

Chapter 2

Molecular Dynamics Simulation

Abstract This section provides a compact description of the basics of MD simulation. It only covers topics that are required to understand MD simulation in process engineering, i.e. in particular molecular modeling, the computation of potentials and forces, as well as the efficient identification of neighboring molecules. Here focus is put on single- and multi-center interactions based on the Lennard-Jones potential for short-range interactions. These detailed descriptions help to elaborate the differences between MD in process engineering and other fields and motivate the development of a specialized code. Such a code is `ls1 mardyn`, whose optimizations are discussed in the up-coming chapters. At the end of the section we provide the general layout of the software.

Keywords Molecular dynamics simulation • Molecular interactions • Short-range interactions • Linked-cells algorithm • Lennard-Jones potential • Single-center interactions • Multi-center interactions • `ls1 mardyn`

In this section, we give a compact description of the basics of MD simulation and cover only topics required to understand MD simulation in process engineering, i.e., in particular molecular modeling the computation of potentials and forces, as well as the efficient identification of neighboring molecules. This description helps to elaborate the differences between MD in process engineering and other fields, thereby focusing on algorithms, and motivates the development of a specialized code. Such a code is `ls1 mardyn` which is described at the end of this section.

2.1 Molecular Models and Potentials

The development of molecular models is a nontrivial task. Models have to capture the typical behavior of fluids and the geometric shape of a molecule to allow for meaningful simulations. At the same time, models should be as simple as possible to be computationally efficient. In this section, we discuss the design space for molecular models, especially from the point of view of algorithms and implementation. After

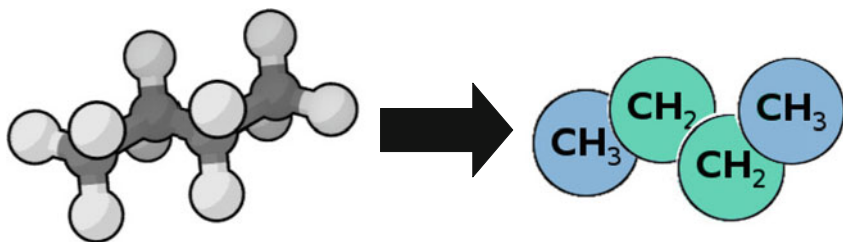


Fig. 2.1 Principle of coarse graining: A fully atomistic (*left*) and united-atom (*right*) model for butane. Atoms of functional groups are combined into a simple *united site*, achieving a compromise between computational tractability and microscopic detail

a description of the numerical system to be solved by time integration, the potential types relevant to `ls1 mardyn` are introduced.

Molecular Models For computer simulation, a model for a molecule, e.g., a polymer as displayed in Fig. 2.1, is a basic prerequisite. Depending on the required level of detail, this can be done in numerous ways. In the simplest case from a modeling point of view, each atom is represented as an individual particle in the model. Often, it is not an individual atom such as a single H-atom that determines the behavior of a molecule, but rather a group of atoms, e.g., a CH_2 group. Thus it is common to combine a group of atoms into one interaction site, which is then called a united-atom model. For some purposes it is possible to abstract even further and to unite several atom groups in one interaction site. It is important to decide if positions and orientations of these groups relative to each other are fixed, i.e., if the molecule is rigid or not. This has dramatic influence on algorithms and implementations and also relates to the time span which can be simulated, as motion takes place on different time scales. Intramolecular vibrations such as between C-H atoms are very fast and require very small time steps, while rotational motion is an order of magnitude slower, only slightly faster than translational motion. Consequently, vibrational degrees of freedom reduce the possible simulation time significantly. The coarser such a model is, the computationally cheaper it is, enabling larger or longer simulations. More complex molecular models necessitate more work for parametrization, on the other hand their transferability may be higher. Thus, the decision for a type of model is a trade-off between development effort, transferability, and computational efficiency.

Figure 2.2a shows a molecular model with two sites, which are fixed relatively to each other, while the model in Fig. 2.2b features internal degrees of freedom, i.e., the bond-length is flexible. The interaction between two interaction sites i and j , separated by a distance r_{ij} , can be described by a potential function $U_{ij}(r_{ij})$, which depends on the type of the interaction sites. For flexible molecules, interaction sites interact with all other interaction sites, including those of the same molecule. This interaction leads to a force on site i :

