Erschienen in: Journal of Computational Chemistry ; 26 (2005), 16. - S. 1719-1751
https://dx.doi.org/10.1002/jcc.20303

The GROMOS Software for Biomolecular Simulation: GROMOS05

MARKUS CHRISTEN, PHILIPPE H. HÜNENBERGER, DIRK BAKOWIES, RICCARDO BARON, ROLAND BÜRGI,* DAAN P. GEERKE, TIM N. HEINZ, MIKA A. KASTENHOLZ, VINCENT KRÄUTLER, CHRIS OOSTENBRINK,† CHRISTINE PETER,‡ DANIEL TRZESNIAK, WILFRED F. VAN GUNSTEREN

Laboratory of Physical Chemistry, Swiss Federal Institute of Technology Zürich, ETH-Hönggerberg, CH-8093 Zürich, Switzerland

Abstract: We present the latest version of the Groningen Molecular Simulation program package, GROMOS05. It has been developed for the dynamical modelling of (bio)molecules using the methods of molecular dynamics, stochastic dynamics, and energy minimization. An overview of GROMOS05 is given, highlighting features not present in the last major release, GROMOS96. The organization of the program package is outlined and the included analysis package GROMOS++ is described. Finally, some applications illustrating the various available functionalities are presented.

Key words: molecular dynamics simulation; programming; GROMOS; biomolecular simulation

Introduction

Starting with GROMOS80, the GROMOS program package has been developed over the past 25 years to facilitate research efforts in the field of biomolecular simulation in a university environment. The GROMOS software was and is meant for use in a scientific environment, which may be characterized by a continuously changing flow of users, who either wish to investigate and implement new simulation algorithms or intend to carry out applications of simulation in a variety of fields, ranging from polymers, glasses and liquid crystals, to crystals and solutions of biomolecules (proteins, nucleic acids, saccharides, and lipids). To this purpose, GROMOS has been developed based on the following principles: (1) transparency of the code, making modifications easy; (2) modular architecture, so that only parts of it need be modified for the implementation of new functionalities designed by users; (3) independence of the simulation code and the force field; and (4) independence of the simulation code and the computer hardware. The major releases of the GROMOS software are GROMOS87[1,2] developed at the University of Groningen, GROMOS96[3,4] developed at ETH Zürich, and now GROMOS05. GROMOS has found widespread use (hundreds of licenses in over 57 countries on all continents except Antarctica, see Fig. 1), triggered by the fact that it has been designed for ease of extendability and that the complete source code is made available to research establishments for a nominal fee (http://www.igc.ethz.ch/gromos). The program code has been further developed in the group for computational chemistry at ETH Zürich (Switzerland) throughout the recent years, leading now to a new major release, GROMOS05. The enhancements were governed by the following criteria: (1) interest of our research group,[5] (2) ease of use, (3) extendability, (4) demonstrated usefulness or efficiency of new methods, (5) well-defined and correct formulae and algorithms, and (6) computational efficiency. The second criterium led to a complete rewrite of the setup and analysis tools, now contained in the GROMOS++ setup and analysis subpackage, written in C++. The third criterium led to a rewrite in C++ of the MD engine, the part that carries out molecular dynamics (MD) or stochastic dynamics (SD) simulations as well as energy minimizations (EM), into a new program called MD++. In parallel, the original FORTRAN version of the

Correspondence to: W. F. van Gunsteren; e-mail: wfvgn@igc.phys.chem.ethz.ch

Contract/grant sponsor: National Center of Competence in Research (NCCR) Structural Biology of the Swiss National Science Foundation (SNSF)

*Present address: Swissre, Zürich, Switzerland.

†Present address: Vrije Universiteit, Pharmaceutical Sciences/Pharmacochemistry, De Boelelaan 1083 P262, NL-1081 HV Amsterdam, The Netherlands.

‡Present address: Max-Planck-Institute for Polymer Research, Ackermannweg 10, D-55128 Mainz, Germany.

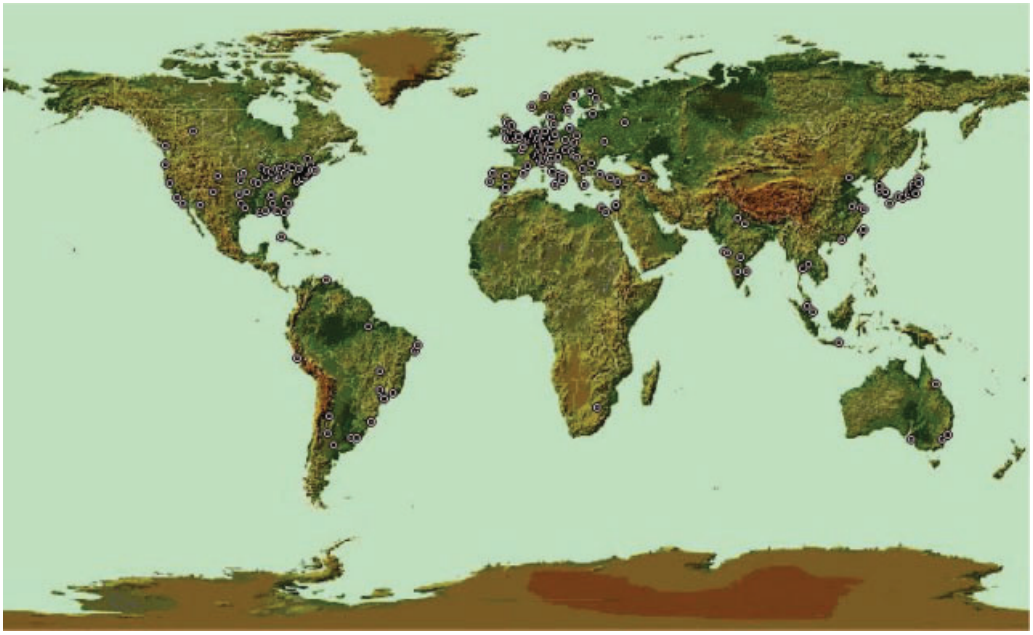

Figure 1. Distribution of GROMOS licenses.

MD engine, PROMD, was further developed to introduce many new features (some of which are not yet available in MD++). In the long term (beyond GROMOS05), MD++ will entirely replace PROMD.

In the next sections the main features of GROMOS05 are described. First, new functionalities with respect to GROMOS96 are highlighted. Second follows the algorithmic description of selected new functionalities. Third, the organization of the code is discussed and an overview of the programs present in the GROMOS++ analysis subpackage is provided. Fourth, examples of applications are reported for some of the newer features. Finally, summary and conclusions are provided.

Overview of Functionalities

Here, the main features of the two MD engines available in GROMOS05, PROMD and MD++ are described. These two programs share most of the basic functionalities, but still differ in a number of aspects. The FORTRAN MD engine (PROMD) retains all features of the GROMOS96 release and adds a number of new functionalities. The C++ MD engine (MD++) contains most of the GROMOS96 features (except four-dimensional and path-integral simulations), a subset of the new functionalities recently introduced into PROMD (since the GROMOS96 release), and some new features of its own.

A nonexhaustive list of the features included is:

- molecular dynamics (MD) simulation, stochastic dynamics (SD) simulation, and energy minimization (EM; steepest descent or conjugate gradient);
- periodic boundary conditions (vacuum, rectangular, truncated

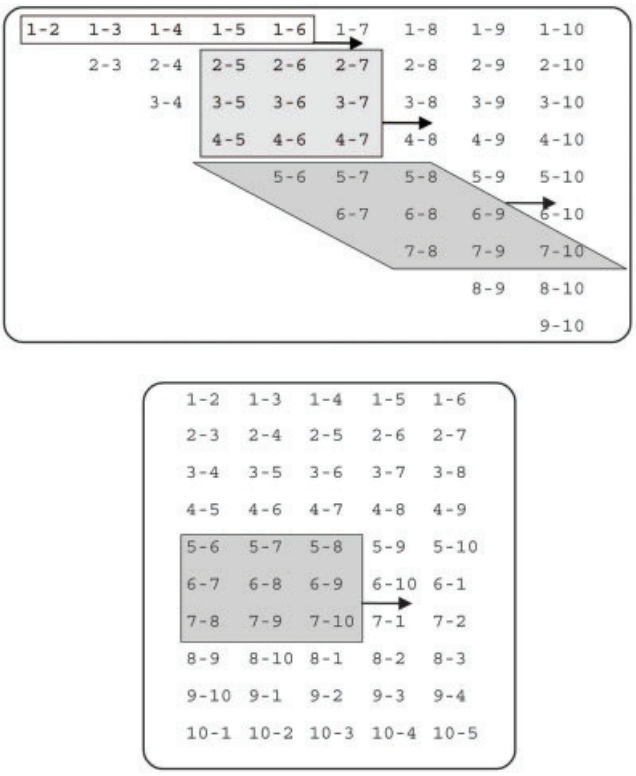

Figure 2. Illustration of the standard double-loop and the improved pairlist algorithm for a set of 10 atoms or charge groups. The standard algorithm scans the triangular atom-pair matrix row by row (top, unshaded box). A quadratic (top, light gray shade) or rhombic (top, dark gray shade) window scans more atom pairs for a given number of atoms loaded into processor cache. The triangular atom-pair matrix may be reordered to become rectangular (bottom), in which case the rhombic window becomes quadratic.

octahedral, or triclinic computational box; possibility of performing multiple-unit-cell simulations);
- temperature control (constraining, weak coupling, Nosé–Hoover or Nosé–Hoover chain; possible coupling of different subsets of degrees of freedom to separate temperature baths);
- pressure control (weak coupling or Andersen–Parrinello–Rahman, isotropic, partially anisotropic, and fully anisotropic coordinate scaling; atom-based or group-based pressure definition);
- long-range electrostatic interactions: straight cutoff truncation, truncation with Poisson–Boltzmann reaction field (RF) correction and lattice-sum (LS) methods, including Ewald summation and particle–particle–particle–mesh (P^3M);
- charge-group based or atom-based cutoff for the nonbonded interactions;
- Grid-based pairlist construction;
- nonphysical interactions: atom-position, atom-distance, dihedral-angle, NOE, and J-value restraints as well as atom-position and atom-distance constraints (SHAKE, M-SHAKE, LINCS);
- enhanced sampling: local elevation MD, replica exchange MD (REMD), and umbrella sampling;
- calculation of free energy changes based on the coupling parameter (λ) approach using thermodynamic integration, slow-growth or one-step perturbation, possibly including soft-core nonbonded interactions;
- path-integral simulation;
- MPI and OMP parallelization.

A number of the new features introduced in GROMOS05 are discussed later. Preexisting features have been described in details elsewhere.[3,4] The functionalities of the pre- and postprocessing programs contained in GROMOS++ are also discussed later. A complete description of the available features will be included in the new GROMOS manuals.

Algorithms

MD Algorithm

The complete MD algorithm based on the leap-frog scheme as implemented in GROMOS is the following.[3] Given initial atomic positions and velocities, which satisfy any given geometrical constraints:

1. Save positions (reset atomic coordinates into the reference computational box in case of periodic boundary conditions) and velocities for later analysis.
2. Remove center of mass motion (if required).
3. Calculate (unconstrained) energies, forces, and virial contribution from the potential energy function (using the nearest image convention in case of periodic boundary conditions). Save these.
4. Enforce any given position constraints by resetting the forces and velocities of positionally constrained atoms to zero.
5. Update the velocities using the leapfrog scheme.
6. Apply temperature coupling (constraining, weak coupling, Nosé–Hoover or Nosé–Hoover chain) by scaling the atom velocities.
7. Update the positions using the leapfrog scheme.
8. Enforce distance constraints (using SHAKE, M-SHAKE, or LINCS) both for positions and velocities, and calculate the corresponding forces and virial contribution. Save these.
9. Calculate the kinetic energy and temperature (possibly on the basis of separate subsets of degrees of freedom).
10. Calculate the pressure (atom-based or group-based pressure definition).
11. Apply pressure scaling (weak coupling or Andersen–Parrinello–Rahman) by scaling atomic positions (isotropic, partially anisotropic, or fully anisotropic scaling).
12. Update the coupling parameter λ for (free energy) simulations involving λ changes (slow growth).
13. Calculate total energies, averages, and fluctuations. Save these.

This sequence is repeated for the required number of simulation steps.

New Features

Spatial Boundary Conditions

Spatial boundary conditions are defined by the shape, size, and orientation of the simulated system, and the nature of the boundary to its surroundings. The GROMOS05 implementation (both PROMD and MD++) admits four types of boundary conditions: (1) vacuum boundary conditions; (2) periodic boundary conditions based on a rectangular box; (3) periodic boundary conditions based on a truncated-octahedral box; (4) periodic boundary conditions based on a triclinic box. In the three latter cases, the system is confined to a (reference) computational box that is surrounded by an infinite number of periodic copies of itself.

When periodic boundary conditions are applied, the shape, size, and orientation of the computational box must be defined. For rectangular and triclinic periodic boundary conditions, this is done by specifying the three edge vectors **a**, **b**, and **c** (defining a right-handed coordinate system) of the reference computational box. For a truncated-octahedral box, these vectors correspond instead to the edges of the cube based on which the truncated octahedron is constructed. In practice, the three vectors are specified by their lengths a, b, and c, the box angles α (between **a** and **b**), β (between **a** and **c**), and γ (between **b** and **c**) they define among each other (all in the range $]0; \pi[$), and the three Euler rotation angles ϕ, θ, and ψ (the two former ones in the range $]-\pi; \pi]$, the latter one in the range $[-\pi/2; \pi/2]$) characterizing the orientation of the box relative to the reference right-handed Cartesian coordinate system ($\mathbf{e}_x$, $\mathbf{e}_y$, $\mathbf{e}_z$). To define the Euler angles, the three edge vectors are used to define a box-linked right-handed Cartesian coordinate system ($\mathbf{e}_{x'}$, $\mathbf{e}_{y'}$, $\mathbf{e}_{z'}$) in the following way: (1) the x'-axis is chosen along and in the direction of **a**; (2) the y'-axis is chosen orthogonal to **a** in the plane defined by **a** and **b**, and oriented in the direction of **b**; (3) the z'-axis is chosen orthogonal to both **a** and **b**, and oriented in the direction of **c**. The reference coordinate system can be rotated onto the box-linked coordinate system by the following series of rotations: (1) a rotation by an angle ϕ around the z-axis; (2) a rotation by an angle θ around the new y-axis; (3) a rotation by an angle ψ around the

new x-axis. The angles ϕ, θ, and ψ thus represent the three Euler rotation angles in a zyx or yaw-pitch-roll convention. The use of a rectangular or truncated-octahedral box requires $\alpha = \beta = \gamma = \pi/2$ and is restricted to nonrotated boxes with $\phi = \theta = \psi = 0$. The use of a truncated-octahedral box also requires $a = b = c$. In the case of vacuum boundary conditions, the system is nonperiodic, and **a**, **b**, and **c** need not be specified.

Based on a general triclinic box in an arbitrary orientation, the position of an atom may be specified through coordinates $\mathbf{r} = (x, y, z)$ within the reference Cartesian coordinate system ($\mathbf{e}_x$, $\mathbf{e}_y$, $\mathbf{e}_z$), or through oblique fractional coordinates $\boldsymbol{\tau} = (u, v, w)$ with reference to the box-edge vectors. The two types of coordinates are related by

$$\mathbf{r} = \underline{\mathbf{L}}\boldsymbol{\tau}, \tag{1}$$

where the matrix $\underline{\mathbf{L}}$ contains the components of **a**, **b**, and **c** in the reference Cartesian coordinate system as its columns. The box volume is

$$V = |\underline{\mathbf{L}}|. \tag{2}$$

This matrix can be decomposed as

$$\underline{\mathbf{L}} = \begin{pmatrix} a_x & b_x & c_x \\ a_y & b_y & c_y \\ a_z & b_z & c_z \end{pmatrix} = \underline{\mathbf{R}}\underline{\mathbf{S}}, \tag{3}$$

where the orthogonal transformation matrix $\underline{\mathbf{R}}$ (rotation between reference and box-linked Cartesian coordinate systems) is given by

$$\underline{\mathbf{R}} = \begin{pmatrix} \cos\theta\cos\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi & \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi \\ \cos\theta\sin\phi & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi \\ -\sin\theta & \sin\psi\cos\theta & \cos\psi\cos\theta \end{pmatrix}, \tag{4}$$

and the transformation matrix $\underline{\mathbf{S}}$ (between box-linked Cartesian coordinates and oblique fractional coordinates) is given by

$$\underline{\mathbf{S}} = \begin{pmatrix} a & b\cos\gamma & c\cos\beta \\ 0 & b\sin\gamma & c\sin\beta\cos\delta \\ 0 & 0 & c\sin\beta\sin\delta \end{pmatrix}, \tag{5}$$

with

$$\cos\delta = \frac{\cos\alpha - \cos\beta\cos\gamma}{\sin\beta\sin\gamma}, \quad \delta \in]0; \pi[. \tag{6}$$

As shown by Bekker,[6] a simulation performed in a truncated-octahedral box can equivalently be performed in a special type of triclinic box, by applying an appropriate coordinate transformation. A possible choice for the edges $\mathbf{a}_t$, $\mathbf{b}_t$, and $\mathbf{c}_t$ of the transformed triclinic box is

$$\mathbf{a}_t = \mathbf{a}, \quad \mathbf{b}_t = (1/2)(\mathbf{a} + \mathbf{b} + \mathbf{c}) \quad \text{and} \quad \mathbf{c}_t = (1/2)(-\mathbf{a} - \mathbf{b} + \mathbf{c}). \tag{7}$$

The corresponding box-edge lengths, box angles, and Euler angles are $a_t = a$, $b_t = c_t = (\sqrt{3}/2)a$, $\alpha_t = \text{acos}(-1/3) \approx 109.5^o$, $\beta_t = \text{acos}(-1/\sqrt{3}) \approx 125.3^o$, $\gamma_t = \text{acos}(1/\sqrt{3}) \approx 54.8^o$, $\phi_t = \theta_t = 0$, and $\psi_t = 45^o$. The mapping of atomic coordinates within a truncated-octahedral box to atomic coordinates within the transformed triclinic box is performed by applying shifts along the $\mathbf{a}_t$, $\mathbf{b}_t$, and $\mathbf{c}_t$ vectors. This formalism is applied for the generalization of grid-based pairlist algorithms and lattice-sum electrostatics to truncated-octahedral boxes. Because the truncated-octahedral case can always be mapped to the triclinic case, subsequent sections will only discuss the case of the triclinic box.

Multiple-Unit-Cell Simulations

Within the GROMOS05 implementation (both PROMD and MD++), it is possible to simulate a periodic computational box (rectangular or triclinic only) consisting of multiple periodic copies of a smaller unit cell (referred to here as subcells). This option may be useful when trying to simulate a single unit cell of a crystal that is too small to allow for the application of a reasonably large cutoff value. The number of subcell boundaries along the three box-edge vectors **a**, **b**, and **c** are M_a, M_b, and M_c, so that the total number of subcells is $M = M_a \cdot M_b \cdot M_c$. In this case, the total number of solute molecules and the total number of solvent molecules must both be integer multiples of M.

Subcell periodicity within the computational box must be fulfilled by the initial coordinates and velocities, and will then be maintained throughout the simulation. The corresponding periodicity constraints on the forces, velocities, and coordinates are checked at each step of the simulation, with reference to the subset of solute and solvent molecules with the smallest sequence numbers. Deviations smaller than user-specified tolerances (numerical drift) are systematically corrected. Deviations larger than the tolerance are reported as an error.

Note that the removal of the center of mass motion, whenever required, is applied to charge groups and solvent molecules gathered in the individual subcells. Note also that the application of the particle–particle–particle–mesh (P^3M) method to evaluate electrostatic interactions will only give rise to exactly periodic forces if the number of P^3M grid subdivisions along each axis is an integer multiple of the corresponding number of subcell boundaries.

In MD++, only a single subcell is simulated. Just for the nonbonded interaction calculation the subcell is multiplied to construct the full box. Energies, forces, and virial contributions need only be calculated for atoms inside the reference subcell, but

Table 1. Properties of Momenta Associated with Rigid-Body Motions in MD or SD Simulations under Vacuum or Periodic Boundary Conditions.

Method	Boundary	$\mathbf{p}_{sys}$	$\mathbf{p}_{box}$	$\mathbf{L}_{sys}$	$\mathbf{L}_{box}$	N_r
MD	Vacuum	Conserved		Conserved		6
MD	Periodic	Infinite	Conserved	Zero	Coupled	3
SD	Vacuum	Coupled		Coupled		0
SD	Periodic	Infinite	Coupled	Zero	Coupled	0

The quantities considered are: $\mathbf{p}_{sys}$ and $\mathbf{p}_{box}$ (linear momentum of the overall system and the computational box, $\mathbf{L}_{sys}$ and $\mathbf{L}_{box}$ (angular momentum of the overall system and the computational box), and N_r (number of uncoupled degrees of freedom associated with rigid-body motions).

those are interacting with all other atoms in the full box. Because of that, fewer (nonbonded and covalent) interactions than in the full box simulation have to be calculated and the positions and velocities are always exactly periodic.

Rigid-Body Motion

The laws of classical mechanics lead to two conserved quantities (besides the total energy): (1) the linear momentum $\mathbf{p}_{sys}$ of the system, and (2) the angular momentum $\mathbf{L}_{sys}$ of the system around its center of mass. In simulations under periodic boundary conditions, the two quantities refer to the infinite periodic system. However, in this case, if the linear momentum $\mathbf{p}_{box}$ of the computational box is also conserved, the corresponding angular momentum $\mathbf{L}_{box}$ is not (because correlated rotational motions in two adjacent boxes exert friction on each other, leading to an exchange of kinetic energy with the other degrees of freedom of the system). Furthermore, the quantity $\mathbf{L}_{sys}$ must vanish (because overall uniform rotation of the infinite periodic system would lead to nonperiodic centrifugal forces). When SD is applied instead of MD, the presence of random and frictional forces couples the system (or box) linear and angular momenta with the other degrees of freedom of the system, so that these quantities are no longer conserved. The inclusion of special (unphysical) forces, such as atom-position restraining or constraining forces on a subset of atoms in the system, may also lead to nonconservation of these quantities. The above observations are summarized in Table 1.

The physical properties of a molecular system are independent of $\mathbf{p}_{sys}$ (or $\mathbf{p}_{box}$). However, for MD simulations under vacuum boundary conditions, they depend on $\mathbf{L}_{sys}$, because the rotation of the system leads to centrifugal forces. For these reasons, in the GROMOS05 implementation (PROMD only), the constraint $\mathbf{p}_{sys} = \mathbf{0}$ (or $\mathbf{p}_{box} = \mathbf{0}$) is imposed at each time step throughout any simulation. In addition, the constraint $\mathbf{L}_{sys} = \mathbf{L}^o_{sys}$, where $\mathbf{L}^o_{sys}$ is a user-specified reference value, is imposed throughout any MD simulation under vacuum boundary conditions. These two constraints will in particular prevent the progressive accumulation of kinetic energy into the uncoupled degrees of freedom due to applying a thermostat by velocity scaling and numerical errors, giving rise to the well-known (and quite unpleasant) "flying ice cube problem."[7–9] In addition roto-translational constraints[10] may be applied to the solute molecule(s) during the simulation.

