Chair for System Simulation

Harald Köstler, Dr.-Ing.
Sebastian Eibl, M. Sc.

# High End Simulation in Practice

Summer Term 2018

Assignment 3: Acceleration Techniques

## Submission Guidelines

To pass the course it is obligatory to submit (to Studon) and present your solution of all three assignment sheets to the tutors. Submission in groups of three is mandatory. Your solution must always include a appropriate Makefile. Additional requirements may be stated in the tasks.

**Due to numerical inaccuracies you might not be able to reproduce the *blocks_big* example exactly. However, please make sure that the general behaviour is comparable!**

In this exercise you will improve the performance and scalability of last assignment's simulation application. You are free to use either CUDA or OpenCL (of course you can also code both versions).

In the Casa Huber CIP-Pool, already OpenCL 1.2 headers are installed, while the available platforms identify themselves as OpenCL 1.1. Therefore (and since C++-bindings for OpenCL 1.2 are not yet installed), you should use the (deprecated) OpenCL 1.1 C++-bindings. To do so, you have to define the C-Macro `CL_USE_DEPRECATED_OPENCL_1_1_APIS` before including `CL/cl.hpp`.

# 1 Theory

As you have determined in the last exercise, the force calculation step of your simulation scales with $\mathcal{O}(N^2)$ which inhibits scalability. One technique to reduce the complexity is to split the computational domain into cells and assigning particles to them. With that approach the neighborhood information can be restricted. This approach is usually referred to as *binning* or *linked-cell algorithm*. Another option is explicitly storing neighbor lists for each particle. To form an efficient algorithm, the extend of the neighborhood is usually increased to make updated of the acceleration structure only necessary every couple of steps. We call this approach *neighbor lists*.

# 2 Implementation

The first step in implementing the linked-cell algorithm is given by supporting domains with limited extend. To this end, you shall introduce axis-aligned cuboids. Compared to *assignment 2*, you have to extend your parameter file so that it can handle these domains. Therefore the following parameters have to be added:

- Floating point values `x_min`, `y_min` and `z_min` to specify the coordinates of the lower corner point and `x_max`, `y_max` and `z_max` to specify the upper corner point.
- Unsigned integer values `x_n`, `y_n` and `z_n` to specify the number of cells in each direction (in preparation of the linked-cell algorithm).

From these parameters you can derive the remaining domain parameters. The *cell length* in $x$-dimension calculates as `len_x=(x_max-x_min)/x_n` (analoguous for $y$ and $z$-direction). The global domain then is

$$\Omega = [\texttt{x\_min}, \texttt{x\_max}] \times [\texttt{y\_min}, \texttt{y\_max}] \times [\texttt{z\_min}, \texttt{z\_max}]. \tag{1}$$

Next, boundary conditions are required. For this assignment, you shall implement periodic boundary conditions, i.e. particles that leave the domain shall enter the domain again at the opposite side. The periodic behavior has to be considered in two ways:

1. After each position update the positions of particles have to be checked: if particles have left the domain their positions have to be corrected. Note that this also has to be considered at initialization time.

2. In the force calculation: cells on the opposite side of the domain may belong to the neighborhood and the distance vector $\mathbf{x}_{ij}$ has to be corrected if two particles are on different sides of a boundary.

After boundary conditions are in place (and working!) you can start with implementing the binning approach. As discussed in the lecture, you have to assign all the particles to a cell, according to their positions. This basically means building a linked list of indices for each cell, which has to be re-done in every step, more precisely after each position update. Moreover, you need to add the *cut-off radius* for the force calculation, given by `r_cut`, to the parameter file. Assert that it is not larger than the smallest cell length. When implementing, also make sure you use the force term coming from the truncated Lennard-Jones potential. For the force calculation, you need to be able to support the cell-parallel approach and the particle-parallel approach as discussed in the exercises. In the case of cell-parallel implementation, it is a good idea to do a 3D indexing of the work-items. Therefore, extend the parameter file by the unsigned integer values `cl_workgroup_3dsize_x`, `cl_workgroup_3dsize_y` and `cl_workgroup_3dsize_z` in addition to the 1D `cl_workgroup_1dsize` (or suitable CUDA variants). The latter might be further used for kernels that are organized in 1D indexed work-items, i.e. executed particle-parallel.

Next, you can start to implement the neighbor lists. Here, each particle keeps a list of other particles in the neighborhood. Extend the parameter file with required parameters for specifying the maximum velocity, the update frequency for the acceleration structure, i.e. how many time steps may pass until a rebuild is necessary, and the maximum number of items in the neighborhood list. From these values you can calculate the required extension beyond the cut-off radius. Think about suitable values for these parameters!

As in the previous assignment, we provide some examples to test your code:

1. *grid*: This small ensemble should stay in perfect equilibrium, the particles should not move at all. If they move slightly then the Lennard-Jones force is probably not calculated in a numerically stable way.

2. *periodicity*: This small ensemble can be used as a check if the periodic boundaries work correctly.

3. *blocks_big*: This is a larger ensemble (1250 particles) that can be used for performance analysis. Note that the lower corner point of the domain is not the origin here! Note also that this setup can get unstable if you run it longer than specified in the parameter file.

Before you continue make sure that all variants of the code (brute force, cell-parallel binning, particle-parallel binning and neighbor lists) yield identical results!

# 3 Performance comparison

After successfully implementing the acceleration techniques it is time to compare the performance. We consider all four variants, i.e. brute force, cell-parallel binning, particle-parallel binning and neighbor lists. First, find suitable values for specialized parameters to tune the single variants. Next, compare the obtained performance results. Compile a PDF showing your findings (parameter configurations and timings!) as well as your interpretation of the results.

As always, please think about which parts of your program you want to profile and what you want to examine before taking any measurements!

# 4 Submission

**To pass you need to upload your code for all four cases (brute force, cell-parallel binning, particle-parallel binning and neighbor lists) together with a suitable Makefile and your performance comparison PDF via StudOn and present them (as a team!) in the computer exercise.**