$$F_i = \sum_j -\nabla U_{ij}(r_{ij}).$$

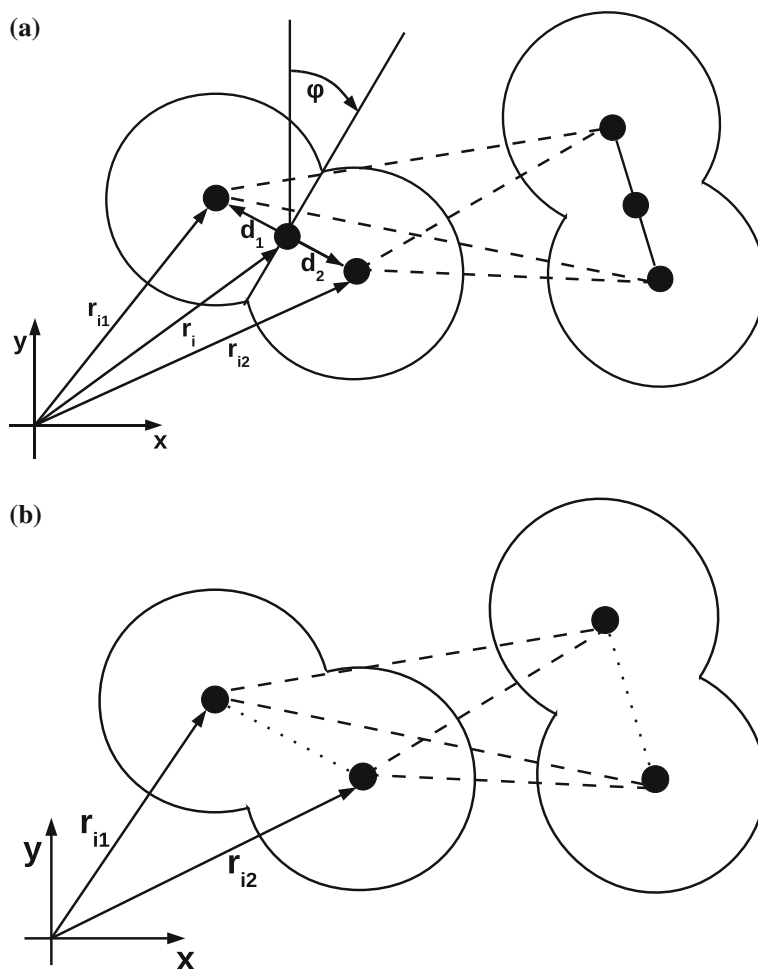


Fig. 2.2 Rigid and flexible model of a simple molecule. **a** Rigid model of a molecule: the positions of the interaction sites 1 and 2 are fixed relative to the center of mass, only sites of distinct molecules interact pairwise (*dashed lines*). The position of all sites is uniquely determined by position and orientation of the molecule. **b** Model of a molecule with internal degrees of freedom: the interaction sites 1 and 2 are not fixed, but interact through bond potentials (*dotted lines*). The positions of the sites have to be stored explicitly with each site

To observe the time evolution of such a system of particles, where each particle has mass m_i , a system of ordinary differential equations has to be solved:

$$F_i = m_i \cdot \ddot{r}_i. \quad (2.1)$$

One way to keep a molecule or parts thereof rigid is to compute forces for each interaction site separately, and to impose geometric constraints on bond lengths, bond or torsion angles. These constraints have to be fulfilled by the algorithm for the time

integration. The most common algorithm is the Shake algorithm [1], which is based on the Störmer-Verlet integration scheme, and also more sophisticated variants such as QSHAKE [2] or PLINCKS [3] have been developed. However, a more efficient way is to compute the torque on molecule i , resulting from the interactions of its interaction sites $n \in \text{sites}_i$, and to integrate the rotational motion. In this model, only forces between sites of different molecules are computed and the total force on a rigid molecule equals

$$F_i = \sum_{\substack{j \in \text{particles} \\ j \neq i}} \sum_{n \in \text{sites}_i} \sum_{m \in \text{sites}_j} -\nabla U_{nm}(r_{nm}).$$

This force is used to solve Eq. (2.1). The forces on the sites at distance d_n from the center of mass at r_i yield a torque on the molecule

$$\tau_i = \sum_{n \in \text{sites}_i} d_n \times F_n.$$

Then the system of equations for the rotational motion can be solved

$$\dot{\omega}_i = \frac{\tau_i}{I_i},$$

where $\dot{\omega}$ is the angular acceleration, τ the torque, and I the moment of inertia. Here, we remark that the computation of forces on a molecule involves all other molecules in the simulation, so the complexity is $O(N^2)$ for both rigid and flexible molecular models.

Many of the fluids targeted in process engineering are composed of comparably simple, small molecules (in the following, we use the term particle interchangeably), which can be approximated by rigid bodies. A rigid model enables a cheaper implementation of the force computation as well as longer time steps. Since our code is based on rigid-body motion, we describe rigid-body molecular dynamics in more detail. Most other current software packages implement rigid-body MD by constraint-motion dynamics, which is less efficient, so this is a key aspect to distinguish `ls1 mardyn` from other simulation codes.

Rigid-Body Molecular Dynamics For molecules modeled as fully rigid units, both equations for translational and rotational motion can be solved at once, if force and torque on its center of mass are known. In `ls1 mardyn`, the Rotational Leapfrog algorithm [4] is implemented.

While the orientation of a body can be expressed in Eulerian angles, it is more convenient to use a quaternion $q = (q_0 \ q_1 \ q_2 \ q_3)^T$, because singularities in the equations of motion are avoided [5]. The computation of the angular acceleration is carried out in a body-fixed coordinate system, i.e., the coordinate system is fixed relative to the rotating molecule. This body-fixed coordinate system should be chosen such that the mass tensor I is a diagonal matrix, simplifying the following equations. From the

quaternion, a rotation matrix $R(q)$ can be defined to express a vector, given in the global coordinate system, in the body-fixed system. The inverse operation is denoted by $R^T(q)$.

The rate of change of the angular momentum j equals the torque τ , $\frac{\partial j}{\partial t} = \tau$, and the angular velocity ω is related to the angular momentum by $\omega = I^{-1}j$. Writing the angular velocity as $\hat{\omega} = [0; \omega]^T$, the rate of change of the orientation can be expressed as

$$\frac{\partial q}{\partial t} = Q\hat{\omega}, \text{ where } Q = \begin{pmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & q_1 & q_0 \end{pmatrix}.$$

Similar to the Leapfrog scheme for the translational motion, the angular momentum j is stored at half time steps $n - \frac{1}{2}$, and the orientations at full time steps n . Starting now at time $n - \frac{1}{2}$, the angular momentum $j^{n-\frac{1}{2}}$ is propagated to time n :

$$j^n = j^{n-\frac{1}{2}} + \frac{1}{2}\Delta t \cdot \tau.$$

It is then rotated to the body-fixed coordinate system:

$$\hat{j}^n = R^T(q^n)j^n,$$

and the angular velocity in body-fixed coordinate frame can be determined component wise:

$$\hat{\omega}_\alpha^n = I_\alpha^{-1} \hat{j}_\alpha^n.$$

The orientation is integrated a half time step, where $q^{n+\frac{1}{2}} = q^n + \frac{\Delta t}{2} Q(q^n)\hat{\omega}^n$. The remaining steps read [4]:

$$j^{n+\frac{1}{2}} = j^{n-\frac{1}{2}} + \Delta t \tau^n,$$

$$\hat{j}^{n+\frac{1}{2}} = R^T(q^{n+\frac{1}{2}})j^{n+\frac{1}{2}},$$

$$\hat{\omega}_\alpha^{n+\frac{1}{2}} = \hat{I}_\alpha^{-1} \hat{j}_\alpha^{n+\frac{1}{2}},$$

$$q^{n+1} = q^n + \Delta t Q(q^{n+\frac{1}{2}})\hat{\omega}^{n+\frac{1}{2}}.$$

In the course of these computations, angular velocity and momentum are computed at the full time step and can be used to apply a thermostat.