Instantaneous Temperature and Pressure

The instantaneous observables $\mathcal{T}$ and $\underline{\mathcal{P}}$, and time averages of which determine the system macroscopic temperature T and pressure tensor $\underline{\mathbf{P}}$, are not uniquely defined.[7,11–14] Acceptable alternative definitions differ by any quantity with a vanishing equilibrium average. Note, however, that the corresponding equilibrium fluctuations depend on the specific definition chosen for the instantaneous observable.

In the GROMOS05 implementation (both PROMD and MD++), the instantaneous temperature $\mathcal{T}$ is defined using the (atom-based) internal kinetic energy of the system, as[7]

$$\mathcal{T} = \frac{2}{k_B N_{df}} \mathcal{K}, \tag{8}$$

where k_B is Boltzmann's constant, N_{df} the number of internal (unconstrained) degrees of freedom of the system, and $\mathcal{K}$ its instantaneous internal kinetic energy. The word "internal" is used here to exclude possible contributions from the degrees of freedom that are "external," that is, uncoupled from the system in terms of kinetic energy exchange.[15] In MD simulations, these are the degrees of freedom associated with the system (or box) rigid-body translation and, under vacuum boundary conditions, system rigid-body rotation. The number of internal degrees of freedom is thus calculated as three times the total number N of atoms in the system, minus the number N_c of geometrical constraints, minus the number N_r of external degrees of freedom (Table 1), that is,

$$N_{df} = 3N - N_c - N_r. \tag{9}$$

The instantaneous internal kinetic energy is defined as

$$\mathcal{K} = \frac{1}{2} \sum_{i=1}^{N} m_i \dot{\mathbf{r}}_i^2, \tag{10}$$

where the internal (also called peculiar) velocities $\dot{\mathbf{r}}_i$ are obtained from the real atomic velocities $\dot{\mathbf{r}}_i^o$ by excluding any component along the external degrees of freedom (it is assumed that the velocities $\dot{\mathbf{r}}_i^o$ are already exempt of any component along possible geometrical constraints). Due to the constraints imposed in the

GROMOS05 implementation (PROMD only) on the system total linear and angular momenta, the internal velocities only differ from the real ones when MD is applied under vacuum boundary conditions. In this case, one has

$$\dot{\mathbf{r}}_i = \dot{\mathbf{r}}_i^o - \underline{\mathbf{I}}_{\mathrm{CM}}^{-1}(\mathbf{r}^o)\mathbf{L}_{\mathrm{sys}}^o \times (\mathbf{r}_i^o - \mathbf{r}_{\mathrm{CM}}^o), \tag{11}$$

where $\mathbf{r}_{\mathrm{CM}}^o$ is the (constant) coordinate vector of the system center of mass, $\mathbf{L}_{\mathrm{sys}}^o$ the (constant) system angular momentum about the CM, and $\underline{\mathbf{I}}_{\mathrm{CM}}$ is the (configuration-dependent) inertia tensor of the system relative to the CM. The latter quantity is defined as

$$\underline{\mathbf{I}}_{\mathrm{CM}}(\mathbf{r}^o) = \sum_{i=1}^{N} m_i(\mathbf{r}_i^o - \mathbf{r}_{\mathrm{CM}}^o) \otimes (\mathbf{r}_i^o - \mathbf{r}_{\mathrm{CM}}^o), \tag{12}$$

where $\mathbf{a} \otimes \mathbf{b}$ denotes the tensor with elements μ, ν equal to $a_\mu b_\nu$.

In the GROMOS05 implementation (both PROMD and MD++), the instantaneous pressure tensor $\underline{\mathcal{P}}$ is related to the group-based virial and group-based internal kinetic energy tensor of the system. The word "group-based" refers to a pressure definition excluding virial and kinetic-energy contributions within user-specified groups of (covalently-linked) atoms.[13,14] These groups will be referred to as virial groups. Single atoms can be used as virial groups, in which case an atom-based pressure definition is recovered. The average pressure is not affected by the specific choice of groups, but the pressure fluctuations are. In practice, atom grouping is used to reduce these fluctuations. The pressure is only calculated for systems under periodic boundary conditions. Note also that the contribution of special (nonphysical) forces (e.g., atom-position or atom-distance restraining) to the pressure is not included.

The instantaneous atom-based pressure tensor is computed as

$$\underline{\mathcal{P}}^* = \frac{2}{\mathcal{V}}(\underline{\mathcal{K}}^* - \underline{\mathcal{W}}^*) \tag{13}$$

where

$$\underline{\mathcal{K}}^* = \frac{1}{2}\sum_{i=1}^{N} m_i \dot{\mathbf{r}}_i \otimes \dot{\mathbf{r}}_i, \tag{14}$$

and

$$\underline{\mathcal{W}}^*_{\mu\nu} = \frac{1}{2}\sum_{\lambda} \frac{\partial \mathcal{U}}{\partial L_{\mu\lambda}} L_{\nu\lambda} \tag{15}$$

are the instantaneous atom-based internal kinetic energy and virial tensors, $\mathcal{V}$ and $\mathcal{U}$ being the instantaneous volume and total potential energy of the system, $\underline{\mathbf{L}}$ the matrix defined by eq. (3), and $\dot{\mathbf{r}}_i$ the internal velocities introduced above. The corresponding isotropic (scalar) quantities are related to the tensor quantities through

$$\mathcal{K}^* = \mathrm{Tr}[\underline{\mathcal{K}}^*], \quad \mathcal{W}^* = \mathrm{Tr}[\underline{\mathcal{W}}^*] \quad \text{and} \quad \mathcal{P}^* = (1/3)\mathrm{Tr}[\underline{\mathcal{P}}^*], \tag{16}$$

where Tr returns the trace of a matrix, $\mathcal{K}^*$ is equivalent to $\mathcal{K}$ in eq. (10), and $\mathcal{W}^*$ is defined as

$$\mathcal{W}^* = \frac{3\mathcal{V}}{2}\frac{\partial \mathcal{U}}{\partial \mathcal{V}}. \tag{17}$$

It is possible to show that:[16,17] (1) the contribution to the atom-based virial tensor of a potential energy term that solely depends on the scalar products or determinants defined by a set of interatomic vectors is symmetric; (2) the contribution to the atom-based virial tensor of a potential energy term that solely depends on the angles defined by a set of vectors is (in addition) traceless. The first observation implies that all covalent (bond stretching or constraint, bond-angle bending, proper and improper dihedral angle), and pairwise nonbonded force field terms lead to a symmetric contribution to the atom-based virial. The second observation implies that covalent bond-angle bending as well as proper and improper dihedral angle (but not bond stretching or constraint and pairwise nonbonded) terms lead to a traceless contribution to the atom-based virial (i.e., no contribution to the scalar atom-based pressure). However, these results are generally not valid for the corresponding group-based tensor (see below).

In the special case of a pairwise-additive interaction term $\mathcal{U}_p$ depending on minimum-image interatomic distances and without explicit dependence on the box dimensions (bond stretching or constraint and pairwise nonbonded terms; but not reciprocal-space lattice-sum interactions[13,14]), eq. (15) leads to a virial contribution

$$\underline{\mathcal{W}}_p^* = -\frac{1}{2}\sum_{i}^{N}\sum_{j>i}^{N} \mathbf{F}_{p,ij} \otimes \bar{\mathbf{r}}_{ij}, \tag{18}$$

where $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$ is the vector connecting j to i, $\bar{\mathbf{r}}_{ij}$ the corresponding minimum-image vector, and $\mathbf{F}_{p,ij}$ the pairwise force exerted by atom j on atom i. This equation is easily generalized to interaction terms involving more than two atoms (bond-angle bending, proper and improper dihedral-angle terms). The atom-based virial contribution of all covalent (including bond constraints) and nonbonded (excluding reciprocal-space lattice-sum interactions) terms is calculated using eq. (18) or one of its generalizations.

The GROMOS05 implementation (both PROMD and MD++) includes the possibility of using a group-based pressure definition (corresponding to any arbitrary repartition of atoms into virial groups), instead of the atom-based one. In this case, the intragroup contribution to the kinetic energy as well as the contribution of intragroup forces to the virial are removed from the pressure definition (which affects the fluctuations of this quantity, but not its average value). As shown elsewhere[13,14] (the equations reported therein should be altered by halving the virial and replacing $\mathbf{r}_{ij}$ by $-\mathbf{r}_{ij}$ to match the present conventions), the group-based virial tensor can be calculated from the corresponding atom-based tensor by adding a simple correction term, which depends on the overall atomic forces and on the virial groups definitions. More precisely, the group-based virial tensor is given by

$$\underline{\mathcal{W}} = \underline{\mathcal{W}}^* + \frac{1}{2}\sum_{I\alpha} \mathbf{F}_{I\alpha} \otimes \mathbf{d}_{I\alpha}, \tag{19}$$

where $I\alpha$ denotes atom α in virial group I, $\mathbf{F}_{I\alpha}$ the overall force on atom $I\alpha$, and $\mathbf{d}_{I\alpha}$ the coordinate vector of atom $I\alpha$ relative to the center of mass of the gathered virial group I containing this atom. The "gathered" representation of the virial group is generated by following the atoms as they drift throughout the periodic system. The group-based pressure tensor is then calculated as

$$\underline{\mathcal{P}} = \frac{2}{\mathcal{V}}(\underline{\mathcal{K}} - \underline{\mathcal{W}}), \tag{20}$$

where $\underline{\mathcal{K}}$ is the group-based internal kinetic energy tensor, defined as

$$\underline{\mathcal{K}} = \frac{1}{2}\sum_{I=1}^{N_s}\left(\sum_{\alpha=1}^{N_a(I)} m_i\right)^{-1}\left(\sum_{\alpha=1}^{N_a(I)} m_i\dot{\mathbf{r}}_i\right) \otimes \left(\sum_{\alpha=1}^{N_a(I)} m_i\dot{\mathbf{r}}_i\right), \tag{21}$$

where N_s is the number of virial groups and $N_a(I)$ the number of atoms in virial group I.

Although the atom-based pressure tensor $\underline{\mathcal{P}}^*$ is typically symmetric, this is generally not the case for the group-based pressure tensor $\underline{\mathcal{P}}$ (although the antisymmetric contribution to this tensor should vanish upon time averaging). When applying a barostat algorithm, the antisymmetric component of $\underline{\mathcal{P}}$ should induce an overall rotation of the computational box (which would alter the box angular momentum), while the symmetric component results in a deformation of the box (which conserves the box angular momentum). In practice, the overall rotation of the box is rather a nuisance, and is avoided by symmetrizing the tensor ($\underline{\mathcal{P}} \rightarrow (1/2)[\underline{\mathcal{P}} + {}^t\underline{\mathcal{P}}]$) prior to application of the barostat algorithm,[18] where the "t" presuperscript indicates the transpose of the matrix.

Thermostat Algorithms

MD simulation relies on integrating the classical (Newtonian) equations of motion for a molecular system, and thus, samples a microcanonical (constant-energy) ensemble by default. However, for compatibility with the experiment, it is often desirable to sample configurations from a canonical (constant-temperature) ensemble instead. A modification of the basic MD scheme with the purpose of maintaining the temperature constant (on average) is called a thermostat algorithm.[7] Note that in contrast, SD automatically generates a canonical ensemble, at a temperature determined by the balance between the magnitudes of the random and frictional forces.

In the GROMOS05 implementation (both PROMD and MD++), four different thermostat algorithms are available: (1) temperature constraining (Woodcock thermostat[19]); (2) temperature relaxation by weak coupling (Berendsen thermostat[20]); (3) temperature relaxation by an extended-system method (Nosé–Hoover thermostat[21,22]); (4) temperature relaxation by the Nosé–Hoover chain thermostat.[23] In all cases, the instantaneous temperature $\mathcal{T}$ is calculated as described earlier, and relaxed towards a temperature T_o associated with the heat bath to which the system is coupled. The three latter algorithms also involve the specification of the characteristic time τ for this relaxation. All the above thermostat algorithms rely on a scaling of the atomic velocities after each integration time step. This scaling should only operate on the internal velocities, excluding any component along the external degrees of freedom (see earlier). Due to the constraints imposed in the GROMOS05 implementation on the system total linear and angular momenta, the internal velocities only differ from the real ones when MD is applied under vacuum boundary conditions. In this case, the velocity scaling is applied on the internal velocities $\dot{\mathbf{r}}_i$ and the real velocities $\dot{\mathbf{r}}_i^o$ can be recovered through the inverse of eq. (11), namely

$$\dot{\mathbf{r}}_i^o = \dot{\mathbf{r}}_i + \underline{\mathbf{I}}_{\mathrm{CM}}^{-1}(\mathbf{r}^o)\mathbf{L}_{\mathrm{sys}}^o \times (\mathbf{r}_i^o - \mathbf{r}_{\mathrm{CM}}^o). \tag{22}$$

When simulating molecular systems involving distinct sets of degrees of freedom with either (1) very different characteristic frequencies or (2) very different heating (or cooling) rates caused by algorithmic noise (e.g., electrostatic cutoff, application of atom-distance constraints), the joint coupling of all degrees of freedom to a single thermostat may lead to different effective temperatures for the different sets of degrees of freedom (due to a too slow exchange of kinetic energy). A typical example is the so-called "hot solvent–cold solute problem" in simulations of macromolecules. Because the solvent is often more significantly affected by algorithmic noise (heating due to the use of an electrostatic cutoff), the coupling of the whole system to a single thermostat may cause the average solute temperature to be significantly lower than the average solvent temperature. In the GROMOS05 implementation (both PROMD and MD++), this problem may be alleviated by separately coupling different subsets of degrees of freedom (e.g., solute, counterions, cosolvent, and solvent) to different independent thermostats.

The prototype of most isothermal equations of motion is

$$\ddot{\mathbf{r}}_i(t) = m_i^{-1}\mathbf{F}_i(t) - \gamma(t)\dot{\mathbf{r}}_i(t). \tag{23}$$

The function $\gamma(t)$ controls the heat exchange between the system and the heat bath. A negative value indicates that heat flows to the system, while a positive value indicates a heat flow in the opposite direction. Practical implementations of eq. (23) rely on the stepwise integration of Newton's second law [eq. (23) with $\gamma(t) = 0$], altered by the scaling of the atomic velocities after each iteration step. In the context of the leapfrog integrator[24] used in GROMOS05, this can be written as

$$\begin{aligned}\dot{\mathbf{r}}_i\left(t + \frac{\Delta t}{2}\right) &= \lambda(t;\Delta t)\dot{\mathbf{r}}_i'\left(t + \frac{\Delta t}{2}\right)\\ &= \lambda(t;\Delta t)\left[\dot{\mathbf{r}}_i\left(t - \frac{\Delta t}{2}\right) + m_i^{-1}\mathbf{F}_i(t)\Delta t\right],\end{aligned} \tag{24}$$

where $\lambda(t;\Delta t)$ is a time- and time step-dependent velocity scaling factor. Imposing the constraint $\lambda(t;0) = 1$, one recovers eq. (23) in the limit of an infinitesimal time step Δt, with

$$\gamma(t) = -\left.\frac{\partial\lambda(t;\,\Delta t)}{\partial(\Delta t)}\right|_{\Delta t=0}. \tag{25}$$

The Woodcock thermostat[19] (also known as temperature constraining thermostat; see also the Hoover–Evans thermostat[25,26]) aims at fixing the instantaneous temperature $\mathcal{T}$ exactly at the reference heat-bath value T_o, without allowing for any fluctuations. In this case, the quantity $\lambda(t;\,\Delta t)$ in eq. (24) is found by imposing $\mathcal{T}[t + (\Delta t/2)] = (g/N_{\rm df})T_o$, leading to

$$\lambda(t;\,\Delta t) = \left[\frac{g}{N_{\rm df}}\frac{T_o}{\mathcal{T}'\left(t+\frac{\Delta t}{2}\right)}\right]^{1/2}, \tag{26}$$

where $\mathcal{T}'[t + (\Delta t/2)]$ is the temperature evaluated from the velocities $\dot{\mathbf{r}}_i'[t + (\Delta t/2)]$ in eq. (24). The corresponding quantity $\gamma(t)$ in eq. (23) is given by

$$\gamma(t) = (gk_BT_o)^{-1}\sum_{i=1}^{N}\dot{\mathbf{r}}_i(t)\cdot\mathbf{F}_i(t). \tag{27}$$

Although $g = N_{\rm df}$ seems to be the obvious choice, it turns out that $g = N_{\rm df} - 1$ is the appropriate one for the algorithm to generate a canonical distribution of configurations (although obviously not of momenta) at temperature T_o.[7,21,25] The reason is that constraining the temperature effectively removes one degree of freedom from the system. Note, however, that the absence of kinetic energy fluctuations may lead to inaccurate dynamics, especially in the context of the microscopic systems typically considered in simulations.

The Berendsen thermostat[20] (also known as weak-coupling thermostat) aims at relaxing the instantaneous temperature $\mathcal{T}$ to the reference heat-bath value T_o based on a first-order (weak-coupling) scheme with a characteristic time τ_B, that is, as

$$\dot{\mathcal{T}}(t) = \tau_B^{-1}[T_o - \mathcal{T}(t)]. \tag{28}$$

In this case, the quantity $\lambda(t;\,\Delta t)$ in eq. (24) is found by imposing $\mathcal{T}[t + (\Delta t/2)] = \mathcal{T}[t - (\Delta t/2)] + \tau_B^{-1}\Delta t(g/N_{\rm df})\{T_o - \mathcal{T}[t - (\Delta t/2)]\}$, where, in principle, $g = N_{\rm df}$, leading to

$$\lambda(t;\,\Delta t) = \left\{\frac{\mathcal{T}\left(t-\frac{\Delta t}{2}\right)}{\mathcal{T}'\left(t+\frac{\Delta t}{2}\right)} + \tau_B^{-1}\Delta t\,\frac{\frac{g}{N_{\rm df}}T_o - \mathcal{T}\left(t-\frac{\Delta t}{2}\right)}{\mathcal{T}'\left(t+\frac{\Delta t}{2}\right)}\right\}^{1/2}$$
$$\approx\left\{1 + \tau_B^{-1}\Delta t\left[\frac{\frac{g}{N_{\rm df}}T_o}{\mathcal{T}'\left(t+\frac{\Delta t}{2}\right)} - 1\right]\right\}^{1/2}. \tag{29}$$

In GROMOS05, the algorithm is implemented following the second (approximate) expression. For either of the two expressions, the corresponding quantity $\gamma(t)$ in eq. (23) is given by

$$\gamma(t) = \frac{1}{2}\,\tau_B^{-1}\left[\frac{g}{N_{\rm df}}\frac{T_o}{\mathcal{T}(t)} - 1\right]. \tag{30}$$

In practice, τ_B is used as an empirical parameter to adjust the strength of the coupling to the heat bath. In the limit $\tau_B = \Delta t$, the Berendsen algorithm is equivalent to the Woodcock algorithm (and thus generates a canonical distribution of configurations, but not of momenta). In the limit $\tau_B \to \infty$, the thermostat becomes inactive and the Newton equation of motion is recovered (which samples a microcanonical ensemble). However, except in the former limit (and only for the configurational part), the ensemble generated by the Berendsen equations of motion is not a canonical ensemble.[27]

The Nosé–Hoover thermostat[21,22] aims at relaxing the instantaneous temperature $\mathcal{T}$ to the reference heat-bath value T_o based on an extended-system approach with a characteristic time $\tau_{\rm NH}$. In the original Nosé algorithm,[28] the real system is extended by addition of an artificial $(N_{\rm df} + 1)^{th}$ (positive) dynamical variable s (associated with a "mass" $Q > 0$ as well as a velocity $\dot{s}$), that plays the role of a time-scaling parameter. Through an appropriate choice for the extended-system Lagrangian, a microcanonical MD trajectory in the extended system can be mapped onto a canonical trajectory in the real system. However, the Nosé thermostat leads to sampling of the real-system trajectory at uneven time intervals, which is quite impractical. This inconvenience is alleviated by rewriting the equations of motion in terms of the real-system variables, as was later shown simultaneously by Nosé[21] and Hoover.[22] In the Nosé–Hoover algorithm, the quantity $\gamma(t)$ in eq. (23) is not uniquely determined by the instantaneous microstate of the system, but is a dynamical variable where the derivative is determined by this instantaneous microstate through

$$\dot{\gamma} = -\tau_{\rm NH}^{-2}\frac{\mathcal{T}}{T_o}\left(\frac{g}{N_{\rm df}}\frac{T_o}{\mathcal{T}} - 1\right), \tag{31}$$

where the effective coupling time $\tau_{\rm NH}$ is related to the (less intuitive) effective mass Q in the Nosé thermostat through

$$\tau_{\rm NH} = (N_{\rm df}k_BT_o)^{-1/2}Q^{1/2}. \tag{32}$$

When γ is negative, heat flows from the heat bath into the system due to eq. (23). When the system temperature increases above T_o, the time derivative of γ becomes positive in eq. (31) and the heat flow is progressively reduced (feedback mechanism). Conversely, when γ is positive, heat is removed from the system until the system temperature decreases below T_o and the heat transfer is slowed down. In practice, eq. (31) is discretized (based on the simulation time step Δt) and integrated simultaneously with the equations of motion for the atomic coordinates and velocities based on the leapfrog scheme.

It can be proven[7,22] that the Nosé–Hoover equations of motion sample a canonical ensemble (in both coordinates and momenta) provided that $g = N_{\rm df}$ and that $\tau_{\rm NH}$ is finite, this irrespective of the actual value of $\tau_{\rm NH}$ and of the initial conditions for the atomic velocities and for the γ variable.

In practice $\tau_{\rm NH}$ is used as an empirical parameter to adjust the strength of the coupling to the heat bath. Too large values of $\tau_{\rm NH}$

(loose coupling) may cause a poor temperature control (the limiting case of the Nosé–Hoover thermostat with $\tau_{NH} \to \infty$ and $\gamma(0) = 0$ is MD, which generates a microcanonical ensemble). On the other hand, too small values (tight coupling) may cause high-frequency temperature oscillations leading to the same effect.

The Nosé–Hoover chain thermostat[23] aims at relaxing the instantaneous temperature $\mathcal{T}$ to the reference heat bath value T_o based on a chain of successive thermostat variables. In this case the single thermostat variable γ of the Nosé–Hoover scheme is replaced by a chain of variables applying a thermostat to each other in sequence. This algorithm has been introduced to alleviate the two main drawbacks of the Nosé–Hoover algorithm: (1) the presence of temperature oscillations, and (2) the nonergodicity of the sampling for small or stiff systems, or systems at low temperatures.[22,29–34] The GROMOS05 implementation follows the formalism described in the original article.[23]

Barostat Algorithms

For compatibility with the experiment, it is often desirable to sample configurations from the isothermal–isobaric ensemble (constant temperature and pressure). Thermostat algorithms have been described above. A modification of the basic MD scheme with the purpose of maintaining the pressure constant (on average) is called a barostat algorithm.

