual wind tunnel with a circular obstacle.
- Submit your solution by uploading all required files to StudOn as a team submission.

7 Bonus tasks

There are two independent bonus tasks available. Make sure the regular task is working correctly before attempting any of them.

7.1 Obstacle from image

Extend the program such that it is possible to use a pgm image file to specify the domain size and the obstacle for the 2D simulation. White pixels with a value of 255 should be considered fluid cells, where all other gray values are considered no-slip cells for the obstacle. The boundary conditions for the channel remain unchanged and should be put around the domain given by the image. Additionally, extend your parameter file such that the `sizex`, `sizey` and all sphere parameters are replaced by a `geometry` parameter that specifies the name of the pgm geometry file:

```
timesteps 50000
uin 0.02
Re 100
vtk_file example_image
vtk_step 300
geometry image.pgm
```

Using the provided input file and image of a wing profile, the outcome can be seen in Fig. 7. Feel free to experiment around and be creative: how about a car placed at the bottom of the wind tunnel? You can create your own geometry files e.g. with the tool GIMP.

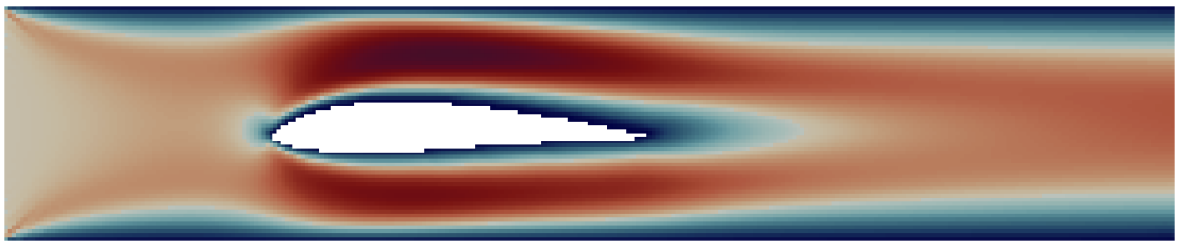


Figure 7: Bonus task 1: Wing profile inside the channel for $Re = 100$.

7.2 Drag and lift forces on obstacle

Extend the functionality of your simulator such that it computes the drag and lift forces that are acting on the obstacle due to the flow around it.

To compute the force $\vec{F}$ acting on the obstacle, use the so-called momentum exchange method which is given as:

$$\vec{F} = \sum_{q} 2 f_q \vec{c}_q, \tag{12}$$

where the q here are the directions from a fluid cell towards the obstacle, evaluated at all fluid cells around the obstacle. This should be carried out right after, or in combination with, the boundary handling.

Then, the x component of this force $\vec{F}$ is the so-called *drag* and the y component is the *lift* force. Print these values on the screen after each time step.

To easier distinguish between the no-slip cells of the boundary layer and the no-slip cells from the obstacle, extend the flag field by another integer value, i.e. 4, to mark the cells of the obstacle. For verification, keep in mind that a sphere placed in the center between the channel walls should feel no lift in the laminar case, i.e. low Re. If you also did the first bonus task, you can check if you actually observe a positive lift force, i.e. a plane with such a wing profile would by able to fly.

8 Credits

1. You are awarded with up to five points if your program correctly performs the above tasks and fulfills all of the above requirements. Submissions with compile errors will lead to zero points! Therefore the computers in the computer science CIP pool act as reference environments.

2. For each bonus task up to one bonus point is awarded for successful completion.