Intermolecular Potentials Theoretically, particle interaction needs to be modeled by many-body potentials, which take the interactions between $n-1$ particles into account when determining the potential energy for the n -th particle. As the construction of

potential functions is a highly nontrivial task, interaction models are simplified to two- or three-body potentials, where the contributions of all pairs or triples of particles are assumed to be strictly additive. Choosing the “right” potential functions, this results in much lower computational cost while sufficient accuracy is maintained. In 1s1 mardyn, the following effective pair potentials are used [6]:

Lennard-Jones-12-6 Potential. This potential models Van der Waals attraction and Pauli repulsion and describes uncharged atoms:

$$U(r_{ij}) = 4\epsilon \left(\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^6 \right). \quad (2.2)$$

Consequently, this potential reproduces properties of noble gases very well, and is both used for the study of ideal fluids as well as a building block for complex molecular models. The potential parameters ϵ and σ are valid only for interaction sites of the same species. For interactions of two unlike species A and B , their value can be determined by the modified Lorentz combination rule [7]

$$\sigma_{AB} = \eta_{AB} \frac{\sigma_A + \sigma_B}{2}, \quad 0.95 < \eta_{AB} < 1.05$$

and the modified Berthelot mixing rule [8, 9]:

$$\epsilon_{AB} = \xi_{AB} (\epsilon_A \epsilon_B)^{\frac{1}{2}}, \quad 0.95 < \xi_{AB} < 1.05,$$

where η_{AB} and ξ_{AB} are empirically determined mixing coefficient. The potential can be truncated at a cut-off distance r_c , assuming a homogeneous particle distribution beyond r_c . This truncated potential, referred to as Truncated-Shifted Lennard-Jones-12-6, allows the construction of efficient algorithms with linear runtime $O(N)$. The error of the potential truncation can be estimated by a mean-field approximation to correct the computed quantities. For rigid-body molecules, the cut-off is applied based on their center of mass.

Electrostatic Potentials. Another basic interaction type are Coulomb interactions:

$$U_{qq}(r_{ij}) = \frac{1}{4\pi\epsilon_0} \frac{q_i q_j}{r_{ij}}, \quad (2.3)$$

where $1/(4\pi\epsilon_0)$ is the Coulomb constant, and q_i and q_j are interacting charges. and r_{ij} is the distance between the charges. Charge distributions with zero net charge may be approximated by higher order point polarities, i.e., dipoles and quadrupoles as described in [10].

If the net charge of molecules equals zero, these potentials can also be truncated at a cut-off distance r_c . The effect of the truncation on the potential energy can be estimated by the Reaction-Field method [11, 12].

2.2 Statistical Ensembles

The computation of macroscopic values from microscopic quantities is the field of statistical mechanics. In the following, we explain the basics as far as necessary to understand the implementation in `ls1 mardyn` and refer to [11] for more details.

The current state of a rigid-body MD simulation can be fully described by the number of particles, their positions, velocities, orientations, and angular momenta. From such a configuration, macroscopic quantities such as temperature or pressure can be computed. Many molecular configurations exist, which map to the same macroscopic value, i.e., these configurations cannot be distinguished on the macroscopic level. The set of configurations forms a so-called ensemble. In order to characterize an ensemble, it is sufficient to determine three thermodynamic variables, e.g., number of particles N , volume V , and total energy E (NVE). For all other thermodynamic variables, fluctuations can occur and their value can be determined through averaging over samples of configurations. Other common ensembles fix temperature T (NVT), pressure P (NPT), or the chemical potential μ (μ VT). In the thermodynamic limit, i.e., for infinite system sizes, these different statistical ensembles are equivalent for homogeneous systems and basic thermodynamic properties can be computed as averages:

- The total **energy** E is computed as the sum of the ensemble averages of the potential energy U_{pot} and the kinetic energy E_{kin} :

$$E = \langle U_{pot} \rangle + \langle E_{kin} \rangle = \left\langle \sum_i \sum_{j>i} U(r_{ij}) \right\rangle + \left\langle \sum_i \frac{1}{2} m_i v_i^2 \right\rangle.$$

- Following the virial theorem [13], the **temperature** T can be computed as

$$T = \left\langle \frac{1}{3N_f k_B} \sum_{i=1}^N v_i^2 m_i \right\rangle.$$

Here, N_f denotes the number of molecular degrees of freedom in the simulation, and k_B the Boltzmann constant.

- The **pressure** P can be split in a ideal part and a configurational or virial part and computed as

$$P = \langle P^{\text{ideal}} \rangle + \langle P^{\text{conf}} \rangle = \langle \rho k_B T \rangle - \left\langle \frac{1}{3V} \sum_i \sum_{j>i} r_{ij} \cdot f_{ij} \right\rangle,$$

where r_{ij} denotes the distance and f_{ij} the force between interaction sites i and j .

Simulations in the NVE ensemble are most self-evident. Energy is kept constant automatically, as solving the Newtonian equations conserves energy and momentum. To exclude boundary effects and to minimize finite-system effects, periodic boundary conditions are typically imposed on simulations. If particles leave the domain on one

boundary, they enter the domain via the opposite boundary again, so the number of particles does not change as well as the volume.

For NVT simulations, a thermostat is needed to keep the system at constant temperature. Conceptually, this is achieved by coupling the simulated system to an external heat bath, so that a weak exchange takes place, without disturbing the system under consideration. While several algorithms have been proposed [14] and especially relaxation schemes are popular, a very simple and effective method is to scale all particle velocities by a factor

$$\beta = \sqrt{\frac{T_{\text{target}}}{T_{\text{current}}}}.$$

Velocity scaling does not strictly preserve the NVT ensemble; however, it is often used assuming that the simulated system is not severely disturbed by the thermostat [15]. To conserve all these ensembles, modified time integration schemes have been developed, which solve the equations of motion in a suitable way. In `ls1 mardyn`, the thermostatted version of the Rotational Leapfrog algorithm [4] is used for NVT simulations.