The use of a barostat is only applicable to simulations under periodic boundary conditions. In the GROMOS05 implementation (both PROMD and MD++), the various options for the variations for the box parameters (and the associated scaling of atomic coordinates) involved in the use of a barostat are: (1) no variations of the box parameters; (2) isotropic scaling, that is, identical relative variations of the box-edge lengths only; (3) partially anisotropic scaling, that is, independent relative variations of the box-edge lengths only; (4) fully anisotropic scaling, that is, independent variations of all box parameters (box-edge lengths, box angles, and Euler angles). For a truncated-octahedral box, only the first two options are allowed. For a rectangular box, only the first three options are allowed. For a triclinic box, all options are allowed. In the latter case, variations in the box shape are accompanied by variations in the box Euler angles, so as to guarantee that the barostat does not introduce a rigid-body rotational component to the box orientation. Note, however, that the location of the box center of mass is affected by any type of coordinate scaling.

Two different barostat algorithms are available (1) pressure relaxation by weak-coupling (Berendsen barostat[20]); (2) pressure relaxation by extended-system method (Andersen–Parrinello–Rahman barostat[21,28,35–40]).

Pairlist Construction

The evaluation of the nonbonded interactions in GROMOS relies on the application of the twin-range method.[41–44] The GROMOS05 implementation (both PROMD and MD++) of this approach includes an increased amount of flexibility, and relies on the definition of: (1) a short-range pairlist distance R_p; (2) a corresponding cutoff distance $\tilde{R}_p \leq R_p$ (optional); (3) a lower bound for the intermediate-range pairlist distance R_s; (4) a corresponding cutoff distance $\tilde{R}_s \geq R_s$ (optional); (5) an upper bound for the intermediate-range pairlist distance R_l; (6) a corresponding cutoff distance $\tilde{R}_l \leq R_l$ (optional); (7) a short-range pairlist update frequency N_s; (8) an intermediate-range pairlist update frequency N_l. Short-range interactions are computed every time step based on a short-range pairlist containing pairs in the distance range $[0; R_p]$, or a filtered subset of this list corresponding to pairs currently (i.e., at the given timestep) in the distance range $[0; \tilde{R}_p]$. The short-range pairlist is reevaluated every N_s time steps. It can be generated either on the basis of distances between charge groups (groups of covalently linked atoms defined in the system topology) or of distances between individual atoms. In the former case, the filtering (based on the distance $\tilde{R}_p$) may be based either on distances between charge groups or on distances between atoms. In the latter case, only atom-based filtering is possible. Intermediate-range interactions are computed every N_l time steps based on all pairs in the distance range $[R_s; R_l]$, or a filtered subset of these pairs in the distance range $[\tilde{R}_s; \tilde{R}_l]$ at the time of the evaluation of these interactions. Only an atom-based filtering is possible here, and it is only meaningful when the initial set of pairs is generated on the basis of distances between charge groups. The energy, forces, and virial contributions associated with intermediate-range interactions are assumed constant between two updates (i.e., during N_l steps).

The evaluated interaction includes Lennard–Jones and electrostatic components. The latter component may include a reaction-field contribution or the real-space contribution to a lattice-sum method. Note that the real-space contribution to a lattice-sum method may only be computed within the short-range contribution to the interaction.

The pairlist construction may be performed in four different ways: (1) using the standard double-loop algorithm included in the GROMOS96 program[3] (merely extended to include the possibility of an atom-based cutoff and of filtering); (2) using an optimized version that improves processor cache usage; (3) using a grid-based pairlist algorithm introduced recently[45] (PROMD only); (4) using a slight variation of the above grid-based algorithm,[45] which permits easier parallelization and avoids periodicity corrections during the interaction evaluation (MD++ only).

Pairlist construction with optimized processor cache usage. An improved version of the standard double-loop algorithm is implemented in PROMD, which tries to optimize processor cache usage. Figure 2 illustrates the basic idea for a coordinate set of $N = 10$ atoms (or charge groups). In a double loop running over atom indices, the standard algorithm of GROMOS96 scans the atom pair matrix row by row to decide whether a given pair of atoms needs to be included in the pairlist (unshaded box). For very large N, the fast but small processor cache obviously needs to be reloaded many times. If we assume that this cache can hold the coordinates of N_{cache} atoms at a time, then a total of $N_{\text{cache}} - 1$ atom pairs can be treated per cache reload. Scanning the atom-pair matrix with quadratic (light gray shade) or rhombic (dark gray shade) windows is obviously more efficient, because $O(N_{\text{cache}}^2)$ atom pairs can be treated per cache reload. The triangular matrix of atom pairs (Fig. 2, top) can be reordered into a rectangular one by simply aligning every row to the left, chopping off the right half and appending each of its columns to the rows of the lower portion of the matrix

Table 2. Efficiency Comparison of GROMOS96 [FORTRAN, Standard Pairlist Algorithm (STD), and Pairlist Algorithm with Optimized Cache Usage (OPT)] and MD++ [C++; Standard Pairlist Algorithm (STD), and Grid-Based Pairlist Algorithm (GRID)].

		GROMOS96 PROMD		MD++	
		STD	OPT	STD	GRID
Alkane	Single	166	102	214	49 (45)
	Dual	—	61	121	40 (30)
Membrane	Single	67	57	95	55 ( )
	Dual	—	34	60	43 (29)
Protein	Single	175	148	349	140 ( )
	Dual	—	82	187	90 (64)

As test systems liquid alkane (23,328 solute atoms, 9.4 × 9.4 × 9.4 nm cubic box), a membrane [6656 solute atoms, 7383 solvent (SPC water) atoms, 6.2 × 6.2 × 6.9 nm rectangular box], and a protein [2445 solute atoms, 47,472 solvent (SPC water) atoms, 7.9 × 7.9 × 8.3 nm rectangular box] were used. All calculations were done on a dual-processor AMD Athlon MP 248 PC (2000-MHz processor frequency, 512-kb cache, 2-GB RAM). The efficiency was measured running on a single processor and running in parallel on both processors, 250 simulation steps for the alkane and membrane system, 100 steps for the protein. All numbers are in seconds. For MD++, the time spent doing interaction calculations is given in parentheses.

as shown in Figure 2 (bottom). In practice, the same rectangular matrix is more conveniently generated from two coordinate vectors, the second one being twice as long as the first one and containing, consecutively, two identical copies of the first one (i.e., $1, 2, 3, \ldots, N-1, N, 1, 2, 3, \ldots, N-1, N$). To generate the entire rectangular matrix, one simply needs to shift the two vectors with respect to each other, by 1 for the first column, by 2 for the second, and so on. The formerly rhombic window becomes quadratic in the reordered matrix, and all rows are of equal length, which makes the algorithm simpler. Some consideration is necessary to handle the last column adequately, because for even N it contains each atom pair in duplicate. The new algorithm requires somewhat more memory than the traditional implementation as it maintains three copies of the same coordinate set. In principle, however, it can handle a total of $(N_{\text{cache}}+1)^2$/nine atom pairs (rather than $N_{\text{cache}}-1$) in a given amount of fast processor cache. In practice, along with additional code optimization but also additional postprocessing necessary to generate properly ordered pairlists, we observe speedup factors of about 2–3 for the pairlist construction in applications to medium-sized and large systems on personal computers (see Table 2 for overall efficiency; timings are given for complete simulations including pairlist construction and force calculation). A shared-memory OpenMP (www.openmp.org)-style parallel version of the algorithm has also been implemented, which distributes the workload, window by window, over several processors.

Grid-based pairlist construction. PROMD includes a recently introduced[45] grid-based pairlist algorithm that permits the fast construction of cutoff-based nonbonded pairlists in molecular simulations under periodic boundary conditions based on an arbitrary box shape (rectangular, truncated-octahedral, or triclinic). The key features of this algorithm are: (1) the use of a one-dimensional mask array (to determine which grid cells contain interacting atoms) that incorporates the effect of periodicity, and (2) the grouping of adjacent interacting cells of the mask array into stripes, which permits the handling of empty cells with a very low computational overhead. Testing of the algorithm on water systems of different sizes (containing about 2000 to 11,000 molecules) has shown that the method: (1) is about an order of magnitude more efficient compared to a standard (double-loop) algorithm, (2) achieves quasi-linear scaling in the number of atoms, (3) is weakly sensitive in terms of efficiency to the chosen number of grid cells.

MD++ includes a slightly modified version of this grid-based pairlist algorithm following ideas similar to those of a published pairlist algorithm.[46,47] In an effort to reduce the number of nearest image determinations during the pairlist generation (or filtering) and the nonbonded force calculation, the system gets extended on all sides before the pairlist construction. The additional atom or charge-group positions are obtained by simple shifts of the original positions by the box vectors. To allow for more efficient (distributed-memory) parallelization and to save money, the central computational box is divided into N layers. Each of the P parallel processes only has to extend over N/P layers. After every extension, the atom pairs consisting of one atom within the layer and a second atom from one of the above (not extended) layers are added to the respective pairlist (using a one-dimensional mask array). Filtering or energy and force evaluation can then be carried out right away (without nearest image determinations owing to the preshifted atomic positions), or at a later stage with the information on the shift vectors encoded into the pairlist, thus enabling a fast reconstruction of the shifted positions.

Reaction-Field Electrostatics

When cutoff truncation is applied to the Coulombic interactions within a molecular system, the mean effect of the omitted electrostatic interactions beyond the (long-range) cutoff distance R_l may be approximately reintroduced through a so-called reaction-field correction term.[48–51] This approximation relies on assuming that the medium beyond the cutoff sphere of each particle (i.e., beyond a specified distance R_{RF}, typically set equal to R_l) is a linearized Poisson–Boltzmann continuum characterized by a relative dielectric permittivity ε and an inverse Debye screening length κ. In the present context, these two parameters may be combined into an effective permittivity[51]

$$\varepsilon_{\text{RF}} = \left[1 + \frac{(\kappa R_{\text{RF}})^2}{2(\kappa R_{\text{RF}}+1)}\right]\varepsilon. \tag{33}$$

In the GROMOS05 implementation (both PROMD and MD++), the corresponding overall electrostatic energy (Coulomb plus reaction-field term) is then written[52]

$$\mathcal{U}_{\text{el}}^{\text{CB+RF}} = \frac{1}{4\pi\varepsilon_o}\left\{\sum_i \sum_{j>i,\, j\notin \text{excl}(i),\, \bar{r}_{ij}<R_l} q_i q_j \left(\bar{r}_{ij}^{-1} + \frac{2(\varepsilon_{\text{RF}}-1)}{2\varepsilon_{\text{RF}}+1}\frac{\bar{r}_{ij}^2}{2R_{\text{RF}}^3}\right.\right.$$

$$-\frac{3\varepsilon_{\mathrm{RF}}}{2\varepsilon_{\mathrm{RF}}+1}\frac{1}{R_{\mathrm{RF}}}\Bigg)+\sum_i\sum_{j>i,j\in\mathrm{excl}(i)}q_iq_j\Bigg(\frac{2(\varepsilon_{\mathrm{RF}}-1)}{2\varepsilon_{\mathrm{RF}}+1}\frac{\bar{r}_{ij}^2}{2R_{\mathrm{RF}}^3}$$

$$-\frac{3\varepsilon_{\mathrm{RF}}}{2\varepsilon_{\mathrm{RF}}+1}\frac{1}{R_{\mathrm{RF}}}\Bigg)-\frac{1}{2}\frac{3\varepsilon_{\mathrm{RF}}}{2\varepsilon_{\mathrm{RF}}+1}\frac{1}{R_{\mathrm{RF}}}\left[\sum_i q_i^2-\varepsilon_{\mathrm{RF}}^{-1}\left(\sum_i q_i\right)^2\right]\Bigg\}, \tag{34}$$

where ε_o is the permittivity of vacuum, $\bar{\mathbf{r}}_{ij}$ is the minimum-image vector corresponding to $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$, and excl($i$) denotes the exclusion list of atom i (including its first and second covalent neighbors; the distance between any two excluded atoms is assumed to be smaller than R_l). Note that current simulation programs (e.g., GROMOS96[3] and GROMACS[53]) typically restrict the implementation of the reaction-field method to the first term in eq. (34). The second term is explicitly included here because excluded neighbors should only be exempted from the direct (Coulombic) interaction and not from the solvent-mediated (reaction-field) interaction.[52] The form of the third term has been chosen for consistency in the context of small molecules (compared to the cutoff radius and box size). For such a small molecule (or ion) gathered by periodicity around its center, $\bar{r}_{ij}$ can be replaced by r_{ij} and the cutoff truncation involved in the first summation of eq. (34) can be omitted. In this case, it can be shown[52] that the reaction-field contribution to $\mathcal{U}_{\mathrm{el}}^{\mathrm{CB+RF}}$ for a neutral molecule matches the correct Onsager expression for the solvation of a dipolar molecule in a spherical cavity[54] (for $\kappa = 0$). For a monoatomic ion, the last term can also be shown[52] to represent a (first-order) correction to the error in solvation free energy caused by the use of effective (non-Coulombic) interactions. Intuitively, this last term may be interpreted as the reversible work required to individually charge the atoms when they are at infinite separation. This contribution only affects the energy of the system, but does not induce atomic forces. However, it may be essential to include it in free-energy calculations involving alterations of the atomic partial charges and comparisons between different media (ε).

Lattice-Sum Electrostatics

Lattice-sum methods for evaluating electrostatic interactions in simulations under periodic boundary conditions (only available in PROMD) rely on two key principles: (1) the treatment of electrostatic interactions as exactly periodic within the periodic system.[55–58] (2) the splitting of these interactions into a short-range component, evaluated by direct summation over atom pairs, and a long-range component, evaluated by Fourier series.[56,59,60]

From a physical point of view, the straight application of lattice-sum methods (with so-called tinfoil boundary conditions, that is, omitting any extrinsic potential contribution) implicitly relies on four modifications of electrostatic interactions compared to the naive picture of assembling an infinite number of copies of the reference computational box in vacuum:[60] (1) inclusion of a homogeneous background charge density within the infinite periodic system (of magnitude related to the net charge of the computational box), that enforces a vanishing net charge in the system;[55] (2) inclusion of a surface charge (with a distribution related to the box dipole moment relative to its center) at the surface of the infinite periodic system, that enforces a vanishing average electric field within the system; (3) inclusion of a homogeneous surface dipole layer (of magnitude related to the trace of the box quadrupole moment relative to its center) at the surface of the infinite periodic system, that enforces a vanishing average potential within the system; (4) suppression of the orientational preferences resulting from a fluid–vacuum interface at the surface of the system, which is effectively equivalent to the inclusion of a compensating homogeneous dipole layer at the surface of the infinite periodic system. These physical modifications have subtle consequences on the simulated properties of molecular systems, which can be altered (on the basis of physical arguments) by the introduction of a so-called extrinsic potential contribution (consisting of a uniform electric field and a constant potential offset). However, the appropriate form for this extrinsic term is still matter of considerable debate.[56,58,60–82] The PROMD implementation for this term is discussed elsewhere.[60]

The splitting of the electrostatic interactions into a short-range and a long-range component relies on the use of a charging-shaping function $a^{-3}\gamma(a^{-1}r)$ of width a. The charge-shaping functions implemented in PROMD are the Gaussian and optimized truncated polynomials of orders 0 to 10,[59] as listed in Table 3. The specific choice of a function must be made before compiling the program (macro definition).

The two lattice-sum methods available differ in the way they evaluate the reciprocal-space interactions.[56] The Ewald method[83] is based on a direct summation over reciprocal-space lattice vectors, while the particle–particle–particle–mesh (P^3M) method[84] makes use of grid discretization and fast Fourier transform (FFT) algorithms. In the PROMD implementation, both lattice-sum methods can also be applied to emulate truncated electrostatic interactions with a reaction-field correction, according to the lattice sum emulated reaction field (LSERF) scheme.[60]

Reciprocal lattice. In the triclinic case, the reciprocal-lattice vectors $\tilde{\mathbf{a}}$, $\tilde{\mathbf{b}}$, and $\tilde{\mathbf{c}}$ associated with the box edge vectors **a**, **b**, and **c** are defined by

$$\tilde{\mathbf{a}} = V^{-1}\mathbf{b}\times\mathbf{c}, \quad \tilde{\mathbf{b}} = V^{-1}\mathbf{c}\times\mathbf{a} \quad \text{and} \quad \tilde{\mathbf{c}} = V^{-1}\mathbf{a}\times\mathbf{b}, \tag{35}$$

where V is the box volume. The matrix containing in its columns the components of $\tilde{\mathbf{a}}$, $\tilde{\mathbf{b}}$, and $\tilde{\mathbf{c}}$ in the reference Cartesian coordinate system ($\mathbf{e}_x$, $\mathbf{e}_y$, $\mathbf{e}_z$) is easily shown to be

$$\begin{pmatrix}\tilde{a}_x & \tilde{b}_x & \tilde{c}_x\\ \tilde{a}_y & \tilde{b}_y & \tilde{c}_y\\ \tilde{a}_z & \tilde{b}_z & \tilde{c}_z\end{pmatrix} = {}^t\underline{\mathbf{L}}^{-1} = \underline{\mathbf{R}}^t\underline{\mathbf{S}}^{-1}. \tag{36}$$

A reciprocal-space vector **k** is defined by

$$\mathbf{k} = 2\pi(l_a\tilde{\mathbf{a}} + l_b\tilde{\mathbf{b}} + l_c\tilde{\mathbf{c}}), \tag{37}$$

where $\mathbf{l} = (l_a, l_b, l_c)$ is a vector with integer (positive or negative) components. Based on a general triclinic box in an arbitrary orientation, a reciprocal-space vector may be specified through the integer vector **l**, through oblique fractional coordinates $\boldsymbol{\chi} = (\chi_u, \chi_v, \chi_w)$ with reference to the reciprocal-lattice vectors, or through

Table 3. Charge-Shaping Functions Available in PROMD.

N_γ	m	$\pi\gamma(\xi)$
−1	∞	$\pi^{-1/2}e^{-\xi^2}$
0	0	$(3/4)H(1-\xi)$
1	1	$3(1-\xi)H(1-\xi)$
2	2	$(15/2)(1-\xi)^2H(1-\xi)$
3	2	$(15/4)(1-\xi)^2(1+2\xi)H(1-\xi)$
4	3	$(105/16)(1-\xi)^3(3\xi+1)H(1-\xi)$
5	4	$(21/2)(1-\xi)^4(4\xi+1)H(1-\xi)$
6	4	$(63/8)(1-\xi)^4(5\xi^2+4\xi+1)H(1-\xi)$
7	5	$(45/4)(1-\xi)^5(8\xi^2+5\xi+1)H(1-\xi)$
8	6	$(165/32)(1-\xi)^6(35\xi^2+18\xi+3)H(1-\xi)$
9	6	$(165/64)(1-\xi)^6(64\xi^3+69\xi^2+30\xi+5)H(1-\xi)$
10	7	$(2145/128)(1-\xi)^7(21\xi^3+19\xi^2+7\xi+1)H(1-\xi)$

N_γ: reference code of the function ($N_\gamma = 0 \ldots 10$: Optimal TP-Function[59] of order N_γ; $N_\gamma = -1$: Gaussian); m: convergence rate of $\hat{\gamma}(\kappa)$ toward zero (convergence is as $\kappa^{-(m+2)}$ when $\kappa \to \infty$; see eq. (43), where $\kappa = ak$); $\pi\gamma(\xi)$: charge-shaping function amplified by π (the actual shaping function is $a^{-3}\gamma(a^{-1}r)$); $H(\xi)$: Heaviside function ($H(\xi) = 1$ when $\xi \geq 0$, zero otherwise). Note that the Gaussian function is infinite-ranged, while all other functions vanish for $\xi \geq 1$.

coordinates $\mathbf{k} = (k_x, k_y, k_z)$ within the reference Cartesian coordinate system. The different coordinates are related through

$$\boldsymbol{\chi} = 2\pi\mathbf{l} \quad \text{and} \quad \mathbf{k} = {}^t\underline{\mathbf{L}}^{-1}\boldsymbol{\chi} = 2\pi{}^t\underline{\mathbf{L}}^{-1}\mathbf{l}. \tag{38}$$

Scalar products between real- and reciprocal-space vectors can be formulated equivalently in the different coordinate representations, that is,

$$\mathbf{k}\cdot\mathbf{r} = \boldsymbol{\chi}\cdot\boldsymbol{\tau}. \tag{39}$$

Definitions. The following discussion considers a periodic system of N_q charges q_i at positions $\mathbf{r}_i$ within a general triclinic computational box with arbitrary orientation. It is convenient to define the box overall charge

$$S = \sum_{i=1}^{N_q} q_i, \tag{40}$$

and the box overall square charge

$$\tilde{S}^2 = \sum_{i=1}^{N_q} q_i^2. \tag{41}$$

Lattice-sum methods rely on the use of the charge-shaping function $a^{-3}\gamma(a^{-1}r)$ of width a (Table 3) to split the electrostatic potential into a real-space contribution and a reciprocal-space contribution, plus a constant. The charge-shaping function is normalized to satisfy the condition

$$4\pi a^{-3}\int_0^\infty drr^2\gamma(a^{-1}r) = 1. \tag{42}$$

The following definitions are related to the charge-shaping function. The Fourier coefficients of (a lattice sum of) the charge-shaping function are given by

$$\hat{\gamma}(ak) = \begin{cases} 4\pi k^{-1}a^{-3}\displaystyle\int_0^\infty drr\,\sin(kr)\gamma(a^{-1}r) & \text{for } k \neq 0 \\ 1 & \text{for } k = 0 \end{cases}. \tag{43}$$

The switch function $\eta(a^{-1}r)$ associated with the charge-shaping function is defined by

$$\eta(a^{-1}r) = 4\pi a^{-3}\int_r^\infty d\rho\rho(\rho - r)\gamma(a^{-1}\rho). \tag{44}$$

Finally, the constants A_1, A_2, and A_3 are defined as

$$A_1 = -4\pi V^{-1}\int_0^\infty drr\eta(a^{-1}r), \tag{45}$$

$$A_2 = 4\pi V^{-1}\sum_{\mathbf{l}\in\mathbb{Z}^3, l\neq 0} k^{-2}\hat{\gamma}(ak), \tag{46}$$

$$A_3 = \lim_{r\to 0}\left[\sum_{\mathbf{n}\in\mathbb{Z}^3}\|\mathbf{r} + \underline{\mathbf{L}}\mathbf{n}\|^{-1}\eta(a^{-1}\|\mathbf{r} + \underline{\mathbf{L}}\mathbf{n}\|) - r^{-1}\right]. \tag{47}$$

In the (common) case where

$$\gamma(a^{-1}r) = 0 \quad \text{for} \quad r \geq \min\{L_x, L_y, L_z\} \tag{48}$$

is (exactly or approximately) valid, eq. (47) becomes

Table 4. Switch Functions Corresponding to the Charge-Shaping Functions Available in PROMD (see Table 3).

N_γ	$\eta(\xi)$
−1	erfc(ξ)
0	$(1/2)(1-\xi)^2(\xi+2)H(1-\xi)$
1	$(1-\xi)^3(\xi+1)H(1-\xi)$
2	$(1/2)(1-\xi)^4(3\xi+2)H(1-\xi)$
3	$(1/4)(1-\xi)^4(4\xi^2+7\xi+4)H(1-\xi)$
4	$(1/8)(1-\xi)^5(15\xi^2+19\xi+8)H(1-\xi)$
5	$(1-\xi)^6(3\xi^2+3\xi+1)H(1-\xi)$
6	$(1/16)(1-\xi)^6(35\xi^3+66\xi^2+51\xi+16)H(1-\xi)$
7	$(1/8)(1-\xi)^7(32\xi^3+49\xi^2+31\xi+8)H(1-\xi)$
8	$(1/16)(1-\xi)^8(105\xi^3+136\xi^2+73\xi+16)H(1-\xi)$
9	$(1/32)(1-\xi)^8(160\xi^4+335\xi^3+312\xi^2+151\xi+32)H(1-\xi)$
10	$(1/128)(1-\xi)^9(1155\xi^4+2075\xi^3+1665\xi^2+697\xi+128)H(1-\xi)$

N_γ: reference code of the function; $\eta(\xi)$: switch function (the real-space interaction function is $r^{-1}\eta(a^{-1}r)$); erfc: complementary error function.

$$A_3 = \lim_{r\to 0} r^{-1}[\eta(a^{-1}r) - 1]. \tag{49}$$

In this case, A_1 and A_3 have analytical expressions for a given charge-shaping function. The above derived quantities[56,59] are listed in Tables 4, 5, and 6 for the charge-shaping functions of Table 3.

Electrostatic interaction energy. The electrostatic reversible-charging (free) energy $\mathcal{U}_{\mathrm{el}}^{\mathrm{LS}}$ of the periodic system of charges under tinfoil boundary conditions can be computed as[56]

$$\mathcal{U}_{\mathrm{el}}^{\mathrm{LS}} = \mathcal{U}_\gamma + \mathcal{U}_\eta + \mathcal{U}_A + \mathcal{U}_{\mathrm{slf,}} \tag{50}$$

with

$$\mathcal{U}_\gamma = (2\varepsilon_o V)^{-1} \sum_{i=1}^{N_q}\sum_{j=1}^{N_q} q_i q_j \sum_{\mathbf{l}\in\mathbb{Z}^3, l\neq 0} k^{-2}\hat{\gamma}(ak)\cos(\mathbf{k}\cdot\mathbf{r}_{ij}), \tag{51}$$

$$\mathcal{U}_\eta = (4\pi\varepsilon_o)^{-1} \sum_{i=1}^{N_q}\sum_{j>i}^{N_q} q_i q_j \sum_{\mathbf{n}\in\mathbb{Z}^3} \|\mathbf{r}_{ij} + \underline{\mathbf{L}}\mathbf{n}\|^{-1}\eta(a^{-1}\|\mathbf{r}_{ij} + \underline{\mathbf{L}}\mathbf{n}\|), \tag{52}$$

$$\mathcal{U}_A = (8\pi\varepsilon_o)^{-1}[A_1 S^2 - (A_1 + A_2)\tilde{S}^2], \tag{53}$$

Table 5. Fourier Coefficients Corresponding to the Charge-Shaping Functions Available in PROMD (see Table 3).

N_γ	$\kappa^{N_\gamma+3}\hat{\gamma}(\kappa)$
−1	$\kappa^2 e^{-\kappa^2/4}$
0	$3[-\kappa C + S]$
1	$12[2 - 2C - \kappa S]$
2	$60[2\kappa + \kappa C - 3S]$
3	$90[8 + (\kappa^2 - 8)C - 5\kappa S]$
4	$630[8\kappa + 7\kappa C + (\kappa^2 - 15)S]$
5	$5040[4(\kappa^2 - 6) - (\kappa^2 - 24)C + 9\kappa S]$
6	$7560[48\kappa - (\kappa^2 - 57)\kappa C + 3(4\kappa^2 - 35)s]$
7	$75600[24(\kappa^2 - 8) - 3(5\kappa^2 - 64)C - (\kappa^2 - 87)\kappa S]$
8	$831600[8\kappa(\kappa^2 - 24) + (\kappa^2 - 123)\kappa C - (18\kappa^2 - 315)S]$
9	$1247400[192(\kappa^2 - 10) + (\kappa^4 - 207\kappa^2 + 1920)C - (22\kappa^2 - 975)\kappa S]$
10	$16216200[64\kappa(\kappa^2 - 30) + (26\kappa^2 - 1545)\kappa C + (\kappa^4 - 285\kappa^2 + 3465)S]$

N_γ: reference code of the function; $\kappa^{N_\gamma+3}\hat{\gamma}(\kappa)$: Fourier coefficient amplified by $\kappa^{N_\gamma+3}$ (the actual Fourier coefficient is $\hat{\gamma}(ak)$); C: short notation for $\cos(\kappa)$; S: short notation for $\sin(\kappa)$.

Table 6. A-Constants Corresponding to the Charge-Shaping Functions Available in PROMD (See Table 3).

N_γ	$-V\pi^{-1}a^{-2}A_1$	$-aA_3$
−1	1	$2\pi^{-1/2}$
0	2/5	3/2
1	4/15	2
2	4/21	5/2
3	3/14	9/4
4	1/6	21/8
5	2/15	3
6	8/55	45/16
7	4/33	25/8
8	4/39	55/16
9	10/91	105/32
10	2/21	455/128

N_γ: reference code of the function; $-V\pi^{-1}a^{-2}A_1$: constant A_1 amplified by $-V\pi^{-1}a^{-2}$; $-aA_3$: constant A_3 amplified by $-a$. The results for A_3 are derived assuming the validity of eq. (48).

and

$$\mathcal{U}_{\text{slf}} = (8\pi\varepsilon_o)^{-1}(A_1 + A_2 + A_3)\tilde{S}^2. \tag{54}$$

The interpretation of the different contributions to $\mathcal{U}_{\text{el}}$ in eq. (50) is given below, with reference to the following terminology for charge densities (which can be added to or subtracted from one another): point charge (p), γ-shaped charge (γ), and homogeneous background charge (b). The term $\mathcal{U}_\gamma$ represents the electrostatic energy (including self-interaction) of a set of ($p - b$)-charges of magnitude $\{q_i\}$ at positions $\{\mathbf{r}_i\}$ in the potential generated by the corresponding periodic system of ($\gamma - b$)-charges. Because this potential is a nonsingular and generally smooth function of position, $\mathcal{U}_\gamma$ is conveniently evaluated in reciprocal space, using the Ewald method[83] or the P^{3}M method.[84] The term $\mathcal{U}_\eta$ represents the electrostatic energy (excluding self interaction) of a set of ($p - b$)-charges of magnitude $\{q_i\}$ at positions $\{\mathbf{r}_i\}$ in the potential generated by the corresponding periodic system of ($p - \gamma$)-charges. With an appropriate choice of charge-shaping function, the function $\eta(a^{-1}r)$ can be made a quickly decreasing function of distance, in which case $\mathcal{U}_\eta$ is conveniently evaluated by direct (real-space) summation over the charge pairs. The terms $\mathcal{U}_A$ and $\mathcal{U}_{\text{slf}}$ are configuration-independent. The term $\mathcal{U}_A$ eliminates the self-energies present in $\mathcal{U}_\gamma$ and contains a small correction due to the constraint of zero average potential within the periodic system. The term $\mathcal{U}_{\text{slf}}$ accounts for the self-energy of set of ($p - b$)-charges of magnitude $\{q_i\}$ in the potential generated by the corresponding periodic system of ($p - b$)-charges (Wigner term[83–87]).

Constant and self-energy terms. The constant term $\mathcal{U}_A$ and the self-energy term $\mathcal{U}_{\text{slf}}$ are given by eqs. (53) and (54), respectively, where the A-constants are defined by eqs. (45), (46), and (47) or (49). Note that these terms give rise to no force contribution (but a virial contribution).

In the general case, the constant A_2 must be computed numerically by direct summation. In the PROMD implementation, this is done by evaluating

$$A_2(l_{\max}) = 4\pi V^{-1} \sum_{\mathbf{l}\in\mathbb{Z}^3, l\neq 0, l_x, l_y, l_z \le l_{\max}} k^{-2}\hat{\gamma}(ak) \tag{55}$$

for increasing values of $l_{\max}$, until a user-specified relative tolerance is reached. In the specific case of a cubic unit cell of edge L, this evaluation is replaced by[49,87] $A_2 \approx \xi_{\text{EW}}L^{-1} - A_1 - A_3$ with $\xi_{\text{EW}} \approx -2.83792748$. The calculation of A_2 with a reasonably high precision is somewhat expensive and may be either (1) omitted; (2) performed once at the beginning of the simulation; (3) performed whenever an energy output is required; (4) performed every step. With the first choice, A_2 is set to zero and the A_2 contributions are omitted in the evaluation of both $\mathcal{U}_A$ and $\mathcal{U}_{\text{slf}}$ [eqs. (53) and (54)]. As a consequence, the splitting between pairwise and self-contributions becomes arbitrary, but the sum of the two quantities (and thus the overall electrostatic energy) remains correct (within the approximation $A_2 \approx \tilde{A}_2$, see below). The second choice is intended for constant-volume simulations. The two last choices are for constant-pressure simulations, the latter one being the more accurate (but also computationally more expensive).

The quantity A_2 calculated through eq. (55) represents the exact (in the limit of large $l_{\max}$) value of A_2 used in the calculation of $\mathcal{U}_{\text{slf}}$. However, because the reciprocal-space interaction energy is evaluated with a finite precision (i.e., through the Ewald or P^{3}M method), this value of A_2 will only approximately remove the reciprocal-space self-energy when included in $\mathcal{U}_A$. Although for most practical purposes, this approximation is sufficient, it is possible to compute a more accurate method-dependent value $\tilde{A}_2$ to be used in the evaluation of $\mathcal{U}_A$, that is,

$$\mathcal{U}_A = (8\pi\varepsilon_o)^{-1}[A_1 S^2 - (A_1 + \tilde{A}_2)\tilde{S}^2], \tag{56}$$

The calculation of this quantity is feasible for either the Ewald or P^{3}M method, as detailed elsewhere.[13]

Real-space contribution. In the most general form, the real-space contribution to the electrostatic interactions is given by eq. (52), together with the corresponding forces and virial contribution. In practice, it is assumed that the charge-shaping function satisfies the condition

$$\gamma(a^{-1}r) = 0 \quad \text{for} \quad r \ge R_p \quad \text{with} \quad R_p \le (1/2)\min\{L_x, L_y, L_z\}, \tag{57}$$

where R_p is the short-range cutoff distance applied to real-space interactions, which implies that the switch function $\eta(a^{-1}r)$ also vanishes beyond R_p. In this case, and taking into account that Coulombic interaction between minimum-image pairs of excluded covalent neighbors should be removed, one may rewrite eq. (52) as

$$\mathcal{U}_\eta = (4\pi\varepsilon_o)^{-1} \sum_{i=1}^{N_q} \sum_{j>i, j\notin\text{excl}(i), \bar{r}_{ij}<R_p} q_i q_j \bar{r}_{ij}^{-1}\eta(a^{-1}\bar{r}_{ij}) + (4\pi\varepsilon_o)^{-1} \sum_{i=1}^{N_q} \sum_{j>i, j\in\text{excl}(i)} q_i q_j \bar{r}_{ij}^{-1}[\eta(a^{-1}\bar{r}_{ij}) - 1], \tag{58}$$

where the distance between any two excluded atoms is assumed to be smaller than R_p. When using a truncated-polynomial charge-shaping function,[59] the evaluation of $\mathcal{U}_\eta$ is exact provided that all atom pairs within a distance smaller than a are included into the short-range pairlist at any time step. When using a Gaussian function, it is always approximate (in practice $a \approx R_p/3$ is a reasonable choice).

Reciprocal-space contribution via Ewald. In the Ewald method,[83] the reciprocal-space contribution $\mathcal{U}_\gamma$ defined by eq. (51), as well as the corresponding forces and virial, are evaluated by direct summation over reciprocal-lattice vectors.

For improved computational speed the triple-sum (over $\mathbf{l}$, i, and j) in eq. (51) is rewritten as a double-sum (over $\mathbf{l}$ and i) and truncated at a given reciprocal-space cutoff l_c, as

$$^{\mathcal{Q}}\mathcal{U}_{\gamma} = (2\varepsilon_o V)^{-1} \sum_{l<l_c} k^{-2}\hat{\gamma}(ak)[C^2(\mathbf{k}) + S^2(\mathbf{k})], \quad (59)$$

with the definitions

$$C(\mathbf{k}) = \sum_{i=1}^{N_q} q_i \cos \mathbf{k}\cdot\mathbf{r}_i \quad \text{and} \quad S(\mathbf{k}) = \sum_{i=1}^{N_q} q_i \sin \mathbf{k}\cdot\mathbf{r}_i. \quad (60)$$

A further increase in computational efficiency can be obtained by noting that the terms in the **l**-sum involved in eq. (59) are invariant upon changing **k** to $-\mathbf{k}$. Thus, the summation can be restricted to a half-space and the resulting energies, forces, and virial contribution multiplied by 2. In the case of a rectangular box, further symmetry considerations allow to restrict the summation to a single octant.

Reciprocal-space contribution via P³M. In the P³M method,[84] the reciprocal-space contribution $^{\mathcal{Q}}\mathcal{U}_{\gamma}$ defined by eq. (51), as well as the corresponding forces and virial, are evaluated by discretization of the triclinic computational box by means of a grid (mesh) and use of a FFT algorithm. The number of grid subdivisions along each of the box axes must be even.

The algorithm consists of six steps:[56] (1) assignment of the charge density associated with the atomic partial charges to the grid points by means of an assignment function; (2) conversion of the charge density grid to reciprocal space by means of a three-dimensional fast Fourier transform (3D-FFT); (3) solution for the reciprocal-space electrostatic potential via multiplication by an optimized influence function; (4) conversion of the reciprocal-space potential grid to real space by means of a 3D-FFT; (5) evaluation of the electrostatic field on the grid by means of a finite-difference operator; (6) interpolation of the field at the location of the atomic partial charges by means of the same assignment function used in the first step. In the *ik*-differentiation variant,[87,88] the fifth and sixth steps are replaced by: (5) evaluation of the reciprocal-space field through multiplication by $i\mathbf{k}$; (6) conversion of the reciprocal-space field grid to real space by means of three 3D-FFTs, one for each field component.

The influence function describes the electrostatic potential generated at the different grid points by a unit $(\gamma - b)$-charge at the origin. It is stored in the form of its corresponding value at each of the reciprocal-space grid points. A key to the accuracy of the algorithm is to preoptimize this function so that it compensates for errors inherent to the discretization process, the use of an approximate assignment function, and the use of an approximate finite-difference operator.[84] When the virial is to be calculated or when the box dimensions may vary in the course of a simulation, six grids are computed simultaneously, containing the relevant derivatives of the optimal influence function with respect to the box parameters.[13] The optimization of the influence function (and the evaluation of its derivatives when required) is computationally expensive. In simulations without variations of the box parameters, however, this function is constant (as well as its derivatives), and the calculation needs to be performed only once at the beginning of a simulation. In simulations involving a variation of the box parameters, the accuracy of the influence function may progressively deteriorate with time as the box changes shape and size. Two mechanisms are then used to improve the accuracy of the current influence function at reasonable computational costs. First, the derivative information computed together with the optimized influence function is used to apply a first-order correction to the current influence function upon variation of the box parameters.[13] Second, the accuracy of the algorithm[88] may be reevaluated at periodic intervals, and a reoptimization of the influence function (and recalculation of its derivatives) undertaken when this accuracy falls below a user-specified threshold.

In the following paragraphs, the P³M algorithm is described in more details.

A general triclinic box may be discretized by means of a grid G, defined by the number of subdivisions N_a, N_b, and N_c along the **a**, **b**, and **c** box-edge vectors. It will be convenient to introduce the diagonal matrix $\underline{\mathbf{N}}$ with elements N_a, N_b, and N_c. The matrix $\underline{\mathbf{H}}$ is then defined as

$$\underline{\mathbf{H}} = \begin{pmatrix} N_a^{-1}a_x & N_b^{-1}b_x & N_c^{-1}c_x \\ N_a^{-1}a_y & N_b^{-1}b_y & N_c^{-1}c_y \\ N_a^{-1}a_z & N_b^{-1}b_z & N_c^{-1}c_z \end{pmatrix} = \underline{\mathbf{L}}\underline{\mathbf{N}}^{-1}. \quad (61)$$

The volume of a grid cell is noted $V_G = |\underline{\mathbf{H}}|$.

Each point of the real-space grid G is associated with an index $\mathbf{n} = (n_a, n_b, n_c)$, with $n_a \in [0; N_a - 1]$, $n_b \in [0; N_b - 1]$, and $n_c \in [0; N_c - 1]$. Points outside this range are periodic copies of points within the range. The real-space vector corresponding to a grid point **n** may be written in the different representations:

$$\boldsymbol{\tau}_{\mathbf{n}} = \underline{\mathbf{N}}^{-1}\mathbf{n} \quad \text{and} \quad \mathbf{r}_{\mathbf{n}} = \underline{\mathbf{H}}\mathbf{n}. \quad (62)$$

Similarly, each point of the reciprocal-space grid G is associated with an index $\mathbf{l} = (l_a, l_b, l_c)$, with $l_a \in [-N_a/2 + 1; N_a/2]$, $n_b \in [-N_b/2 + 1; N_b/2]$, and $n_c \in [-N_c/2 + 1; N_c/2]$. Points outside this range are periodic copies of points within the range. The reciprocal-space vector corresponding to a reciprocal-space grid point **n** may be written in the different representations as

$$\boldsymbol{\chi}_{\mathbf{l}} = 2\pi\mathbf{l} \quad \text{and} \quad \mathbf{k}_{\mathbf{l}} = 2\pi\underline{\mathbf{L}}^{-1}\mathbf{l}. \quad (63)$$

All gridded functions, that is, real- or reciprocal-space functions that only take a value at a grid point of G, will be indicated with a "g" subscript.

The forward 3D-FFT operation converts a gridded function $f_g(\mathbf{r}_{\mathbf{n}})$ on the grid G into its finite Fourier coefficients $\hat{f}_g(\mathbf{k}_{\mathbf{l}})$ on the same grid, as

$$\hat{f}_g(\mathbf{k}_{\mathbf{l}}) = V_G \sum_{\mathbf{n}\in G} f_g(\mathbf{r}_{\mathbf{n}}) e^{-i\mathbf{k}_{\mathbf{l}}\cdot\mathbf{r}_{\mathbf{n}}}. \quad (64)$$

The backward 3D-FFT performs the reverse operation, namely,

$$f_g(\mathbf{r}_{\mathbf{n}}) = V^{-1} \sum_{\mathbf{l}\in G} \hat{f}_g(\mathbf{k}_{\mathbf{l}}) e^{i\mathbf{k}_{\mathbf{l}}\cdot\mathbf{r}_{\mathbf{n}}}. \quad (65)$$

The P^{3}M algorithm starts by distributing the N_q atomic partial charges q_i at locations $\mathbf{r}_i$ within the computational box onto the neighboring grid points (taking periodicity into account), so as to generate the charge-density grid s_g. This assignment is performed as

$$s_g(\mathbf{r_n}) = \sum_{i=1}^{N_q} q_i \sigma_g(\mathbf{r_n}; \mathbf{r}_i) \tag{66}$$

with

$$\sigma_g(\mathbf{r_n}; \mathbf{r}) = P(\mathbf{r_n} - \mathbf{r}), \tag{67}$$

where P is a so-called assignment function (discussed in more details in the next subsection). The charge density grid is then converted to its (complex) reciprocal-space representation $\hat{s}_g(\mathbf{k_l})$ by applying a forward 3D-FFT to $s_g(\mathbf{r_n})$.

The reciprocal-space potential, that is, the potential generated by the corresponding gridded $(\gamma - b)$-charges is then computed in reciprocal space as

$$\hat{\Phi}_{\gamma,g}(\mathbf{k_l}) = \varepsilon_o^{-1} \hat{G}_g^{\dagger}(\mathbf{k_l}) \hat{s}_g(\mathbf{k_l}), \tag{68}$$

where $\hat{G}_g^{\dagger}$ represents the Fourier coefficients of the influence function. If all charges were located exactly at grid points or if the grid spacing was infinitesimal, the quantity $\hat{G}_g^{\dagger}$ would be given by $k_{\mathbf{l}}^{-2}\hat{\gamma}(ak_{\mathbf{l}})$. However, in practice, a significant gain in accuracy is reached by optimizing $\hat{G}_g^{\dagger}$ to compensate for errors linked with the discretization procedure, taking into account possible variations in the shape and size of the computational box. To this purpose, the influence function $\hat{G}_g^{\dagger}$ is defined as[13]

$$\hat{G}_g^{\dagger}(\mathbf{k_l}) = \hat{G}_g^{o}(\mathbf{k_l}) - \mathrm{Tr}[\underline{\hat{\mathbf{I}}}_g^{o}(\mathbf{k_l})({}^t\underline{\mathbf{L}}^o)^{-1}({}^t\underline{\mathbf{L}} - {}^t\bar{\underline{\mathbf{L}}}^o)], \tag{69}$$

where $\hat{G}_g^o$ is the influence function optimized for a given set $\underline{\mathbf{L}}^o$ of box parameters and $\underline{\hat{\mathbf{I}}}_g^o$ contains the corresponding first-order derivative information in the form

$$\hat{\mathbf{I}}_g^o(\mathbf{k_l}) = -\frac{\partial \hat{G}_g^o(\mathbf{k_l})}{\partial \underline{\mathbf{L}}^o} {}^t\underline{\mathbf{L}}^o. \tag{70}$$

The calculation of the quantities $\hat{G}_g^o$ and $\hat{\mathbf{I}}_g^o$ is described in more details in the next subsection. The optimal influence function $\hat{G}_g^o$ is only optimal for a specific set $\underline{\mathbf{L}}^o$ of box parameters $\underline{\mathbf{L}}$. When the shape and size of the computational box may vary, the second term in eq. (69) includes a first-order correction to the influence function optimized at $\underline{\mathbf{L}}^o$, based on the derivative information calculated simultaneously.