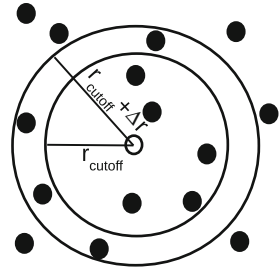
2.3 Algorithms for Nearest-Neighbor Search

As already noted in the explanation of the truncated-shifted Lennard-Jones potential, it is often possible to truncate potentials at a cut-off radius r_c , such that potential and force on a particle depend only on its local neighborhood. The efficient identification of that neighborhood can bring down the runtime complexity from $O(N^2)$ to $O(N)$, allowing for an asymptotically optimal algorithm. In the following, we discuss algorithms commonly implemented in MD codes.

Direct Summation. This is the simplest implementation of neighbor search. The distances between all particle pairs are computed, and only for those separated by less than r_c interactions are computed. While still quadratic runtime complexity is maintained, this is the most efficient algorithm for small particle sets, as it does not incur overhead for the particles' organization. Additionally, it can easily be vectorized and parallelized, and well-known optimizations such as cache blocking can be applied, such that the implementation becomes truly compute bound and achieves a high fraction of peak performance. Direct summation became especially popular for implementations on GPGPUs, due to its simplicity and inherently high parallelism, which fits well to the architecture and programming model.

Verlet Neighbor Lists [16]. Another frequently used approach are Verlet neighbor lists, shown in Fig. 2.3. For each particle, pointers to molecules in a “skin” radius $r_c + \Delta r$ are stored in a list. In order to find all neighboring particles within distance r_c , only the particles in the list have to be checked. Depending on the movement of the particles and the value of Δr , that neighbor list has to be updated every n time

Fig. 2.3 Schematic of the Verlet neighbor list



steps, to make sure it contains all neighboring molecules. In principle, this update is of $O(N^2)$ complexity, as again the mutual distance between all particles has to be computed, so modern implementations combine it with the linked-cells algorithm as explained in the next paragraph to achieve linear runtime.

It is an obvious advantage of the neighbor lists that they approximate the geometry of the cut-off sphere well, and only few unnecessary distance computations have to be performed. On the other hand, the complexity of the implementation is slightly higher, as both the linked-cells and the neighbor list method have to be implemented. Verlet lists are most efficient in static scenarios, where the movement of particles between time steps is very slow. This holds, e.g., for simulations at very low temperatures, at small time steps, or generally in the simulation of solid bodies such as crystals. The overhead for large number of pointers per molecule, maybe even up to a few hundreds, has to be considered as well. While it is usually not a serious issue on current computers, there is a clear trade-off between memory and computational overhead. A more severe question is the runtime-efficient implementation of neighbor lists on current hardware. Pointers do not preserve locality, as it is required for vectorization. In contrast to earlier vector computers, today's vector architectures do not support gather and scatter operations efficiently yet which would facilitate the implementation. Apart from that, the multiple memory accesses traversing the neighbor list can seriously degrade performance on current architectures [17].

Linked-Cells Algorithm [18, 19]. As depicted in Fig. 2.4a, the computational domain is subdivided into cells of length r_c . In every time step, the particles are sorted in these cells according to their spatial coordinates. In order to identify all particles within the cut-off radius around a given particle, only 8 neighboring cells in 2D or 26 cells in 3D as well as the cell of the particle itself have to be searched. Assuming a homogeneous particle distribution, each cell contains $\frac{N}{c}$ particles, where c denotes the number of cells; so the distance computation can be done in $O(N)$, as long as the particle density is kept constant. As we will see later, this algorithm is inherently cache-friendly, as for each cell $(N/c)^2$ computations are performed.

Since its invention, a lot of work has gone into the optimization of the linked-cells algorithm. Evident from simple geometric considerations, roughly 78 % of the particle distance computations are actually wasted, because the particles are separated by a larger distance than r_c . One refinement is the generalized linked-cells algorithm, which chooses cells of smaller size to better approximate the geome-

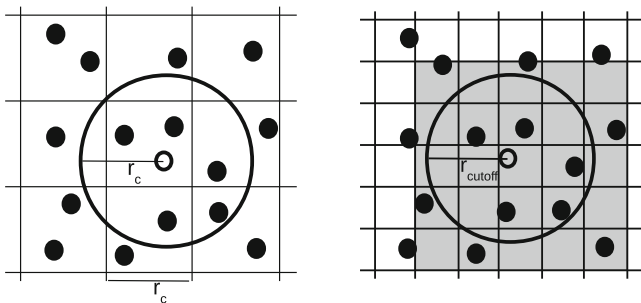


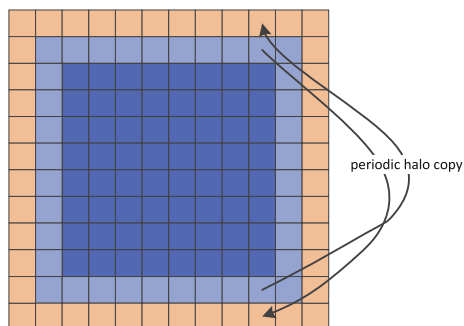
Fig. 2.4 Standard linked-cells algorithm and generalized linked-cells algorithm. **a** Schematic of the original linked-cells idea with edge length $l = r_c$. **b** Schematic of the original linked-cells idea with edge length $l = \frac{r_c}{2}$

try of the cut-off sphere, cf. Fig. 2.4b. Thereby, the volume that has to be searched for neighboring particles is decreased as well as the number of distance computations [20]. The efficiency of such schemes has also been investigated in [21], among others. Buchholz [22] extended this idea by choosing the size of the cells adaptively, depending on the density of the fluid: For regions with high number density, smaller cells are chosen to decrease the overhead of distance computation, for regions with low-number density, larger cells are chosen to avoid the overhead associated with many small cells. That scheme is called adaptive linked-cells algorithm. A different optimization technique is interaction sorting [23], where for each cell pair the coordinates of the particles are projected onto the vector connecting the two cell centers. Then the particles are sorted according to their position on that vector. In that way, the distance between particles needs to be computed only as long as the distance along the vector of the cell centers is smaller than r_c , greatly reducing the number of superfluous computations. A summary and comparison of the different approaches can be found in [24].

In comparison to direct summation, some overhead occurs due to the cell data structure and its update. An advantage is the seamless integration of periodic boundary conditions, as depicted in Fig. 2.5. The cell structure is extended by one cell layer at each boundary. Every time step, the particles of the opposite boundary are replicated in these so-called halo-cells. These particles are used during the force computation and deleted thereafter.