The reciprocal-space contribution $\mathcal{U}_\gamma$ to the total electrostatic energy is given by

$$\mathcal{U}_\gamma = (2\varepsilon_o V)^{-1} \sum_{i=1}^{N_q} \sum_{j=1}^{N_q} \sum_{\mathbf{l}\in G, l\neq 0} q_i \hat{\sigma}_g(\mathbf{k_l}; \mathbf{r}_i) q_j \hat{\sigma}_g^*(\mathbf{k_l}; \mathbf{r}_j) \hat{G}_g^{\dagger}(\mathbf{k_l}). \tag{71}$$

For computational efficiency, this pairwise sum is evaluated as a single sum, through

$$\mathcal{U}_\gamma = (2\varepsilon_o V)^{-1} \sum_{\mathbf{l}\in G, l\neq 0} \hat{G}_g^{\dagger}(\mathbf{k_l}) |\hat{s}_g(\mathbf{k_l})|^2. \tag{72}$$

The (approximate) forces associated with the energy contribution $\mathcal{U}_\gamma$ are obtained through the evaluation of the gridded field $\mathbf{E}_g$ and its interpolation at the location of the charges. The reciprocal-space force on atom i is then written as

$$\mathbf{F}_{\gamma,i} = q_i \mathbf{E}(\mathbf{r}_i) \tag{73}$$

with

$$\mathbf{E}(\mathbf{r}) = V_G \sum_{\mathbf{n}\in G} P(\mathbf{r} - \mathbf{r_n}) \mathbf{E}_g(\mathbf{r_n}). \tag{74}$$

The same assignment function P should be used here and for the charge assignment, to ensure conservation of the total linear momentum during the dynamics.[84] The gridded field $\mathbf{E}_g$ to be used in eq. (74) can be obtained in either of two ways.

The first method (finite-difference) relies on performing a backward 3D-FFT of the potential to obtain the (real) real-space potential, and using a finite-difference approximation to compute the gridded field as

$$\mathbf{E}_g(\mathbf{r_n}) = -\sum_{\mathbf{n}'\in G} iV_G \mathbf{D}_g(\mathbf{r_n} - \mathbf{r_{n'}}) \Phi_{\gamma,g}(\mathbf{r_{n'}}), \tag{75}$$

where $iV_G\mathbf{D}_g$ is a so-called finite-difference operator (discussed in more details in the next subsection).

The second method (ik-differentiation) relies on computing the exact gridded field in reciprocal-space as[88,89]

$$\hat{\mathbf{E}}_g(\mathbf{k_l}) = -i\mathbf{k_l}\hat{\Phi}_{\gamma,g}(\mathbf{k_l}). \tag{76}$$

One then performs three backward 3D-FFTs (one for each Cartesian component) to obtain the corresponding (real) quantity $\mathbf{E}_g$ in real-space.

Quantities involved in the P^{3}M algorithm. The assignment function P of order p [eqs. (67) and (74)] performs the distribution (interpolation) of a continuous function at an arbitrary location onto (from) values at the neighboring p^3 grid points. This function is defined as[56]

$$P(\mathbf{r}) = V_G^{-1} \sum_{\mathbf{n}\in\mathbb{Z}^3} \tilde{P}(\mathbf{r} + \underline{\mathbf{L}}\mathbf{n}) \tag{77}$$

with

$$\tilde{P}(\mathbf{r}) = w_p([\underline{\mathbf{H}}^{-1}\mathbf{r}]_a) w_p([\underline{\mathbf{H}}^{-1}\mathbf{r}]_b) w_p([\underline{\mathbf{H}}^{-1}\mathbf{r}]_c). \tag{78}$$

Here, $w_p(\xi)$ is a normalized one-dimensional function vanishing for $|\xi| \geq p/2$. These functions are listed in Table 7. The assign-

Table 7. One-Dimensional Functions $w_p(\xi)$ Used to Define the Assignment Functions [eqs. (77) and (78)].

p	$w_p(\xi)$	Range
1	1	$\lvert\xi\rvert < 1/2$
2	$1 - \lvert\xi\rvert$	$\lvert\xi\rvert < 1$
3	$-(3/4)(4\xi^2 - 1)$	$\lvert\xi\rvert < 1/2$
	$(1/8)(2\lvert\xi\rvert - 3)^2$	$1/2 < \lvert\xi\rvert < 3/2$
4	$(1/6)(3\lvert\xi\rvert^3 - 6\xi^2 + 4)$	$\lvert\xi\rvert < 1$
	$-(1/6)(\lvert\xi\rvert - 2)^3$	$1 < \lvert\xi\rvert < 2$
5	$(1/192)(48\xi^4 - 120\xi^2 + 115)$	$\lvert\xi\rvert < 1/2$
	$-(1/96)(16\xi^4 - 80\lvert\xi\rvert^3 + 120\xi^2 - 20\lvert\xi\rvert - 55)$	$1/2 < \lvert\xi\rvert < 3/2$
	$(1/384)(2\lvert\xi\rvert - 5)^4$	$3/2 < \lvert\xi\rvert < 5/2$

The functions $w_p(\xi)$ vanish for $\lvert\xi\rvert \geq p/2$.

ment scheme is formulated in terms of oblique coordinates. Thus, in the general case, the distribution (interpolation) of the function from (at) an arbitrary location onto (from) the neighboring grid points is not necessarily correlated with the Cartesian distance between the points (this is only the case for a rectangular computational box).

The Fourier coefficients $\hat{P}$ of the assignment function P are given by

$$\hat{P}(\mathbf{k}) = \hat{w}_p([{}^t\underline{\mathbf{H}}\mathbf{k}]_a)\hat{w}_p([{}^t\underline{\mathbf{H}}\mathbf{k}]_b)\hat{w}_p([{}^t\underline{\mathbf{H}}\mathbf{k}]_c), \tag{79}$$

where $\hat{w}_p(\kappa)$ is the continuous Fourier transform of $w_p(\xi)$, which evaluates to

$$\hat{w}_p(\kappa) = [2\kappa^{-1}\sin(\kappa/2)]^p(1 - \delta_\kappa) + \delta_\kappa. \tag{80}$$

The finite-difference operator $iV_G\mathbf{D}_g$ of order q estimates the gradient of a given function at a grid point based on the values of the function at the $6q$ neighboring grid points along the three box axes. The operator $\mathbf{D}_g$ is defined as[56]

$$\mathbf{D}_g(\mathbf{r_n}) = \sum_{j=1}^{q} c_j\mathbf{d}_{g,j}(\mathbf{r_n}). \tag{81}$$

In this expression, the centered-difference operator $iV_G\mathbf{d}_{g,j}$ generates the gradient contribution evaluated from the function at the six neighboring points distant from the grid point by $\pm jN_a^{-1}a$, $\pm jN_b^{-1}b$, and $\pm jN_c^{-1}c$ along the three box axes. The operator $\mathbf{d}_{g,j}$ is given by

$$\mathbf{d}_{g,j}(\mathbf{r_n}) = \underline{\mathbf{R}}\underline{\mathbf{S}}\mathbf{e}_{g,j}(\mathbf{r_n}), \tag{82}$$

where $\underline{\mathbf{R}}$ and $\underline{\mathbf{S}}$ are the matrices defined in eqs. (4) and (5), $\mathbf{e}_{g,j}$ is the corresponding operator in terms of oblique fractional coordinates, that is,

$$e_{g,j,a}(\mathbf{r_n}) = iV_G^{-1}(2jN_a^{-1}a)^{-1}\delta_{n_b}\delta_{n_c}\sum_{m\in\mathbb{Z}}(\delta_{n_a-j+N_am} - \delta_{n_a+j+N_am}), \tag{83}$$

with similar expressions for the b and c components.

The Fourier coefficients $\hat{\mathbf{D}}_g$ of the operator $\mathbf{D}_g$ are given by

$$\hat{\mathbf{D}}_g(\mathbf{k_l}) = \sum_{j=1}^{q} c_j\hat{\mathbf{d}}_{g,j}(\mathbf{k_l}), \tag{84}$$

where

$$\hat{\mathbf{d}}_{g,j}(\mathbf{k_l}) = \underline{\mathbf{R}}^t\underline{\mathbf{S}}^{-1}\hat{\mathbf{e}}_{g,j}(\mathbf{k_l}) \tag{85}$$

and

$$\hat{e}_{g,j,a}(\mathbf{k_l}) = (jN_a^{-1}a)^{-1}\sin(j[{}^t\underline{\mathbf{H}}\mathbf{k_l}]_a), \tag{86}$$

with similar expressions for the b and c components. Taken together, eqs. (84), (85), and (86) may be written

$$\hat{\mathbf{D}}_g(\mathbf{k_l}) = {}^t\underline{\mathbf{H}}^{-1}\sum_{j=1}^{q} c_j j^{-1}\begin{pmatrix}\sin(j[{}^t\underline{\mathbf{H}}\mathbf{k_l}]_a)\\ \sin(j[{}^t\underline{\mathbf{H}}\mathbf{k_l}]_b)\\ \sin(j[{}^t\underline{\mathbf{H}}\mathbf{k_l}]_c)\end{pmatrix}. \tag{87}$$

Note that the exact difference operator (as used in ik-differentiation) corresponds to

$$\hat{\mathbf{D}}_g(\mathbf{k_l}) = \mathbf{k_l}. \tag{88}$$

For a given finite-difference operator of order q, the coefficients c_j in eq. (81) may be selected so as to minimize the difference between the corresponding finite-difference operator and the exact-difference operator. In reciprocal space, this leads to the requirement

$${}^t\underline{\mathbf{H}}^{-1}\sum_{j=1}^{q} c_j j^{-1}\begin{pmatrix}\sin(2\pi jN_a^{-1}l_a)\\ \sin(2\pi jN_b^{-1}l_b)\\ \sin(2\pi jN_c^{-1}l_c)\end{pmatrix} \approx \mathbf{k_l}. \tag{89}$$

Solving the resulting system of equations results in the optimal coefficients listed in Table 8.

Table 8. Optimal Weighting Coefficients c_j ($j \leq q$) for the Finite-Difference Operator.

q	c_1	c_2	c_3	c_4	c_5
1	1				
2	4/3	−1/3			
3	3/2	−3/5	1/10		
4	8/5	−4/5	8/35	−1/35	
5	5/3	−20/21	5/14	−5/63	1/126

For a given set of box parameters characterized by the matrix $\underline{\mathbf{L}}^o$, the influence function that optimally compensate for discretization errors can be computed as[84]

$$\hat{G}_g^o(\mathbf{k_l}) = \frac{\hat{\mathbf{D}}_g(\mathbf{k_l}) \cdot [\sum_{\mathbf{m}\in\mathbb{Z}^3} \mathbf{k}_{\mathbf{l,m}} k_{\mathbf{l,m}}^{-2} \hat{\gamma}(ak_{\mathbf{l,m}}) \hat{P}^2(\mathbf{k}_{\mathbf{l,m}})]}{\hat{\mathbf{D}}_g^2(\mathbf{k_l})[\sum_{\mathbf{m}\in\mathbb{Z}^3} \hat{P}^2(\mathbf{k}_{\mathbf{l,m}})]^2}, \qquad (90)$$

where

$$\mathbf{k}_{\mathbf{l,m}} = \mathbf{k}_{\mathbf{l}+\underline{\mathbf{N}}\mathbf{m}} = \mathbf{k_l} + 2\pi' \underline{\mathbf{H}}^{-1}\mathbf{m} \qquad (91)$$

is an alias vector of $\mathbf{k}$ (with $\mathbf{m} \in \mathbb{Z}^3$). Equation (90) is valid for $\mathbf{l} \in G$ and $\mathbf{l} \neq \mathbf{0}$, together with $\hat{G}_g(\mathbf{0}) = 0$. In practice, the summation over alias vectors is restricted to $\mathbf{m}$ vectors with integer components in the range $[-m_{\max} \ldots m_{\max}]$. A value of 2 or 3 for $m_{\max}$ is usually sufficient to reach convergence. The quantity $\hat{P}$ is given in by eq. (79), the quantity $\hat{\mathbf{D}}_g$ by eqs. (87) (finite-difference) or (88) (ik-differentiation), and the quantity $\hat{\gamma}$ by eq. (43) (see Table 4). The negative derivative of the optimal influence function with respect to the box parameters $\underline{\mathbf{L}}^o$, amplified on the right by ${}^t\underline{\mathbf{L}}^o$ may be calculated simultaneously with the influence function as

$$\begin{aligned}\hat{\underline{\mathbf{I}}}_g^o(\mathbf{k_l}) = \frac{1}{\hat{\mathbf{D}}_g^2(\mathbf{k_l})[\sum_{\mathbf{m}\in\mathbb{Z}^3} \hat{P}^2(\mathbf{k}_{\mathbf{l,m}})]^2} \sum_{\mathbf{m}\in\mathbb{Z}^2} \frac{\hat{P}^2(\mathbf{k}_{\mathbf{l,m}})}{k_{\mathbf{l,m}}^2} \Bigg\{ \Bigg\{ k_{\mathbf{l,m}} \otimes \hat{\mathbf{D}}_g(\mathbf{k_l}) \\ + \hat{\mathbf{D}}_g(\mathbf{k_l}) \otimes k_{\mathbf{l,m}} - \frac{2[\mathbf{k}_{\mathbf{l,m}} \cdot \hat{\mathbf{D}}_g(\mathbf{k_l})]\mathbf{k}_{\mathbf{l,m}} \otimes \mathbf{k}_{\mathbf{l,m}}}{k_{\mathbf{l,m}}^2} \\ - \frac{2[\mathbf{k}_{\mathbf{l,m}} \cdot \hat{\mathbf{D}}_g(\mathbf{k_l})]\hat{\mathbf{D}}_g(\mathbf{k_l}) \otimes \hat{\mathbf{D}}_g(\mathbf{k_l})}{\hat{\mathbf{D}}_g^2(\mathbf{k_l})} \Bigg\} \hat{\gamma}(ak_{\mathbf{m}}) \\ + \frac{[\mathbf{k}_{\mathbf{l,m}} \cdot \hat{\mathbf{D}}_g(\mathbf{k_l})]k_{\mathbf{l,m}} \otimes k_{\mathbf{l,m}}}{k_{\mathbf{l,m}}^2} ak_{\mathbf{l,m}} \hat{\gamma}'(ak_{\mathbf{l,m}}) \Bigg\} \qquad (92)\end{aligned}$$

with $\hat{\gamma}'(\kappa) = d\hat{\gamma}(\kappa)/d\kappa$ valid for $\mathbf{l} \in G$ and $l \neq \mathbf{0}$, together with $\hat{\underline{\mathbf{I}}}_g^o = \underline{\mathbf{0}}$.

Replica-Exchange Simulation

To obtain canonical distributions for complex molecular systems, efficient sampling of the configurational space is necessary. Finding the global minimum on the typically rough potential energy landscape of a peptide or protein is likewise difficult. In recent years, the replica-exchange method[90–95] (also known as parallel tempering[95]) has received much attention. A number of noninteracting replicas are simulated simultaneously at different conditions (e.g., different temperatures). After a given simulation time, an exchange between two replicas is attempted, followed by another (individual) simulation period. The method has been applied to biomolecular systems,[96–99] using Monte Carlo (MC) techniques and molecular dynamics (REMD) to propagate the individual replicas. The probability of each state $x = (\mathbf{r}, \mathbf{p})$ in the canonical ensemble at temperature T is proportional to the weight factor

$$W(x) = \exp(-\beta H(\mathbf{r}, \mathbf{p})), \qquad (93)$$

where H is the Hamiltonian and $\beta = 1/k_B T$, k_B being Boltzmann's constant. The weight factor for the global state X determined by the states of the M replicas is the product of the single weights, that is,

$$W_{\mathrm{REM}}(X) = \exp\left(-\sum_{i=1}^{M} \beta_i H(\mathbf{r}_i, \mathbf{p}_i) \right). \qquad (94)$$

After a fixed number of MD integration steps, an MC exchange between two replicas is attempted (changing from state X to state X'). To sample canonical ensembles at each temperature, the detailed balance condition on the transition probability $w(X \to X')$

$$W_{\mathrm{REM}}(X) w(X \to X') = W_{\mathrm{REM}}(X') w(X' \to X) \qquad (95)$$

has to be fulfilled. This can be satisfied, for instance, by the usual Metropolis criterion

$$p(X \to X') = \frac{w(X \to X')}{w(X' \to X)} = \begin{cases} 1 & \text{for } \Delta \leq 0, \\ \exp(-\Delta) & \text{for } \Delta > 0, \end{cases} \qquad (96)$$

with

$$\Delta = (\beta_i - \beta_j)(U(\mathbf{r}_j) - U(\mathbf{r}_i)), \qquad (97)$$

where $U(\mathbf{r})$ is the potential energy associated with the configuration $\mathbf{r}$. If the exchange was successful, the momenta of the exchanged replicas are scaled to correspond to their new temperatures.

An extension of the replica-exchange method to sample the isothermal–isobaric ensemble has been suggested.[100] In this case, an additional term incorporating the pressure and volume change appears in the exchange probability

$$\Delta = (\beta_i - \beta_j)(U(\mathbf{r}_j) - U(\mathbf{r}_i)) + (\beta_i P_i - \beta_j P_j)(V_j - V_i). \qquad (98)$$

Unfortunately, the application of the REMD method to explicit solvent simulations, although successful for small systems, is rather difficult.[101–107] Because the exchange probability between two states decreases with increasing system size, explicit-solvent simulation requires many more states (replicas) separated by small temperature differences. This problem may be alleviated by not exchanging a thermodynamic property like the temperature between the individual replica, simulations but rather altering spe-

cific interactions.[108–110] Then, the replicas are distinguished by their Hamiltonians

$$H_i(\mathbf{r}_i, \mathbf{p}_i) = K(\mathbf{p}_i) + U_i(\mathbf{r}_i) \tag{99}$$

using for each simulation a different potential energy function U_i. In MD++ the different Hamiltonians H_i are defined using a coupling parameter λ and a perturbation topology. The replica with $\lambda_i = 0$ corresponds to state A (normal topology) in a perturbation simulation, the replica at $\lambda_i = 1$ to state B (fully perturbed state; with, e.g., scaled-down nonbonded or bonded interaction terms). The other replicas are distributed in between these two ($0 < \lambda_i < 1$).

Inserting the individual H_i into eq. (94), leads to

$$W_{\mathrm{REM}}(X) = \exp\left(-\sum_{i=1}^{M} \beta_i H_i(\mathbf{r}_i, \mathbf{p}_i)\right), \tag{100}$$

and, using the detailed balance criterion to

$$p(X \rightarrow X') = \frac{w(X \rightarrow X')}{w(X' \rightarrow X)} = \begin{cases} 1 & \text{for } \Delta \leq 0, \\ \exp(-\Delta) & \text{for } \Delta > 0, \end{cases} \tag{101}$$

with

$$\Delta = \beta_i(U_i(\mathbf{r}_j) - U_i(\mathbf{r}_i)) - \beta_j(U_j(\mathbf{r}_j) - U_j(\mathbf{r}_i)), \tag{102}$$

where the potential energy of the two configurations $\mathbf{r}_i$ and $\mathbf{r}_j$ needs to be evaluated twice, with Hamiltonians H_i and H_j.

The replica-exchange method was implemented in MD++ based on sockets and tcp as communication protocol. A server distributes the short MD runs corresponding to the different replicas to a (dynamical) number of clients. After a given number of simulation steps has been carried out, the client reports back to the server the final (potential) energies (evaluated using the Hamiltonian H_i and the Hamiltonian H_j (if different), and the final configuration. The server then calculates the switching probability $p(i \rightarrow j)$, draws a random number and, if the switch is successful, exchanges the states. As soon as a client is free, the next replica gets assigned to it.

Replicas can differ in the temperature and in the coupling parameter λ. A replica-exchange state consists of replicas for all possible λ values at each temperature T_i ($M = N_\lambda \cdot N_T$ replicas). The MC exchange attempt is alternated between exchanges of (neighboring) λ's and temperatures T.

Coarse-Grained Simulation

Most molecular simulations are making use of atom-level (AL) models. This limits the time scale of such simulations for solvated macromolecules to the nanosecond range. Longer time scales can be reached by treating molecules or molecular fragments as single particles or beads, whose motion is simulated using a simple force field describing interbead interactions. When the energy function of such a coarse-grained (CG) model is chosen to be smooth and short-ranged, the efficiency of CG simulations can be orders of magnitude (10^3–10^5) higher than the corresponding AL simulations, be it at the expense of the loss of atomic detail and some accuracy.[111–115]

A recently proposed CG model[116] for liquid simulations has the same functional form as the GROMOS force field,[3,117] except for the use of a switching function[118] for the nonbonded Lennard–Jones and electrostatic interactions at distances just below the cutoff distance. This CG model was implemented into GROMOS05 (MD++ only), however with a slightly different switching function, because the GROMACS one[118] appeared to be discontinuous and led to nonconservation of energy in MD simulation.

In the absence of switching, the nonbonded interaction energy between particles i and j can be written as ($\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$)

$$V(\mathbf{r}_{ij}) = \sum_{\alpha=1,6,12} V_\alpha(\mathbf{r}_{ij}) = \sum_{\alpha=1,6,12} c_\alpha \Phi_\alpha(\mathbf{r}_{ij}) \tag{103}$$

with ($r = |\mathbf{r}|$)

$$\Phi_\alpha(r) = r^{-\alpha} \tag{104}$$

and

$$\begin{aligned} c_1 &= \frac{q_i q_j}{4\pi\varepsilon_0\varepsilon_1} \\ c_6 &= -4\varepsilon_{ij}\sigma_{ij}^6 = -C_6(i,j) \\ c_{12} &= +4\varepsilon_{ij}\sigma_{ij}^{12} = +C_{12}(i,j), \end{aligned} \tag{105}$$

where standard notations for atomic charges (q_i) and van der Waals interaction parameters (C_6 and C_{12}) have been used. In the CG model all nonbonded interactions are smoothly switched to zero over the range [R_{sw}, R_c], where R_{sw} denotes the start of the switching and R_c the cutoff radius. In this model[116] one has R_c = 1.2 nm and R_{sw} = 0 nm for the Coulomb interaction and R_{sw} = 0.9 nm for the van der Waals interactions. The nonbonded interaction energy function including switching reads for the three terms in eq. (103)

$$\Phi_\alpha^s(r) = \begin{cases} \Phi_\alpha(r) & r \leq R_{\mathrm{sw}} \\ \Phi_\alpha(r) + S_\alpha(r) & R_{\mathrm{sw}} \leq r \leq R_c. \\ 0 & r \geq R_c \end{cases} \tag{106}$$

Requiring that the functions $S_\alpha(r)$ switch the energy, the force, and the derivative of the force smoothly (without discontinuities) to zero at $r = R_c$, yields the conditions

$$S_\alpha(R_{\mathrm{sw}}) = S'_\alpha(R_{\mathrm{sw}}) = S''_\alpha(R_{\mathrm{sw}}) = 0 \tag{107}$$

and

$$\Phi_\alpha^s(R_c) = \Phi_\alpha^{s'}(R_c) = \Phi_\alpha^{s''}(R_c) = 0. \tag{108}$$

The conditions of eq. (107) are satisfied by a fourth-degree polynomial

$$S_\alpha(r) = -\frac{1}{3}A(r - R_{sw})^3 - \frac{1}{4}B(r - R_{sw})^4 - C. \quad (109)$$

The conditions of eq. (108) determine the constants

$$A = \frac{\alpha[(\alpha + 1)R_{sw} - (\alpha + 4)R_c]}{R_c^{\alpha+2}(R_c - R_{sw})^2}, \quad (110)$$

$$B = -\frac{\alpha[(\alpha + 1)R_{sw} - (\alpha + 3)R_c]}{R_c^{\alpha+2}(R_c - R_{sw})^3}, \quad (111)$$

$$C = \frac{1}{R_c^\alpha} - \frac{1}{3}A(R_c - R_{sw})^3 - \frac{1}{4}B(R_c - R_{sw})^4. \quad (112)$$

The expression for the shifted or switched force on particle i by particle j for the three nonbonded interaction terms $V^s_\alpha(r_{ij})$ is then

$$\mathbf{f}^s_{\alpha i}(\mathbf{r}_{ij}) = -\frac{\partial V^s_\alpha(\mathbf{r}_{ij})}{\partial r_{ij}}\frac{\partial r_{ij}}{\partial \mathbf{r}_i} = -c_\alpha \Phi^{s'}_\alpha(r_{ij})\frac{\mathbf{r}_{ij}}{r_{ij}}, \quad (113)$$

with

$$\Phi^{s'}_\alpha(r) = \begin{cases} \Phi'_\alpha(r) & r \le R_{sw} \\ \Phi'_\alpha(r) + S'_\alpha(r) & R_{sw} \le r \le R_c, \\ 0 & r \ge R_c \end{cases} \quad (114)$$

$$\Phi'_\alpha(r) = \frac{-\alpha}{r^{\alpha+1}}, \quad (115)$$

and

$$S'_\alpha(r) = -A(r - R_{sw})^2 - B(r - R_{sw})^3. \quad (116)$$

We note that the switched potential energy determined by eq. (106) and forces from eq. (113) are only correct for a distance dependence of the potential energy function of the form of eq. (104). So, it cannot be used in the presented form if the soft-core interaction or reaction field forces as defined in GROMOS are to be used.

Free-Energy through One-Step Perturbation

The calculation of relative binding free energies of many ligands to a common receptor is of relevance for drug design and screening purposes, and for obtaining a better understanding of interactions governing molecular complexation in general. The one-step perturbation technique[119] allows for the calculation of a great many relative free energies from a single simulation of a (not necessarily physically meaningful) reference state.[120–127] The idea behind the method is to simulate a judiciously chosen reference compound R generating an ensemble of structures that contains conformations representative for many physically relevant compounds. The free energy difference between any real ligand A and the reference compound R can be obtained from the perturbation formula

$$\Delta G_{AR} = \Delta G_A - \Delta G_R = -k_B T \ln\langle e^{-(H_A - H_R)/k_B T}\rangle_R, \quad (117)$$

where the angular brackets indicate the ensemble average of the configurations generated in a simulation of R. H_A and H_R are the Hamiltonians for the real compound (A) and the reference compound (R), respectively. Because this expression involves the difference between two Hamiltonians, only interactions that differ between compounds A and R need to be reevaluated over the ensemble. This allows for the calculation of thousands[125] to millions[126] of relative free energies from a handful of simulations of reference states R.

The success of the method critically depends on the choice of the reference state R; it should allow wide sampling, but not so wide that insufficient statistics is obtained. One of the key elements that allow wide sampling is the use of soft-core nonbonded interactions, which allows for a spatial overlap between these atoms. In GROMOS96, the soft-core nonbonded interaction was chosen to be of the form[3,4,119]

$$V^{sc}(r_{ij}) = \frac{4\varepsilon_{ij}\sigma_{ij}^6}{s_{LJ}(i,j)\lambda^2\sigma_{ij}^6 + r_{ij}^6}\left[\frac{\sigma_{ij}^6}{s_{LJ}(i,j)\lambda^2\sigma_{ij}^6 + r_{ij}^6} - 1\right] + \frac{q_i q_j}{4\pi\varepsilon_0\varepsilon_1}\left[\frac{1}{(s_C(i,j)\lambda^2 + r_{ij}^2)^{1/2}} - \frac{\frac{1}{2}C_{rf}r_{ij}^2}{(s_C(i,j)\lambda^2 + R_{rf}^2)^{3/2}} - \frac{1 - \frac{1}{2}C_{rf}}{R_{rf}}\right]. \quad (118)$$

In GROMOS96, the soft-core parameters s_{LJ} and s_C were taken equal for all soft-core atom pairs. In GROMOS05 (MD++ only), $s_{LJ}(i, j)$ and $s_C(i, j)$ are calculated by combining the distinct softness parameter specified per atom, allowing fine tuning of the reference state, that is,

$$s(i,j) = \begin{cases} \frac{1}{2}(s(i) + s(j)) & s(i) \ne 0,\ s(j) \ne 0 \\ s(i) & s(i) \ne 0,\ s(j) = 0 \\ s(j) & s(i) = 0,\ s(j) \ne 0 \\ 0 & s(i) = 0,\ s(j) = 0. \end{cases} \quad (119)$$

The GROMOS++ postprocessing programs `pt_top` (to generate a real topology from a topology and a perturbation topology) and `ener` (to recalculate the interaction energy of specified atoms) or `m_pt_top` and `m_ener` (to do the same for multiple physical compounds at the same time), and `dg_ener` (to calculate the relative free energies) may be used to analyze the reference state ensemble.

Features under Construction

A number of algorithmic and force field developments are currently investigated and only provisionally implemented in GROMOS05. These are not yet part of the standard software.

Polarizability

During the past decade, the accuracy of the GROMOS force field could still be improved by reparametrizing it to reproduce the thermodynamic properties of small molecules.[117,128,129] The latest parameter set, 53A6, reproduces the free energies of apolar (cy-

clohexane) and of polar (water) solvation for typical biomolecules to within 1–2 kJ mol^{-1} (about $k_BT/2$).[117] However, this study indicated that the lack of explicit electronic polarizability is limiting a further increase in the accuracy of nonpolarizable force fields. Various possibilities to explicitly account for polarizability in MD simulations have been reviewed.[130] In our view, the most promising approach in terms of striking a balance between accuracy on the one hand and simplicity and computational efficiency on the other is the so-called charge-on-spring (COS) type of models.[131] A COS model for polarizable water that is, in principle, compatible with the GROMOS type of force fields has been developed[132,133] and a GROMOS software version with COS polarization is in its test phase, based on the implementation previously sketched.[130]

Enhanced One-Step Perturbation

The efficiency of the one-step perturbation technique can be enhanced along two lines. First, because the choice of the nonphysical reference state R is completely free, even nonatomic reference compounds can be used.[127] Using the concept of soft-core atoms, one may use atoms of dual character in the reference compound: atom i may interact through a soft-core interaction with atom j, but through a normal interaction with atom k. In this manner the part of the system that is softened to enhance the sampling can be restricted such that no unnecessary softness is introduced that aggravates the sampling. Second, when evaluating the difference $H_A - H_R$ in eq. (117), the Hamiltonian $H_A(\mathbf{r}_R)$ or energy of the real compound A is to be calculated using the ensemble $\mathbf{r}_R$ of configurations of the reference compound R in the trajectory of R. This requires a definition of the interaction sites of H_A in terms of atom (mass) positions of H_R, which should be the same for all configurations of the trajectory of R. However, because eq. (117) is independent of translation and rotation of the coordinate system that is used to define H_A, additional sampling in the form of translation and rotation of the real compound with respect to the trajectory configurations of the reference compound R can be carried out,[122] for example, through limited translation and rotation. Furthermore, if the reference compound is composed of atomic soft sites on each of which more than one real ligand atom can be superimposed, the calculation of $H_A - H_R$ for the many combinations of real ligand atoms at the different soft sites can be decomposed in single soft-site calculations, of which the contributions to $H_A - H_R$ can be added. This allows a considerable increase in efficiency.

Code Organization, Implementation

MD Engine in FORTRAN: PROMD

The FORTRAN MD engine (PROMD) is an enhancement of the GROMOS96 MD engine. It is written in FORTRAN77 except for the use of included files and macro preprocessing. Macros are in particular used to get rid of unnecessary features (such as four-dimensional simulation, unused periodicity code, or unused perturbation code) so as to improve the performance for specific applications through the use of a specialized code. To facilitate performance tuning, timing routines have been included, and the time spent within various components of the program is reported at the end of each simulation. Major additional algorithmic features (with respect to GROMOS96) have been described (see earlier).

MD Engine in C++: MD++

The C++ MD engine (MD++) has been written from scratch. The major motivation was to further increase the modularity and therefore the extendability of the MD program. The code is split into two parts, the first one being an MD library containing basic functions necessary to run an MD simulation, the second one being the actual MD program. This second part is very small. It is therefore easy to write other specialized MD programs that make use of a subset of the functions provided in the library or apply them in a different order. The source code of the library is, in turn, split up into nine different parts: *math, simulation, topology, configuration, algorithm, interaction, io, util,* and *check* (represented as C++ *namespaces*).

- *math* contains classes for vectors, matrices, and vector arrays, mathematical operations, physical constants, and periodic boundary treatment.
- *simulation* contains the simulation parameters supplied to run an MD or SD simulation or an EM.
- *topology* contains the topology of the simulated system, possibly also including a perturbation topology.
- *configuration* contains the state of a system: its coordinates, velocities, forces, restraints data, and so on.
- *algorithm* contains classes that use information from *simulation* and *topology* to act upon a *configuration.* All steps during an MD or SD simulation or EM can be carried out using an *algorithm.*
- *interaction* contains the largest *algorithm:* the energy, forces, and virial evaluation. Here, all interaction terms and their parameters are defined. Because of its size, *interaction* is a separate part, although it formally belongs to *algorithm.* The *interaction* part is further split into *bonded, nonbonded,* and *special* interactions.
- *io* contains classes to read in or write out information. All file access is block oriented and the files are human readable.
- *util* contains a few extra classes that are necessary to set up a simulation but which do not exactly belong to it. Parsing of command line arguments, generation of initial velocities, or setting of debug levels are examples of classes found herein.
- *check* contains test routines. Testing includes the automatic calculation of energies under different conditions as well as the calculation of forces, virial tensor, and energy λ-derivatives and their comparison to values obtained by finite difference calculations.

One step of an MD or SD simulation or EM consists of several `Algorithms` (Fig. 3) applied to the `Configuration` in the right order. The `Algorithm_Sequence` class (Fig. 4) is a container for all these algorithms. When a simulation is set up, they are inserted in the correct order into the `Algorithm_Sequence`. Before the start of a simulation, all algorithms will be initialized (by calling the `init ( )` function). During an MD

step (`Algorithm_Sequence::run( )`), the algorithms are applied (by calling `Algorithm::apply( )`). The `force field` itself is also an `algorithm`, which, when applied, calculates the energies, forces, and virial contribution of all force field terms for the complete system. The force field terms themselves are `Interaction` classes. The `Forcefield` is therefore a container to store the different `Interaction` objects (in analogy to the `Algorithm_Sequence` and `Algorithm` classes). When the force field is applied, it calls `calculate_interactions( )` on all interaction objects. There are distinct interaction objects for the covalent interactions (bond length, bond angle, improper–dihedral, and torsional–dihedral interactions), the nonbonded interactions (pairlist construction, long-range interactions, and short-range interactions) and the nonphysical interactions (atom position, atom distance, dihedral angle, NOE, or *J*-value restraints). It is very easy to add a custom `Interaction` class to calculate a nonstandard interaction.

The classes corresponding to the steps in the MD, SD, or EM algorithm are shown in Table 9 and an overview of the (nonbonded) interaction classes is given in Figure 5. The `Nonbonded_Sets` contain independent subsets of the nonbonded interactions. Their `calculate_interactions ( )` method may be called in parallel (using either *shared* or *distributed* memory parallelization). The `Nonbonded_Sets` share (through the `Nonbonded_Interaction`) a pairlist construction algorithm, which they call to create the part of the complete pairlist relevant to them. These different parts of the pairlist stay together with the `Nonbonded_Set` and need never be assembled into the complete pairlist. To gain flexibility, the calculation of the individual atom–atom pair interaction is further split up into a `Nonbonded_Outerloop` (loops over the atom–atom pairs), a `Nonbonded_Innerloop` (prepares the parameters necessary to calculate the interaction), and a `Nonbonded_Term` (calculates the atom–atom pair interaction energy, force, and virial contribution). The `Storage` class provides directly accessible (local) memory for each `Nonbonded_Set`.

```
class Algorithm{
  public:
  Algorithm(string name) : name(name) {}
  ~Algorithm() {}
  virtual int init(Topology & topo,
                   Configuration & conf,
                   Simulation & sim) = 0;

  virtual int apply(Topology & topo,
                    Configuration & conf,
                    Simulation & sim) = 0;

  string name;
};
```

Figure 3. Interface of the `Algorithm` class.

```
class Algorithm_Sequence : public vector<Algorithm *> {
  public:
  Algorithm_Sequence();
  ~Algorithm();

  int init(Topology & topo,
           Configuration & conf,
           Simulation & sim);

  int run(Topology & topo,
          Configuration & conf,
          Simulation & sim);

  Algorithm * algorithm(string name);
};
```

Figure 4. Interface of the `Algorithm_Sequence` class.

Efficiency

The main goal for writing a new C++ MD engine was to further improve on modularity (using some object-oriented features) and extendability (using clear and common interfaces between the modules). Nevertheless, a simulation code has to be reasonably efficient to be of practical use. The complete code is written in standard C++,[134] no language extensions or machine-specific parts are used anywhere, resulting in a highly portable program. This means that the compiler has to do all machine-specific optimizations. We believe that the absence of any machine-specific parts of code, which require duplication to be able to run on different machines, facilitates future modification. Furthermore, current compilers are getting ever better at producing fast programs, making use of the specific features available on the machine.

In the inner loops of the interaction calculation, `templates` are used to generate specialized code. There are, for instance, specialized periodicity classes for the different implemented types of periodic boundary conditions (vacuum, rectangular, truncated octahedral, and triclinic). The `Innerloop` methods are called with the boundary type as a template argument. Thus, the compiler will generate different specialized versions of the inner loops for different boundary conditions automatically. In the same manner, the interaction function term of the nonbonded interaction can also be chosen (e.g., with or without switching function for nonbonded interactions) without any *if* statement required in the compiled inner loop. An example code fragment is shown in Figures 6 and 7. The same technique is used to implement perturbation simulations and different definitions of the virial tensor.

Some algorithms do rely on information from the previous integration step. To help implementing those kind of algorithms, the complete current and old state (positions, velocities, forces, energies, restraint and constraint data, averages, and so on) of the simulation are stored. During the leap-frog algorithm, the current state becomes the old state and the updated information is stored in the new current state. This transfer is done by a simple (and fast) pointer exchange. This duplication slightly increases memory usage (but the required space is still small compared to that used to store the pairlists).

A comparison of the efficiency of the C++ code with respect to the GROMOS96 MD engine (in FORTRAN) is given in Table

Table 9. Classes Corresponding to MD Algorithm Steps.

1.	Write position and velocity components.	Out_Trajectory
2.	Remove center of mass motion.	Remove_COM_Motion
3.	Calculate (unconstrained) forces and energies from the potential energy function (using nearest image convention in case of periodic boundary conditions).	Forcefield Bond_Interaction Angle_Interaction Improper_Dihedral_Interaction
	Save these.	Dihedral_Interaction Nonbonded_Interaction Position_Restraints_Interaction Distance_Restraints_Interaction NOE_Restraints_Interaction JValue_Restraints_Interaction
4.	Satisfy position constraints.	Position_Constraints_Interaction
5.	Update the velocities using the leapfrog scheme.	Leapfrog_Velocities
6.	Apply temperature coupling [weak coupling or Nosé–Hoover(-chains)].	Berendsen_Thermostat Nose_Hoover_Thermostat
7.	Update the positions using the leapfrog scheme.	Leapfrog_Positions
8.	Satisfy distance constraints (using SHAKE, M-SHAKE, or LINCS).	Shake MShake Lincs
9.	Calculate temperature(s).	Temperature_Calculation
10.	Calculate pressure.	Pressure_Calculation
11.	Apply pressure scaling (weak coupling).	Berendsen_Barostat
12.	Update lambda and topology for slow-growth simulations.	Slow_Growth
13.	Calculate total energies, averages, and fluctuations. Save these.	Energy_Calculation

2. This comparison shows that MD++, using the standard pairlist algorithm is approximately a factor 2 slower than GROMOS96 (standard pairlist algorithm); improved algorithms (like a grid-based pairlist construction) may have a large impact on the performance reducing the time spent to two-thirds for the membrane and even by a factor of 3 for the protein system. Note that the optimized pairlist construction algorithm implemented in the FORTRAN MD engine (PROMD) only benefits from more efficient processor cache usage but still scales as $O(N^2)$ with system size. Still, for the systems tested here, it achieves equal overall efficiency as the $O(N)$ scaling grid-based pairlist algorithm in MD++. The future will show whether the improved extendability of MD++ will, through improved algorithms, lead to a faster C++ code than the FORTRAN (PROMD) code, or whether slightly inferior performance is the price to pay for a more structured code layout.

Debugging Information

It is often difficult to figure out what is going on during an MD or SD simulation or an EM, and many users tend to use the program as a *black box.* MD++ tries to improve this situation by enabling the user to select a tunable amount of information to be printed out during the simulation. Every (output or debugging) message is associated with a debugging level, and the message is printed only if the requested debugging level is high enough. Additionally, every code section belongs to a *module* and a *submodule.* Different debug levels can be specified for all combinations of *modules* and *submodules.* In that way, fine-grained control is achieved on how much information from which part of the MD++ code should be printed.

Parallelization

Computationally, the interaction calculation is by far the most expensive part of an MD or SD simulation or an EM, while the nonbonded interactions constitute the bulk of the effort. Again, MD++ is focused on achieving parallelization without complicating the code. The nonbonded interaction is split up into `Nonbonded_Sets`, each containing its own storage space for a pairlist, energies, forces, and virials. In this way, the standard code is ready for shared and distributed memory parallelization without any need for code duplication. If the system is using distributed memory, the (updated) positions have to be copied from the master to all other processes before the next interaction calculation. While composing the pairlist in parallel, only a subset of atoms is considered per process, so that each processor creates its own partial and local pairlist. The interactions are calculated from this partial pairlist and stored in local arrays. This ensures synchronization for shared memory machines and replicated data parallelization for distributed memory systems. After the partial interaction calculations have finished, the energies, forces, and virials of all nonbonded sets are summed up and stored in the `Configuration` of the master process.

MD++ can use `OpenMP` (www.openmp.org) for shared memory and `MPI` (www.mpi-forum.org) for distributed memory parallelization. Reasonable parallelization (using a small number of parallel processes) can be achieved with only a few additional lines

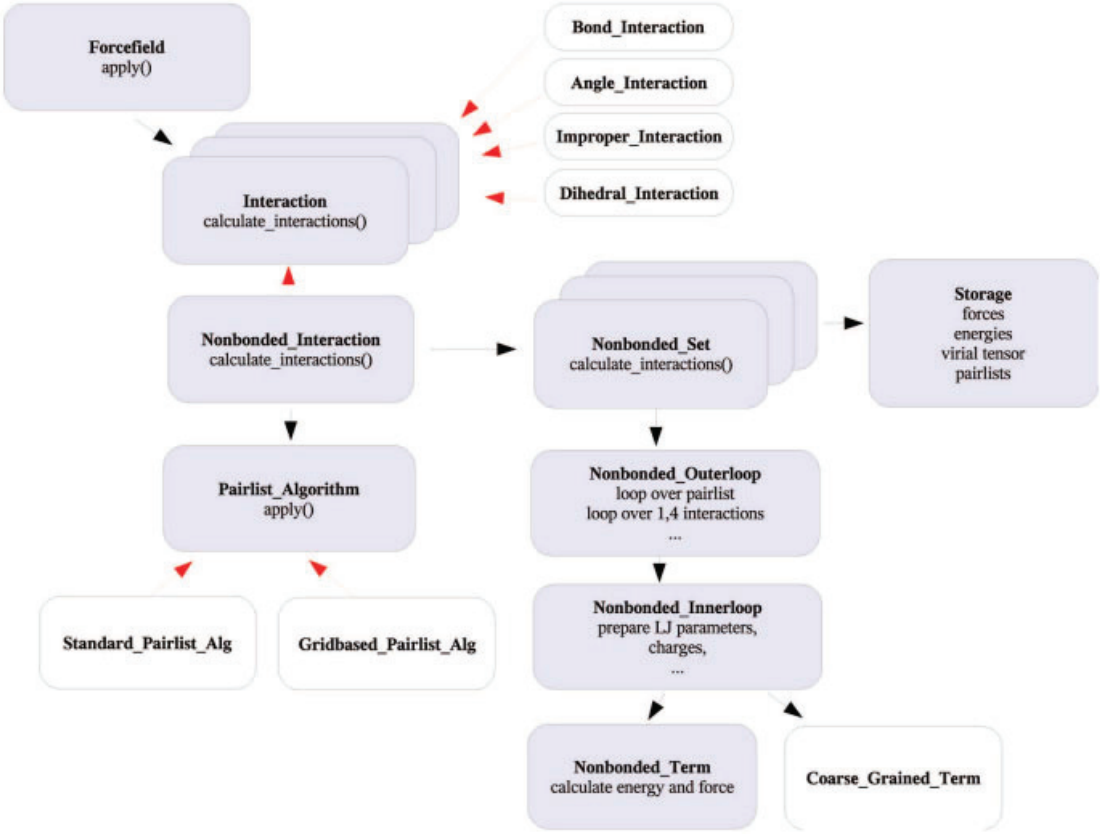

Figure 5. Illustration of the `Interaction` classes in MD++. The red arrows denote a *is-a* relationship, the black arrows *has-a.* All `Interaction` classes inherit from `Interaction` and, therefore, can be stored in the `Force field`, which is a `vector` of `Interaction` classes. The `Nonbonded_Interaction` consists of a `Pairlist_Algorithm` (either a `Standard_Pairlist_Algorithm` or a `Grid_Pairlist_Algorithm`) and (depending on parallelization) one or more `Nonbonded_Sets`. Those, in turn, consist of `Storage` (to locally store forces, energies, virial tensor, and pair lists) and an `Outerloop` (to calculate the interactions). The `Outerloop` relies on the `Innerloop` and on `Term` to calculate the interactions. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

of code (almost) completely separate from the nonbonded routines (see Table 2).

Analysis Modules: GROMOS++

All the FORTRAN analysis programs of GROMOS96 have been rewritten in C++. They accept a standard set of command line arguments to specify input. It is easy to add new analysis programs using the functionality provided within the GROMOS++ library. Following is a short description of the existing programs.