Linked-Cells: Parallelization based on Spatial Domain Decomposition. In a similar way, parallelization based on spatial domain decomposition fits especially well to the linked-cells data structure. Here, the computational domain is subdivided according to the number of processes, so that equally sized subdomains are assigned to each process. Similar to the integration of periodic boundaries, each subdomain is extended by one cell layer, which contains molecules residing on neighboring processes, which have to be communicated every iteration. Depending on the implementation, forces for each molecule pair crossing a process boundary have to be communicated, or

Fig. 2.5 Implementation of periodic boundary conditions: After each time step, particles in the boundary cells are copied into the halo-cells on the other side in the same coordinate direction



are computed redundantly on each process. Spatial domain decomposition can be efficiently combined with load-balancing scheme, its implementation will be topic of Sect. 3.3 and 3.4.

2.4 Characteristics of Large-Scale Molecular Dynamics Simulation of Fluids

A number of well-known simulation programs for molecular simulation exist such as *NAMD* [25], *GROMACS* [26], *Desmond* [27], *CHARMM* [28], *Amber* [29], *Espresso* [30] or *LAMMPS* [31]. Most of them having their background in molecular biology, these codes can be used for simulations in process engineering in principle. Then, however, the application of these tools may not be straightforward, requires odd work-flows and lacks computational efficiency, rendering these tools non optimal. In the following, we discuss properties of MD simulations in biology and in chemical engineering. On the basis of the preceding sections, we outline algorithmic differences and contrast requirements on implementations for each field of application.

MD Simulation in Molecular Biology. A typical use case of MD simulation in biology is the study of protein folding, where the probability of conformations is determined. Such simulations deal with only few but large macromolecules to observe conformational changes. Since results are investigated in lab experiments in more detail, MD is used to dramatically narrow the search space for lab experiments.

In explicit solvent-type simulations, macromolecules float in a large homogeneous bulk of water or aqueous salt solution, which is the natural environment of proteins. In order to mitigate finite-size effects (e.g., the interaction of a protein with its own periodic image), a sufficiently large simulation box filled with water has to be simulated [32]. Yet the total number of molecules is comparably small and typically in the order of 1000–10,000. These simulations have to take place at the atomic level, slightly increasing the number of bodies to be dealt with, e.g., by a factor of three in the case of TIP3P water. Due to intramolecular vibrational motions,

small time steps have to be chosen. Simulation parameters are standardized to a high degree, e.g., computer experiments are run at ambient temperature, employing TIP3P, TIP4P, or SPCE water models [25, 26]. Every atom may participate in a number of interactions of different type, e.g., nonbonded electrostatic or Lennard-Jones interactions and bonded interactions. These characteristics strongly influence established simulation codes.

Consequences for Biomolecular Simulation Codes. Algorithms and their implementations are chosen to match with these properties as well as possible. For several reasons, Verlet neighbor lists are the method of choice for neighbor search in all the aforementioned simulation packages. Typical scenarios do not exhibit strong dynamics such as flows, and comparably small time steps due to the internal degrees of freedom have to be applied. Therefore, movement of atoms between two time steps is limited, which is favorable for Verlet lists. Moreover, bonded atoms have to be excluded from nonbonded interactions. This can be accomplished with exclusion lists, which in turn integrate nicely with neighbor lists. In contrast, computation with the linked-cells algorithm might require multiple force evaluations [33, p. 203]. Due to the high level of standardization, force fields can be supplied in form of libraries. For commonly used solvents such as TIP3P or TIP4P, GROMACS offers specially tuned interaction kernels boosting performance. The consideration of internal degrees of freedom results in a higher arithmetic intensity per atom: First of all, neighbor search is based on atoms instead of molecules, increasing computational complexity by a constant factor, e.g., nine in the case of a three-site water model. In addition, constraint-motion algorithms such as Rattle, Shake, or P-LINCS have to be applied, which are computationally more expensive and impair scalability.

Due to the presence of ions, it is necessary to treat long-range coulomb interactions with appropriate methods. As the number of molecules is rather small, Ewald summation techniques are implemented as standard methods, and FFT-accelerated Ewald techniques seem to be optimal. Because of the comparably low particle count, it is common to write trajectory files for each time step of the simulation and to investigate quantities of interest in a post-processing step.

MD Simulation in Chemical Engineering. In chemical engineering, MD simulations are used, e.g., to predict thermodynamic properties of mixtures of fluids, so these predictions have to match real data quantitatively with high precision [34].

While the simulation of a bulk of solvent is an unwanted necessity in biological applications, it is now the main purpose, and interest focuses on the computation of macroscopic properties such as transport coefficients or nucleation rates. To reduce statistical uncertainties and finite-size effects, large numbers of molecules up to several millions are required. Applications cover a wide range of thermodynamic states, e.g., very high or low pressures and temperatures. Often, molecular force fields have to be developed to correctly reproduce properties in these ranges [35]. For many applications, it is sufficient to model fluids composed of comparably simple, i.e. rigid molecules without internal degrees of freedom. Finally, applications such as phase transitions or processes at the liquid–vapor interface are characterized by strongly heterogeneous particle distributions.

Consequences for Simulation Codes in Engineering. Rigid molecular models simplify both computations of intermolecular interactions, as well as the solution of the equations of motion. The cut-off condition is not evaluated per atom, but for a whole molecule based on its center of mass, reducing the number of distance computations. Rigid molecules are uniquely assigned to a process, which reduces the complexity of an efficient parallelization. For molecules with internal degrees of freedom, atoms may reside on different processes, which requires additional communication and synchronization. Due to the rigidity, larger time steps are practical, allowing for longer simulation times in the end.

The number of molecules required for a meaningful simulation in chemical engineering can be larger by magnitudes, and has tremendous effects. While Verlet neighbor lists still may be usable, the linked-cell algorithm is a better choice, as memory overhead for storing pointers is avoided. Moreover, the simulation of flows or nucleation exhibits higher dynamics of molecules, so neighbor lists need to be rebuilt frequently, especially in combination with larger time steps. Due to the particle number, it is advisable to compute statistical data on the fly, instead of storing the particles' trajectories and running tools for post-analysis. While this is feasible for a scenario with e.g., 10,000 molecules, input/output (i/o) becomes a bottleneck for large-scale simulations with millions of particles. Post-processing tools would need to be parallelized to handle large amounts of data efficiently, so implementations are of similar complexity as the actual simulation code.