Setup of Simulations (Preprocessing)

- `make_top` builds a topology from a building block sequence.
- `com_top` combines two topologies.
- `con_top` converts topologies to a different force field parameter set.
- `red_top` reduces topologies by specified parts.
- `pt_top` combines topologies with perturbation topologies to produce new topologies or perturbation topologies.
- `pert_top` creates a perturbation topology to perturb specified atoms to dummies.
- `check_top` checks topologies for common mistakes.
- `pdb2g96` converts a pdb (Protein Data Bank) structure into GROMOS coordinates.
- `build_box` builds a simulation box containing *N* molecules at a specified density.
- `ran_box` builds a simulation box containing *N* molecules at a specified density, placing and orienting them randomly.
- `bin_box` builds a simulation box containing a binary mixture at a specified density.
- `sim_box` puts a simulation box around a molecule and fills it with solvent molecules from an equilibrated solvent configuration.

```
enum boundary_type {vacuum, rectangular, triclinic};
template<boundary_type boundary>
class Periodicity;

template<>
class Periodicity<vacuum>{
  public:
  void nearest_image(Vec const & v1, Vec const & v2,
                     Vec & v3);
};

template<>
class Periodicity<rectangular>{
  public:
    void nearest_image(Vec const & v1, Vec const & v2,
                       Vec & v3);
};

template<>
class Periodicity<triclinic>{
  public:
    void nearest_image(Vec const & v1, Vec const & v2,
                       Vec & v3);
};

template<boundary_type boundary>
class Interaction{
  public:
    virtual int calculate_interactions(
      Topology const & topology,
      Configuration & configuration,
      Simulation const & simulation){

        Vec v;
        Periodicity<boundary>
          periodicity(configuration.current().box);

        periodicity.nearest_image(
          configuration.current().pos(0),
          configuration.current().pos(1),
          v);

        // and so on
      }
};
```

Figure 6. Specialized code generation using templates.

- `ran_solvation` builds a simulation box around a molecule and fills it randomly with solvent molecules.
- `check_box` checks box properties (size).
- `copy_box` multiplies a box in any direction.
- `explode` increases intermolecule distances to vacuum conditions.
- `cry` applies rotations and translations to a system to create a crystal unit cell.
- `ion` replaces a specified number of solvent molecules by ions.
- `gch` generates hydrogen atom coordinates for a molecule.
- `gca` generates atomic Cartesian coordinates from a set of internal coordinates.
- `mk_script` prepares an MD, SD, or EM job script.

Analysis of Trajectories (Postprocessing)

- `tstrip` removes solvent from a trajectory.
- `filter` filters out specified atoms from a trajectory.
- `cog` calculates center of geometries for specified atoms.
- `tser` calculates time series of specified properties (distances, angles, torsions, order parameters, etc.).
- `tcf` calculates time correlation functions of time series.
- `dist` calculates distributions of specified properties.
- `ditrans` monitors dihedral-angle transitions.
- `propertyrmsd` calculates root-mean-square differences for a set of properties.
- `dipole` calculates dipole moments with respect to the center of molecules.
- `rmsd` calculates atom-positional root-mean-square differences between structures.
- `rmsf` calculates atom-positional root-mean-square fluctuations for specified atoms.
- `ene_ana` calculates averages, fluctuations, and error estimates for energies, pressure, and volume.
- `epsilon` calculates the dielectric permittivity for liquids.
- `visco` calculates the shear viscosity of liquids.
- `rgyr` calculates the radius of gyration.
- `rdf` calculates the radial distribution function for selected atoms.
- `mdf` gives the time series of the closest particle index to a selected atom.
- `m_widom` performs widom particle insertion.
- `sasa` calculates the solvent accessible surface area (SASA) for a specified part of a molecule.
- `hbond` analyses hydrogen bonding.
- `dssp` analyses secondary structure elements.
- `prep_noe` prepares for an NOE calculation.
- `noe` calculates NOE distances.
- `post_noe` analyses NOE distances.
- `oparam` calculates order parameters for lipids in membranes.
- `nhoparam` calculates N—H order parameters.
- `diffus` calculates the diffusion coefficient of specified atoms.
- `rmsdmat` calculates the RMSD between all structure pairs in a trajectory.
- `cluster` analyses an RMSD matrix to separate the structures into clusters.
- `postcluster` analyses the cluster output for lifetimes, folding pathways, and central-member structures.
- `iondens` calculates ion densities.
- `edyn` performs an essential dynamics analysis.
- `rot_rel` calculates the rotational relaxation time for solvent molecules.
- `ener` calculates any energy for a system.
- `espmap` calculates the vacuum electrostatic potential on a grid

```
int main(int argc, char **argv){

  Interaction<triclinic> interaction;
  interaction.calculate_interactions(
    topology, configuration, simulation);

  return 0;
}
```

Figure 7. Using the interaction class.

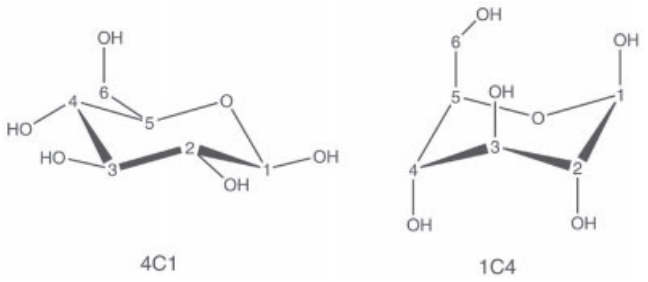

Figure 8. Glucose molecule with atom numbering.

around selected molecules of a given configuration from the partial charges in the topology.

Miscellaneous

- `frameout` converts trajectories into other formats or extracts snapshots from trajectories.
- `inbox` puts the solute into the center of the box.
- `atominfo` prints (topological) information on specified atoms.
- `shake_analysis` analyses a specified configuration.
- `cmt_list` lists atoms within a specified distance from a given atom.

Examples of Application

Local-Elevation Simulation of Glucose

The technique of local-elevation (LE) MD has been developed to enhance the searching of the configurational space of a molecule by progressively elevating the local potential energy of the configurations that are visited during an MD trajectory.[135] The total potential energy function consists of two terms: the standard physical terms $V_{\mathrm{phys}}(\mathbf{r}(t))$, and the local-elevation term $V_{\mathrm{LE}}(\mathbf{r}(t), t)$, which depends also explicitly on the time t. When the molecule

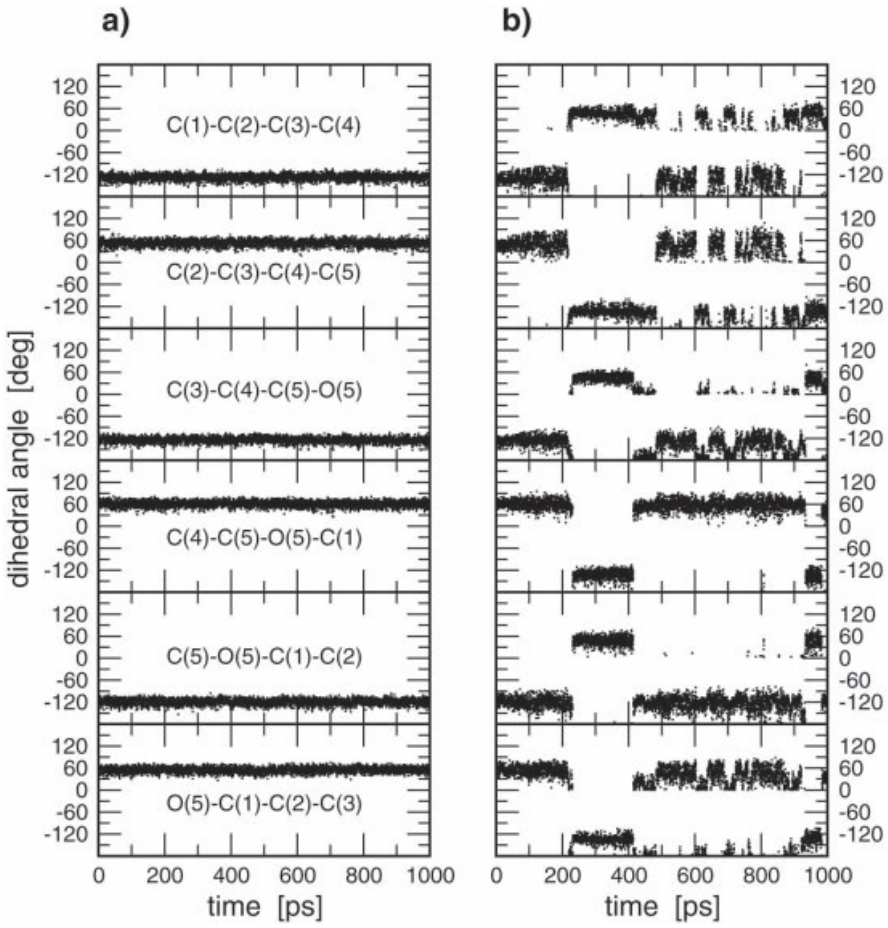

Figure 9. Time course of the six torsional dihedral angles of the glucose ring as obtained from standard MD (a) and local-elevation (LE-) MD (b). The LE weight factor was $E_{\phi}^{\mathrm{le}} = 2$ kJ mol^{-1} and the four torsional angles C(1)–C(2)–C(3)–C(4), C(2)–C(3)–C(4)–C(5), C(3)–C(4)–C(5)–O(5), and C(5)–O(5)–C(1)–C(2) were chosen as LE degrees of freedom.

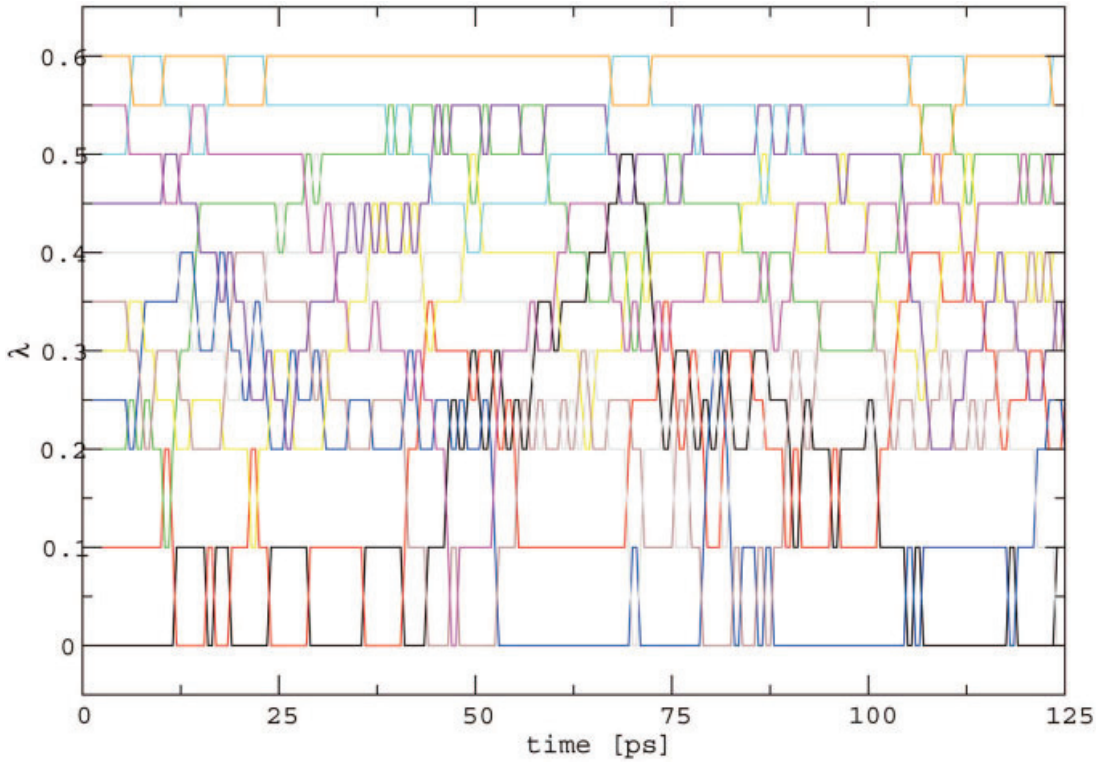

Figure 10. REMD of liquid butane, starting from an all *trans* configuration of the torsional angle. The path in λ space for the 11 replicas (starting at λ values 0.0, 0.1, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, and 0.6) is shown. Exchanges were attempted every 0.5 ps, in total 250. The overall exchange probability was 0.25. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

is trapped in a low-energy basin of the potential energy hypersurface, the LE algorithm gradually elevates the bottom of this basin using additional Gaussian-shaped energy functions, which will eventually force the molecular system out of the basin into a neighboring basin of the energy hypersurface. In this way, the energy surface is much more efficiently sampled than using standard MD. For low-dimensional systems LE-MD will lead to a flat potential energy as soon as all parts of the LE configuration space have been visited.[135] If

$$V_{\text{phys}}(\mathbf{r}(t)) + V_{\text{LE}}(\mathbf{r}(t), t) \tag{120}$$

is flat after a (long) time t_l, by construction of the local-elevation potential energy term, $V_{\text{LE}}(\mathbf{r}, t_l)$ represents the negative of the free-energy surface or potential of mean force for the LE degrees of freedom of the molecule.

LE-MD was already implemented in GROMOS96.[3,4] Here, we illustrate its sampling efficiency using as an example the conformational sampling of a glucose molecule (Fig. 8) solvated in SPC water.[136] The time course of the six exocyclic torsional dihedral angles of the sugar ring are shown for a standard MD and for LE-MD simulation in Figure 9. In the standard MD at 300 K and 1 atm, no conformational transitions are observed on a 1-ns time scale, while the LE-MD simulation with an energy weight factor that is raised by 2 kJ mol^{-1} every time a configuration is revisited already leads to a first conformational transition after 200 ps. After 500 ps many transitions are observed, indicating that the potential energy surface eq. 120 is becoming flat, the free-energy surface can then be obtained in the form of $-V_{\text{LE}}(\mathbf{r}, t > 1000$ ps).

Replica-Exchange Simulation of Butane

Five hundred butane molecules, all in *trans*-configuration, have been simulated at 273 K. The force constant of the torsional angle has been increased by a factor 3. The time to reach the equilibrium state of *gauche* and *trans* butane can be determined by monitoring the width of the torsional-angle distribution. The potential energy barrier between the *trans* and the *gauche* configuration is too high (≈18 kJ/mol) to overcome at 273 K. REMD is applied to increase the sampling of configurational space at 273 K. To this effect, 11 replicas of the system with changed torsional-angle force constants (scaled by 1.0, 0.9, 0.8, 0.75, 0.7, 0.65, 0.6, 0.55, 0.5, 0.45, and 0.4, respectively) were simulated simultaneously. Every 0.5 ps an exchange between two neighboring replicas was attempted. Figure 10 shows the path in λ space for all replicas. The overall exchange probability during the simulation was 0.25. The time series of the RMSD from the average of the torsional angle of all butane molecules is depicted in Figure 11. The larger the RMSD, the more

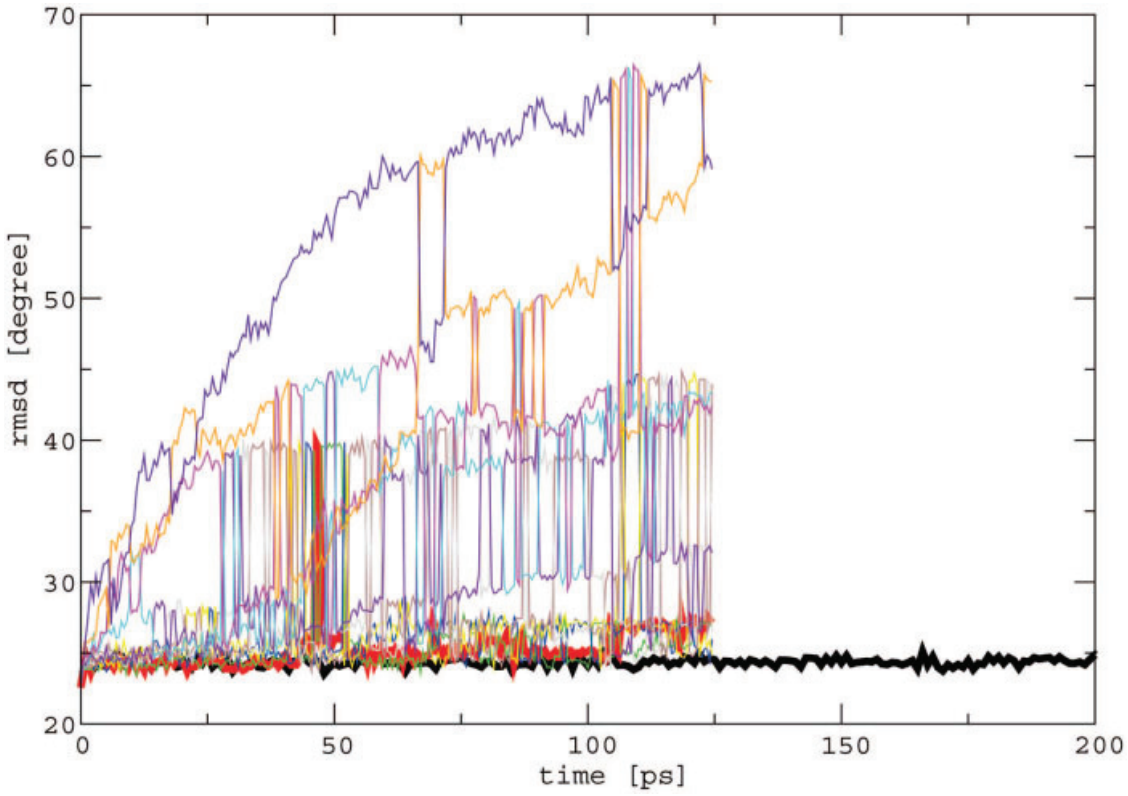

Figure 11. Simulation of liquid butane, starting from an all *trans* configuration of the torsional angle. The time series of the root-mean-square deviation (RMSD) from the average torsional angle is shown. The bold black line depicts the RMSD in the standard MD simulation. No broadening of the distribution is visible. The bold red line denotes the RMSD of the replica at $\lambda_i = 0.0$ (corresponding to the standard MD simulation). Clearly, the relaxation towards the equilibrium state is much faster using the replica-exchange method than in the standard MD simulation. The other lines denote the other replicas (at $0.0 < \lambda_i \leq 1.0$, many of them reaching their equilibrium state already after about 50 ps. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

torsional angle transitions from *trans* to *gauche* have happened. Through replica exchange, also the simulation running at $\lambda = 0$ contains butane molecules in a *gauche* conformation, unlike the simulation carried out without replica exchange. Note that the RMSD is dependent on the width of the valleys, so it is dependent on the force constant. This, in turn, means that the equilibrium value of the RMSD is different for the different replicas.

Coarse-Grained Simulation of Alkanes

Coarse-grained (CG) models allow for much more efficient sampling of the molecular configurational space than atomic-level (AL) models (at the expense of loss of atomic detail). Yet a CG model should be able to reproduce the properties of an AL model, assuming that the latter is correct. To illustrate this requirement for CG models, some properties (conformational distributions and configurational entropies) of *n*-alkanes in the liquid phase[137] have been compared. The main results are summarized here in the context of hexadecane, where the AL model was the standard GROMOS 45A3 force field[129] and the CG model the one discussed before.[116] For the AL model 128 hexadecane molecules were simulated at 323 K and 1 atm in a periodic box over 25 ns. For the CG model 512 molecules were simulated over 1000 ns under the same conditions. In the AL model, a hexadecane molecule consists of a linear chain of 16 united atoms. In the CG model, four united atoms are represented by one bead, so that hexadecane consists of four beads. To compare AL configurations of united atoms with CG configurations of beads, the AL trajectories were mapped to the CG level by considering only the centers of mass of the four united atoms that represent one bead at the CG level. This mapping of the atomic level onto the coarse-grained level is indicated by the symbol MAP.

Figure 12 shows the distribution of the values of the two pseudobond angles and the one pseudotorsional angle of the hexadecane molecules at the CG level for the MAP (gray) and CG (black) trajectories. The difference in torsional-angle distribution can be explained from the absence of torsional potential energy terms in the CG model.[116] Table 10 shows the configurational

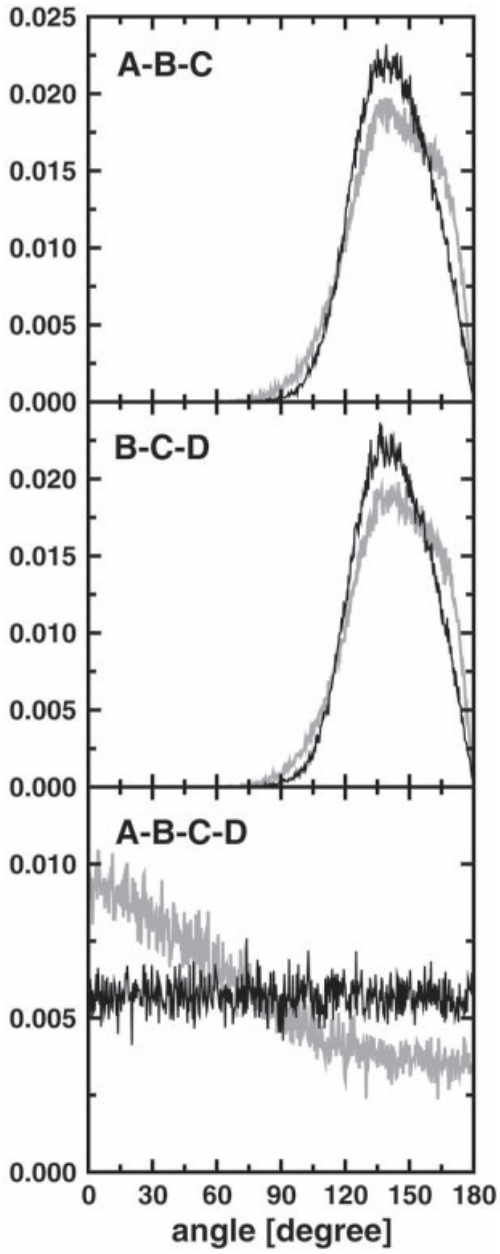

Figure 12. Bond-angle (A–B–C, B–C–D) and torsional dihedral angle (A–B–C–D) distributions at the coarse-grained level. Gray: A–D are centers of mass of fragments consisting of four united atoms as obtained from AL trajectories. Black: A–D are beads of the CG model obtained from CG simulations.

entropies of the four united atom fragments of the hexadecane molecules at the atomic level (AL) and at the CG level (MAP), together with those obtained from the CG simulations (CG). At the CG level the configurational entropies of MAP and CG models agree very well, to within 2 J mol^{-1} K^{-1}.

These data illustrate that the CG model[116] is able to reproduce the properties of the GROMOS AL model rather well.

One-Step Perturbation Calculations on the Free Energy of Ligand Binding to the Estrogen Receptor

The one-step approach to calculate relative free energies of complexation or ligand binding is particularly efficient when many structurally not too different ligands are to be considered. Previously,[124] the Gibbs free energy of binding of 17 polychlorinated biphenyls to the estrogen receptor were calculated from two MD simulations of an unphysical reference compound, one when bound to the protein and one free in solution. Here, the efficiency of the one-step technique is illustrated by calculating more than 1500 binding free energies from the two simulations.