Particle distributions cause severe load imbalances. The density of liquid differs from that of gas roughly by two orders of magnitude. As the computational effort scales quadratically with the density, liquid phases are about 10,000 times as compute intensive as gas phases. For some processes such as nucleation, the distribution of particles evolves dynamically in an unpredictable manner, so an efficient dynamic load-balancing scheme is needed [22]. Dealing with heterogeneities can be supported by the choice of spatially adaptive algorithms, such as the adaptive linked-cells algorithm.

Concluding, requirements on codes for simulation in engineering are different from those for codes in biology. While the well-established codes for simulation in biology or chemistry can be used for the simulation of processes in chemical engineering, the characteristics of such simulations are quite different. In order to allow for high usability and to boost computational efficiency, codes specifically tailored to their field of application are essential.

2.5 Simulation Code Mardyn

Tackling the field of process engineering, the simulation code `ls1 mardyn` [6, 36] has now been developed for about a decade. Main contributors have been the groups at the High-Performance Computing Center Stuttgart (HLRS), at the Chair for Thermodynamics and Energy Technology (ThET) at University of Paderborn, and the Laboratory for Engineering Thermodynamics (LTD) at the University of

Kaiserslautern as well as the Chair for Scientific Computing in Computer Science (SCCS) at Technische Universität München.

The development has been inspired by `ms2` [37], a mature Fortran code for the molecular simulation of thermodynamic properties. Supporting both classical MD and MC simulation with rich functionality, `ms2` focuses on small molecular systems, so the investigation of nucleation or flow processes is hardly possible. Also the investigation of competing domain specific codes such as `Towhee`¹ or `GIBBS`² confirmed that codes for large-scale MD simulation in engineering sciences were rather limited [38]. Therefore, the work on a modern C++ code for large-scale parallel MD simulation was started.

In the current software layout, two design principles are dominating: The first is **Separation of Concerns**. Modular design is a key requirement for several reasons. First of all, academic partners from different disciplines develop the code at different geographic locations. Optimally, modifications or additions of features affect only small parts of the code, facilitating distributed software development. Furthermore, technical aspects should be separated from application-specific aspects. For example, a chemical engineer implementing the computation of new statistical quantities should not need to understand details of parallelization. In academic software development, developers change rather frequently, so modular design makes it easier to focus on specific aspects of MD simulation without the requirement to understand all parts of the software. Finally, modularity fosters the exchange of algorithms and their implementations.

The second principle is **one code base for sequential and parallel execution**, rather than having two distinct codes. Targeting parallel simulations, software development is simplified, if code can be developed, executed, and to a certain extent also tested sequentially. Maintenance of a single code base is less error-prone than having two similar codes, which need to be kept synchronous. Alternatively, sequential code could be interleaved with precompiler directives, which can make code harder to understand. Therefore, parts directly related to parallelization are hidden behind interfaces, and application-specific classes are implemented and tested independently of types of parallelizations.

Software Structure Although the above two design principles have not been strictly realized, they are heavily reflected in the software design. An UML diagram containing the main components of `ls1 mardyn` is shown in Fig. 2.6. The class `Simulation` is the central component, so all other classes are arranged around it. The main components and their relation is discussed in the following.

Class Simulation. This class is the heart piece of `ls1 mardyn` and ties together the different parts of the code. It is responsible for setting up a simulation run and executing the simulation loop, see the pseudo code Lst. 2.1.

After the initialization, i.e., after reading the phase space and creating a domain decomposition with initial particle exchange, the main loop is executed. First the

¹ <http://towhee.sourceforge.net/>.

² <http://www.materialsdesign.com/medea/medea-gibbs>.

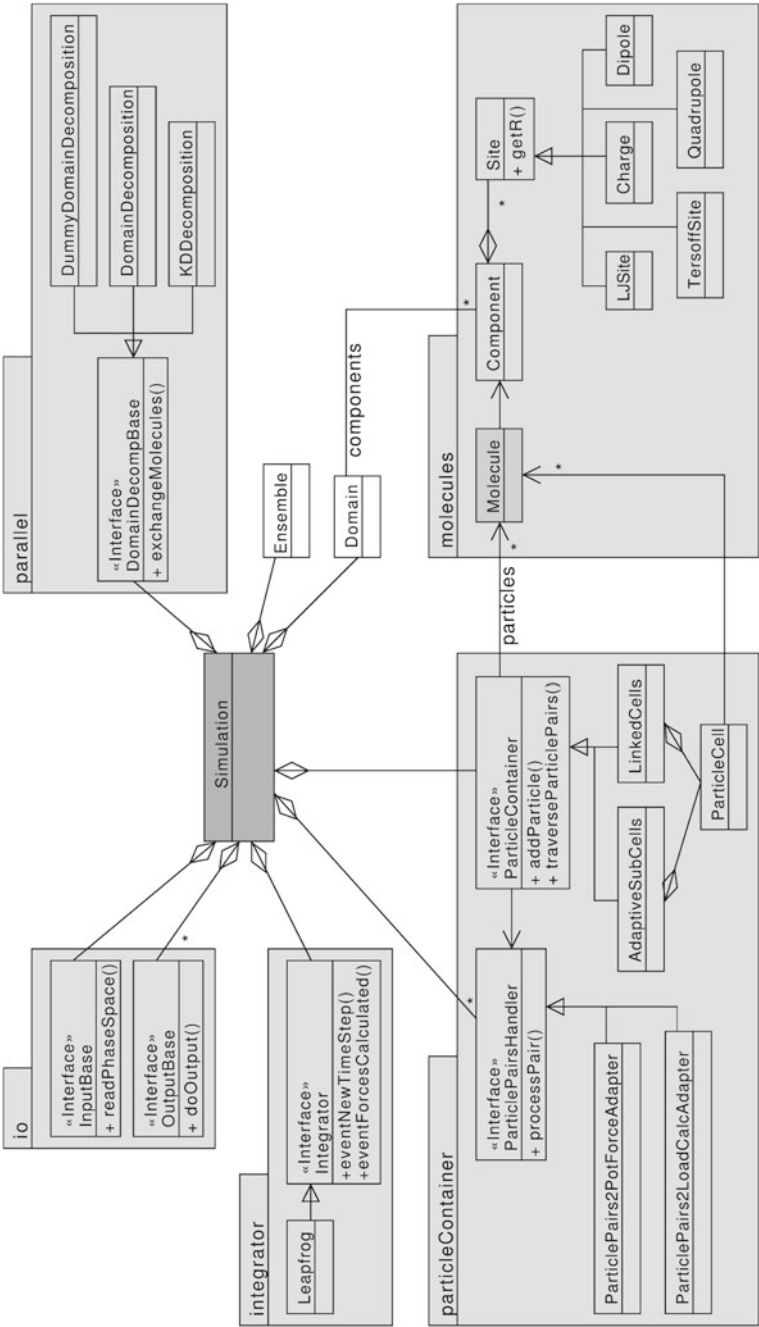


Fig. 2.6 Software layout of Martyn: The packages for parallelization, particleContainer, io, integration and the molecular model are centered around the main class Simulation

integrator performs a half-step time integration to promote the velocities to the full time step, then halo-particles are exchanged and load balancing may take place. Then forces and potential are calculated, the thermostat as well as computations of other thermodynamic statistical quantities are applied. At the end of the loop, the integrator performs the second half-step integration, and i/o is performed.

Listing 2.1 Pseudocode of the main simulation loop in the class `Simulation`.

```
inputReader->readPhaseSpace();
domainDecomposition->balanceAndExchangeMolecules();

for ( i < numberOfTimesteps ) {

    integrator->eventNewTimestep();
    domainDecomposition->balanceAndExchangeMolecules();
    container->traverseParticlePairs(pairs2ForceAdapter);
    thermostat();
    integrator->eventForcesCalculated();

    for ( k < numberOfOutputPlugins ) {
        outputPlugin[k]->doOutput();
    }
}
```

Package parallel. This package comprises everything related to parallelization based on the domain decomposition scheme explained in Sect. 3. Its interface is defined by `DomainDecompBase`, which is responsible for particle exchange and interprocess communication. `DummyDomainDecomposition` provides an implementation of this interface for sequential execution. `DomainDecomposition` is the standard domain decomposition method for MPI and `KDDecomposition` is an implementation providing load balancing based on KD-trees for MPI. In the case of sequential compilation, the latter two implementations are excluded.

Package io. `Io` provides two interfaces for file input and output. The method `readPhaseSpace` of `InputBase` reads the phase space, i.e., the definition of molecule types together with positions, orientations, and velocities of molecules. An `OutputBase` writes files containing e.g., visualization or restart data.

Package particleContainer. This package contains data structures for molecule storage and traversal. Thus, the main characteristics of a `ParticleContainer` are that molecule pairs can be traversed according to the cut-off radius. Initially, two implementations for the standard linked-cells algorithm and its adaptive version existed. Both organize particles with the help of `ParticleCells`. During the traversal of particle pairs, a `ParticlePairsHandler` is called for each pair with distance smaller than the cut-off radius r_c . Implementations of that interface compute interactions (`ParticlePairs2PotforceAdapter`) or determine the computational load associated (`ParticlePairs2LoadCalcAdapter`) in the context of load balancing.

Package integrator. Though providing an interface, only the `Leapfrog` integrator is supported at the moment. Implementing the Leapfrog Rotational Algorithm, it solves the molecules' equations of motion every time step.

Classes Domain and Ensemble. These classes are designed to contain all application-specific aspects such as evaluation of thermodynamic properties or enforcing the correct statistical ensembles. Consequently, the computation of energies, pressure, profiles, and long-range corrections is found here.

Package molecules. The implementation of the molecular model is based on the Flyweight design pattern [39]. Usually there is a large number of molecules of the same type in a simulation. A type or `Component` describes the number of `LJSites` or electrostatic interaction sites and the respective potential parameters. Each `Site` stores its position relative to the molecule, so its absolute global position has to be computed from the position and orientation of the molecule. For each component in the simulation, exactly one object is created and referenced by all molecules of the same type.

References

1. J.-P. Ryckaert, G. Ciccotti, H.J. Berendsen, Numerical integration of the cartesian equations of motion of a system with constraints: molecular dynamics of n-alkanes. *J. Comput. Phys.* **23**(3), 327–341 (1977)
2. T.R. Forester, W. Smith, SHAKE, rattle, and roll: efficient constraint algorithms for linked rigid bodies. *J. Comput. Chem.* **19**(1), 102–111 (1998)
3. B. Hess, P-LINCS: a parallel linear constraint solver for molecular simulation. *J. Chem. Theory Comput.* **4**(1), 116–122 (2008)
4. D. Fincham, Leap frog rotational algorithms. *Mol. Simul.* **8**, 165–178 (1992)
5. J.B. Kuipers, Quaternions and rotation sequences (Princeton University Press, Princeton, 1999)
6. C. Niethammer, S. Becker, M. Bernreuther, M. Buchholz, W. Eckhardt, A. Heinecke, S. Werth, H.-J. Bungartz, C.W. Glass, H. Hasse, J. Vrabec, M. Horsch, `Isl mardyn`: the massively parallel molecular dynamics code for large systems. *J. Chem. Theory Comput.* (2014)
7. H.A. Lorentz, Über die Anwendung des Satzes vom Virial in der kinetischen Theorie der Gase. *Ann. Phys.*, 12(1):127–136, (1881). Addendum 12(4):660–661
8. D. Berthelot, Sur le mélange des gaz. *Comptes rendus hebdomadaires des séances de l'Académie des Sciences*, 126:1703–1706, (1898). Addendum: vol. 126, no. 4, pp. 1857–1858
9. T. Schnabel, J. Vrabec, H. Hasse, Unlike Lennard-Jones parameters for vapor-liquid equilibria. *J. Mol. Liq.* **135**, 170–178 (2007)
10. C.G. Gray, K.E. Gubbins, *Theory of molecular fluids, Volume 1: Fundamentals* (Clarendon Press, Oxford, 1984)
11. M.P. Allen, D.J. Tildesley, *Computer Simulation of Liquids* (Oxford University Press, Oxford, 1989)
12. J. Barker, R. Watts, Monte Carlo studies of the dielectric properties of water-like models. *Mol. Phys.* **26**(3), 789–792 (1973)
13. R. Clausius, XVI on a mechanical theorem applicable to heat. *Philos. Mag. Ser. 4*, 40(265):122–127 (1870)
14. P.H. Hünenberger, Thermostat algorithms for molecular dynamics simulations, *Advanced Computer Simulation* (Springer, Berlin, 2005), pp. 105–149

15. L. Woodcock, Isothermal molecular dynamics calculations for liquid salts. *Chem. Phys. Lett.* **10**(3), 257–261 (1971)
16. L. Verlet, Computer experiments on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules. *Phys. Rev. Online Arch. (Prola)* **159**(1), 98–103 (1967)
17. S. Pll, B. Hess, A flexible algorithm for calculating pair interactions on SIMD architectures. *Comput. Phys. Commun.*, (2013). Accepted for publication
18. R. Hockney, S. Goel, J. Eastwood, Quiet high-resolution computer models of a plasma. *J. Comput. Phys.* **14**(2), 148–158 (1974)
19. P. Schofield, Computer simulation studies of the liquid state. *Comput. Phys. Commun.* **5**(1), 17–23 (1973)
20. M. Bernreuther, H.-J. Bungartz, Molecular simulation of fluid flow on a cluster of workstations, in *Proceedings of the 18th Symposium Simulationstechnique (ASIM 2005), Volume 15 of Fortschritte in der Simulationstechnik—Frontiers in Simulation*, ed. by F. Hülsemann, M. Kowarschik, U. Rüde (SCS European Publishing House, Erlangen, 2005), pp. 117–123
21. G. Sutmann, V. Stegailov, Optimization of neighbor list techniques in liquid matter simulations. *J. Mol. Liq.* **125**, 197–203 (2006)
22. M. Buchholz, Framework zur Parallelisierung von Molekulardynamiksimulationen in verfahrenstechnischen Anwendungen. Dissertation, Institut für Informatik, Technische Universität München (2010)
23. P. Gonnet, A simple algorithm to accelerate the computation of non-bonded interactions in cell-based molecular dynamics simulations. *J. Comput. Chem.* **28**(2), 570–573 (2007)
24. U. Welling, G. Germano, Efficiency of linked cell algorithms. *Comput. Phys. Commun.* **182**(3), 611–615 (2011)
25. J.C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R.D. Skeel, L. Kale, K. Schulten, Scalable molecular dynamics with NAMD. *J. Comput. Chem.* **26**(16), 1781–1802 (2005)
26. B. Hess, C. Kutzner, D. van der Spoel, E. Lindahl, Gromacs 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation. *J. Chem. Theory Comput.* **4**(3), 435–447 (2008)
27. K.J. Bowers, E. Chow, H. Xu, R.O. Dror, M.P. Eastwood, B.A. Gregersen, J.L. Klepeis, I. Kolossvary, M.A. Moraes, F.D. Sacerdoti, J.K. Salmon, Y. Shan, D.E. Shaw, Scalable algorithms for molecular dynamics simulations on commodity clusters. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, SC '06, ACM, New York, USA (2006)
28. B.R. Brooks, C.L. Brooks, A.D. Mackerell, L. Nilsson, R.J. Petrella, B. Roux, Y. Won, G. Archontis, C. Bartels, S. Boresch et al., CHARMM: the biomolecular simulation program. *J. Comput. Chem.* **30**(10), 1545–1614 (2009)
29. R. Salomon-Ferrer, D.A. Case, R.C. Walker, *An overview of the Amber biomolecular simulation package* (Computational Molecular Science, Wiley Interdisciplinary Reviews, 2012)
30. A. Arnold, O. Lenz, S. Kesselheim, R. Weeber, F. Fahrenberger, D. Roehm, P. Košovan, C. Holm, Espresso 3.1: molecular dynamics software for coarse-grained models, in *meshfree methods for partial differential equations VI*, p. 1–23. (Springer, 2013)
31. S. Plimpton, Fast parallel algorithms for short-range molecular dynamics. *J. Comput. Phys.* **117**(1), 1–19 (1995)
32. D.A. Case, T.E. Cheatham, T. Darden, H. Gohlke, R. Luo, K.M. Merz, A. Onufriev, C. Simmerling, B. Wang, R.J. Woods, The Amber biomolecular simulation programs. *J. Comput. Chem.* **26**(16), 1668–1688 (2005)
33. M. Griebel, S. Knapek, G.W. Zumbusch, Numerical simulation in molecular dynamics: numerics, algorithms, parallelization, applications, vol 5. (Springer, 2007)
34. B. Eckl, J. Vrabec, H. Hasse, On the application of force fields for predicting a wide variety of properties: ethylene oxide as an example. *Fluid Phase Equilib.* **274**(1–2), 16–26 (2008)
35. T. Merker, C. Engin, J. Vrabec, H. Hasse, Molecular model for carbon dioxide optimized to vapor-liquid equilibria. *J. Chem. Phys.*, 132(23), (2010)
36. The ls1 mardyn website (2014), <http://www.ls1-mardyn.de/>

37. S. Deublein, B. Eckl, J. Stoll, S.V. Lishchuk, G. Guevara-Carrion, C.W. Glass, T. Merker, M. Bernreuther, H. Hasse, J. Vrabec, ms2: a molecular simulation tool for thermodynamic properties. *Comput. Phys. Commun.* **182**(11), 2350–2367 (2011)
38. K.E. Gubbins, J.D. Moore, Molecular modeling of matter: impact and prospects in engineering. *Ind. Eng. Chem. Res.* **49**(7), 3026–3046 (2010)
39. E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design patterns: elements of reusable object-oriented software*. (Addison-Wesley, 1994)

Supercomputing for Molecular Dynamics Simulations

Handling Multi-Trillion Particles in Nanofluidics

Heinecke, A.; Eckhardt, W.; Horsch, M.; Bungartz, H.-J.

2015, X, 76 p. 35 illus., 13 illus. in color., Softcover

ISBN: 978-3-319-17147-0