Figure 13 shows the biphenyl ligand with the nine atoms that are made soft atoms in the unphysical reference state. At these nine soft sites, five different real substituents (H, F, Cl, Br, and CH_3) can be placed, yielding $5^9 - 1 = 1{,}953{,}124$ relative binding energies for the ligands. Here, we calculated free energies for all possible polyfluorinated, polychlorinated, and polybrominated ligands (in total $3 \cdot 2^9 - 1 = 1535$ relative free energies) from one simulation of the free ligand in water and one bound to the estrogen receptor. For every class of substituted biphenyl ligands, the three best binding structures are depicted in Figure 13. (We note that the binding affinity of the polybrominated biphenyls might be underestimated by the choice of the reference state: the soft-core atoms chosen have smaller van der Waals radii than the Bromine atoms.) This application illustrates the efficiency of the one-step perturbation technique for screening purposes in drug design.

Other Applications

GROMOS can be used for molecular modelling of any type of molecular system. Below, a number of applications are mentioned, which, for convenience, have mainly been taken from our own more recent work.

The structural stability of proteins,[138–144] peptides,[145–152] sugars,[126,153] and DNA[154,155] as function of their composition, chain lengths, and solvent environment or temperature and pressure can be studied. Solvation, both in pure solvents and in mixtures, can be investigated in atomic detail.[156–159] Motional properties, NMR coupling constants, and dielectric relaxation times can be analyzed.[139,160–162] ^3J-coupling constants, NOE's and NMR order parameters and CD spectra can be compared to experimental values.[163–167] GROMOS can also be used for structure refinement

Table 10. Configurational Entropy (in J K^{-1} mol^{-1}) of the Four (A–D) Hexadecane Fragments That Correspond to the Four Beads of the Coarse-Grained (CG) Model for Hexadecane in the Liquid Phase.

Fragment	AL	MAP	CG
A	211	133	131
B	209	111	110
C	209	111	110
D	211	133	131

AL: atomic-level entropies; MAP: fragment entropies from the AL trajectories; CG: bead entropies from the CG trajectories.

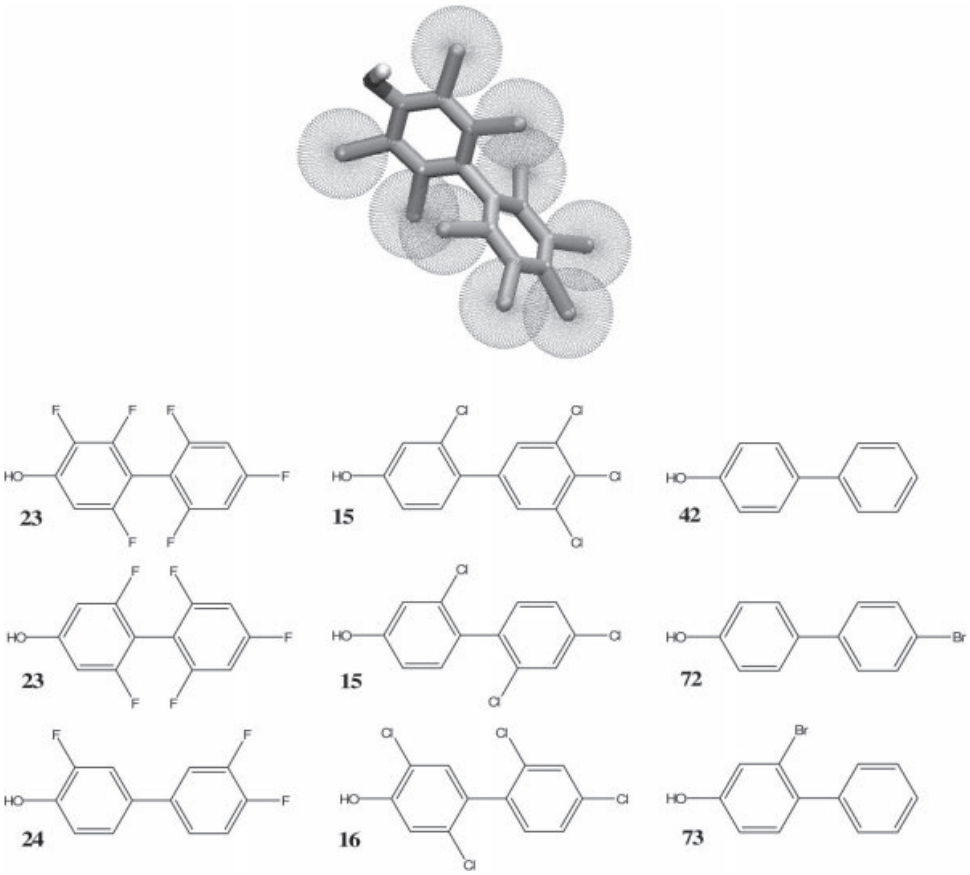

Figure 13. Polysubstituted byphenyls. Soft-core sites in the reference state are indicated as spheres. Of the $3 \cdot 2^9$ real ligands for which the relative free energy of binding to the estrogen receptor were calculated, the ones with lowest free energy of binding (in kJ mol^{-1}) are shown. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

of biomolecules based on NMR data.[168–170] Molecular host–guest complexes can be studied in terms of structural properties and free energy and entropy of binding.[123,124,127,171–173] Polypeptide (un)folding equilibria can be simulated in atomic detail.[174–179] Biochemical reactions can be mimicked in QM/MM simulations, in which interfaces to quantum chemistry software have to be used.[180–182]

A variety of types of molecules have been simulated: proteins, DNA, RNA, saccharides, lipids, and a range of solvents: water, DMSO, methanol, chloroform, carbontetrachloride, acetonitrile, and mixtures of these and other cosolvents such as urea.[183–186] Also, membranes and micelles have been simulated using GROMOS.[187–192]

Conclusions

The GROMOS software for biomolecular simulation has been extended with new functionality and extended analysis possibilities and put partially into C++, which makes extension of functionalities easier. GROMOS05 comes with the latest thermodynamically calibrated GROMOS force field parameter sets `45A3/4` and `53A5/6`, which are suitable for a broad range of molecular systems. The source code of GROMOS is obtainable for a nominal fee (http://www.igc.ethz.eh/gromos), and should allow both methodological investigations and structural, dynamical, and energetic explorations of biomolecular systems, which may lead to an enhanced understanding of the properties of such systems.

References

1. van Gunsteren, W. F.; Berendsen, H. J. C. Groningen Molecular Simulation (GROMOS) Library Manual, Biomos b.v.; University of Groningen, Groningen: The Netherlands, 1987.
2. Scott, W. R. P.; van Gunsteren, W. F. In Methods and Techniques in Computational Chemistry: METECC-95; Clementi, E.; Corongiu, G., Eds.; STEF: Cagliari, Italy, 1995, p. 397.
3. van Gunsteren, W. F.; Billeter, S. R.; Eising, A. A.; Hünenberger, P. H.; Krüger, P.; Mark, A. E.; Scott, W. R. P.; Tironi, I. G. Biomolecular Simulation: The GROMOS96 Manual and User Guide; Vdf Hochschulverlag AG an der ETH Zürich: Zürich, Switzerland.
4. Scott, W. R. P.; Hünenberger, P. H.; Tironi, I. G.; Mark, A. E.;

Billeter, S. R.; Fennen, J.; Torda, A. E.; Huber, T.; Krüger, P.; van Gunsteren, W. F. J Phys Chem A 1999, 103, 3596.
5. van Gunsteren, W. F.; Bakowies, D.; Bürgi, R.; Chandrasekhar, I.; Christen, M.; Daura, X.; Gee, P.; Glättli, A.; Hansson, T.; Oostenbrink, C.; Peter, C.; Pitera, J.; Schuler, L.; Soares, T.; Yu, H. Chimia 2001, 55, 856.
6. Bekker, H. J Comput Chem 1997, 18, 1930.
7. Hünenberger, P. H. Adv Polym Sci 2005, 173, 105.
8. Harvey, S. C.; Tan, R. K.-Z.; Cheatham, T. E., III. J Comput Chem 1998, 19, 726.
9. Chen, T.; Fowler, A.; Toner, M. Cryobiology 2000, 40, 277.
10. Amadei, A.; Chillemi, G.; Ceruso, M. A.; Grottesi, A.; Di Nola, A. J Chem Phys 2000, 112, 9.
11. Rugh, H. H. Phys Rev Lett 1997, 78, 772.
12. Butler, B. D.; Ayton, G.; Jepps, O. G.; Evans, D. J. J Chem Phys 1998, 109, 6519.
13. Hünenberger, P. H. J Chem Phys 2002, 116, 6880.
14. Oliva, B.; Hünenberger, P. H. J Chem Phys 2002, 116, 6898.
15. Graben, H. W.; Ray, J. R. Phys Rev A 1991, 43, 4100.
16. Bekker, H.; Ahlström, P. Mol Simulat 1994, 13, 367.
17. Bekker, H.; Berendsen, H. J. C.; van Gunsteren, W. F. J Comput Chem 1995, 16, 527.
18. Paci, E.; Marchi, M. J Phys Chem 1996, 100, 4314.
19. Woodcock, L. V. Chem Phys Lett 1971, 10, 257.
20. Berendsen, H. J. C.; Postma, J. P. M.; van Gunsteren, W. F.; DiNola, A.; Haak, J. R. J Chem Phys 1984, 81, 3684.
21. Nosé, S. J Chem Phys 1984, 81, 511.
22. Hoover, W. G. Phys Rev A 1985, 31, 1695.
23. Martyna, G. J.; Klein, M. L.; Tuckerman, M. J Chem Phys 1992, 97, 2635.
24. Hockney, R. W. Methods Comput Phys 1970, 9, 136.
25. Hoover, W. G.; Ladd, A. J. C.; Moran, B. Phys Rev Lett 1982, 48, 1818.
26. Evans, D. J. J Chem Phys 1983, 78, 3297.
27. Morishita, T. J Chem Phys 2000, 113, 2976.
28. Nosé, S. Mol Phys 1984, 52, 255.
29. Posch, H. A.; Hoover, W. G.; Vesely, F. J. Phys Rev A 1986, 33, 4253.
30. Jellinek, J.; Berry, S. R. Phys Rev A 1989, 40, 2816.
31. Hamilton, I. P. Phys Rev A 1990, 42, 7467.
32. Toxvaerd, S. Ber Bunsenges Phys Chem 1990, 94, 274.
33. Calvo, F.; Galindez, J.; Gadéa, F. X. J Phys Chem A 2002, 106, 4145.
34. D'Alessandro, M.; Tenenbaum, A.; Amadei, A. J Phys Chem B 2002, 106, 5050.
35. Anderson, H. C. J Chem Phys 1980, 72, 2384.
36. Parrinello, M.; Rahman, A. Phys Rev Lett 1980, 45, 1196.
37. Parrinello, M.; Rahman, A. J Phys Chem 1982, 76, 2662.
38. Ryckaert, J. P.; Ciccotti, G. J Chem Phys 1983, 78, 7368.
39. Nosé, S.; Klein, M. L. Mol Phys 1983, 50, 1055.
40. Heyes, D. M. Chem Phys 1983, 82, 285.
41. Finney, J. L. J Comput Chem 1978, 28, 92.
42. Streett, W. B.; Tildesley, D. J.; Saville, G. Mol Phys 1978, 35, 639.
43. van Gunsteren, W. F.; Berendsen, J. J. C. Angew Chem Int Ed Engl 1990, 29, 992.
44. Chialvo, A. A.; Debenedetti, P. G. Comput Phys Commun 1992, 70, 467.
45. Heinz, T.; Hünenberger, P. H. J Comput Chem 2004, 25, 1474.
46. Bekker, H.; Berendsen, H. J. C.; Dijkstra, E. J.; Achterop, S.; Drunen, R. v.; Spoel, D. v. d.; Sijbers, A.; Keegstra, H.; Reitsma, B.; Renardus, M. K. R. In Conf Proc Physics Computing '92, 257, World Scientific Publishing Co: Singapore.
47. Bekker, H. Thesis Rijksuniversiteit Groningen, 1996.
48. Barker, J. A.; Watts, R. O. Mol Phys 1973, 26, 789.
49. Barker, J. A. Mol Phys 1994, 83, 1057.
50. Tironi, I. G.; Sperb, R.; Smith, P. E.; van Gunsteren, W. F. J Chem Phys 1995, 102, 5451.
51. Hünenberger, P. H.; van Gunsteren, W. F. J Chem Phys 1998, 108, 6117.
52. Bergdorf, M.; Peter, C.; Hünenberger, P. H. J Chem Phys 2003, 119, 9129.
53. Berendsen, H. J. C.; van der Spoel, D.; van Drunen, R. Comput Phys Commun 1995, 91, 43.
54. Onsager, L. J Am Chem Soc 1936, 58, 1486.
55. Hünenberger, P. H.; McCammon, J. A. J Chem Phys 1999, 110, 1856.
56. Hünenberger, P. H. In Simulation and Theory of Electrostatic Interactions in Solution: Computational Chemistry, Biophysics and Aqueous Solution; Pratt, L. R.; Hummer, G., Eds.; AIP: New York, 1999, p. 17.
57. Weber, W.; Hünenberger, P. H.; McCammon, J. A. J Phys Chem B 2000, 104, 3668.
58. Kastenholz, M. A.; Hünenberger, P. H. J Phys Chem B 2004, 108, 774.
59. Hünenberger, P. H. J Chem Phys 2000, 113, 10464.
60. Heinz, T. N.; Hünenberger, P. H. J Chem Phys, submitted.
61. Redlack, A.; Grindlay, J. Can J Phys 1972, 50, 2815.
62. Euwema, R. N.; Surratt, G. T. J Phys Chem Solids 1975, 36, 67.
63. Redlack, A.; Grindlay, J. J Phys Chem Solids 1975, 36, 73.
64. Stuart, S. N. J Comput Phys 1978, 29, 127.
65. de Leeuw, S. W.; Perram, J. W.; Smith, E. R. Proc R Soc Lond A 1980, 373, 27.
66. de Leeuw, S. W.; Perram, J. W.; Smith, E. R. Proc R Soc Lond A 1980, 373, 57.
67. de Leeuw, S. W.; Perram, J. W. Physica A 1981, 107, 179.
68. Steinhauser, O. Chem Phys 1983, 79, 465.
69. de Leeuw, S. W.; Perram, J. W.; Smith, E. R. Proc R Soc Lond A 1983, 388, 177.
70. Allen, M. P.; Tildesley, D. J. Computer Simulations of Liquids; Oxford Science: Oxford, 1987.
71. Kusalik, P. G. J Chem Phys 1990, 93, 3520.
72. Caillol, J.-M. J Chem Phys 1994, 101, 6080.
73. Roberts, J. E.; Schnitker, J. J Chem Phys 1994, 101, 5024.
74. Roberts, J. E.; Schnitker, J. J Phys Chem 1995, 99, 1322.
75. Boresch, S.; Steinhauser, O. Ber Bunsenges Phys Chem 1997, 101, 1019.
76. Challacombe, M.; White, C.; Head–Gordon, M. J Chem Phys 1997, 107, 10131.
77. Boresch, S.; Steinhauser, O. J Chem Phys 1999, 111, 8271.
78. Boresch, S.; Ringhofer, S.; Höchtl, P.; Steinhauser, O. Biophys Chem 1999, 78, 43.
79. Vorobjev, Y. N.; Hermans, J. J Phys Chem 1999, 103, 10234.
80. Hünenberger, P. H. In Simulation and Theory of Electrostatic Interactions in Solution: Computational Chemistry, Biophysics, and Aqueous Solution; Hummer, G.; Pratt, L. R., Eds.; American Institute of Physics: New York, p. 17.
81. Peter, C.; van Gunsteren, W. F.; Hünenberger, P. H. J Chem Phys 2002, 116, 7434.
82. Kastenholz, M. A.; Hünenberger, P. H. J Chem Phys, submitted.
83. Ewald, P. P. Ann Phys 1921, 64, 253.
84. Hockney, R. W.; Eastwood, J. W. Computer Simulation Using Particles; Institute of Physics Publishing: Bristol, 1988, 2nd ed.
85. Wigner, E. Trans Faraday Soc 1938, 34, 678.
86. Felderhof, B. U. Physica A 1985, 130, 34.
87. Nijboer, B. R. A.; Ruijgrok, T. W. J Stat Phys 1988, 53, 361.
88. Deserno, M.; Holm, C. J Chem Phys 1988, 109, 7694.
89. Deserno, M.; Holm, C. J Chem Phys 1988, 109, 7678.

90. Hukushima, K.; Nemoto, K. J Phys Soc Jpn 1996, 65, 1604.
91. Hukushima, K.; Takayama, H.; Nemoto, K. Int J Mod Phys C 1996, 7, 337.
92. Swendsen, R. H.; Wang, J.-S. Phys Rev Lett 1986, 57, 2607.
93. Geyer, C. J. In Computing Science and Statistics, Proceedings 23rd Symp on the Interface; Keramidas, E. M., Eds.; Interface Foundation: Fairfax Station, 1991, p. 156.
94. Tesi, M. C.; van Rensburg, E. J. J.; Orlandini, E.; Whittington, S. G. J Stat Phys 1996, 82, 155.
95. Marinari, E.; Parisi, G.; Ruiz–Lorenzo, J. J. In Spin Glasses and Random Fields; Young, A. P., Eds.; World Scientific: Singapore, 1988, p. 59.
96. Irbäck, A.; Potthast, F. J Chem Phys 1995, 103, 10298.
97. Hansmann, U. H. E.; Okamoto, Y. Phys Rev E 1996, 54, 5863.
98. Irbäck, A.; Peterson, C.; Potthast, F.; Sommelius, O. J Chem Phys 1997, 107, 273.
99. Hansmann, U. H. E.; Okamoto, Y. J Comput Chem 1997, 18, 920.
100. Okabe, T.; Kawata, M.; Okamoto, Y.; Mikami, M. Chem Phys Lett 2001, 335, 435.
101. Zhou, R.; Berne, B. J.; Germain, R. Proc Natl Acad Sci USA 2001, 98, 14931.
102. Garcia, A. E.; Sambonmatsu, K. Y. Proteins Struct Funct Genet 2001, 42, 345.
103. Sanbonmatsu, K. Y.; Garcia, A. E. Proteins Struct Funct Genet 2002, 46, 225.
104. Pitera, J. W.; Swope W. Proc Natl Acad Sci USA 2003, 100, 7587.
105. Yang, W. Y.; Pitera, J. W.; Swope, W. C.; Gruebele, M. J Mol Biol 2004, 336, 241.
106. Swope, W. C.; Pitera, J. W.; Suits, F. J Phys Chem B 2004, 108, 6571.
107. Swope, W. C.; Pitera, J. W.; Suits, F.; Pitman, M.; Eleftheriou, M.; Fitch, B. G.; Germain, R. S.; Rayshubski, A.; Ward, T. J. C.; Zhestkov, Y.; Zhou, R. J Phys Chem B 2004, 108, 6582.
108. Sugita, Y.; Kitao, A.; Okamoto, Y. J Chem Phys 2000, 113, 6042.
109. Fukunishi, H.; Watanabe, O.; Takada, S. J Chem Phys 2002, 116, 9058.
110. Affentranger, R.; Tavernelli, I.; Di Iorio, E. E. Biophys J, submitted.
111. Smit, B.; Hilbers, P. A. J.; Esselink, K.; Rupert, L. A. M.; van Os, N. M.; Schlijper, A. G. Nature 1990, 348, 624.
112. Baschnagel, J.; Binder, K.; Doruker, P.; Gusev, A. A.; Hahn, O.; Kremer, K.; Mattice, W. L.; Müller–Plathe, F.; Murat, M.; Paul, W.; Santos, S.; Suter, U. W.; Tries, V. Adv Polym Sci 2000, 152, 41.
113. Shelley, J. C.; Shelley, M. Y. Curr Opin Colloid Interface Sci 2000, 5, 101.
114. Müller, M.; Katsov, K.; Schick, M. J Polym Sci Part B: Polym Phys 2003, 41, 1441.
115. Tozzini, V. Curr Opin Struct Biol 2005, 15, 144.
116. Marrink, S. J.; de Vries, A. H.; Mark, A. E. J Phys Chem B 2004, 108, 750.
117. Oostenbrink, C.; Villa, A.; Mark, A. E.; van Gunsteren, W. F. J Comp Chem 2004, 25, 1656.
118. van der Spoel, D.; van Buuren, A. R.; Apol, E.; Meulenhoff, P. J.; Tielemann, D. P.; Sijbers, A. L. T. M.; Hess, B.; Feenstra, K. A.; van Drunen, R.; Berendsen, H. J. C. Gromacs User Manual; The Netherlands; http://www.gromacs.org.
119. Beutler, T. C.; Mark, A. E.; van Schaik, R. C.; Gerber, P. R.; van Gunsteren, W. F. Chem Phys Lett 1994, 222, 529.
120. Liu, H.; Mark, A. E.; van Gunsteren, W. F. J Phys Chem 1996, 100, 9485.
121. Schäfer, H.; van Gunsteren, W. F.; Mark, A. E. J Comput Chem 1999, 20, 1604.
122. Pitera, J. W.; van Gunsteren, W. F. J Phys Chem B 2001, 105, 11264.
123. Oostenbrink, C.; van Gunsteren, W. F. J Comput Chem 2003, 24, 1730.
124. Oostenbrink, C.; van Gunsteren, W. F. Proteins 2004, 54, 234.
125. Oostenbrink, C.; van Gunsteren, W. F. Chem Eur J, 2005, 11, 4340.
126. Yu, H.; Amann, M.; Hansson, T.; Köhler, J.; Wich, G.; van Gunsteren, W. F. Carbohydr Res 2004, 339, 1697.
127. Oostenbrink, C.; van Gunsteren, W. F. Proc Natl Acad Sci USA 2005, 102, 6750.
128. Daura, X.; Mark, A. E.; van Gunsteren, W. F. J Comput Chem 1998, 19, 535.
129. Schuler, L. D.; Daura, X.; van Gunsteren, W. F. J Comput Chem 2001, 22, 1205.
130. Yu, H.; van Gunsteren, W. F. Comput Phys Commun, in press.
131. Straatsma, T. P.; McCammon, J. A. Mol Simulat 1990, 5, 181.
132. Yu, H.; Hansson, T.; van Gunsteren, W. F. J Chem Phys 2003, 118, 221.
133. Yu, H.; van Gunsteren, W. F. J Chem Phys 2004, 121, 9549.
134. Programming languages—C++, ISO 14882; 2003.
135. Huber, T.; Torda, A. E.; van Gunsteren, W. F. J Comput Aided Mol Design 1994, 8, 695.
136. Berendsen, H. J. C.; Postma, J. P. M.; van Gunsteren, W. F.; Hermans, J. In Intermolecular Forces; Pullman, B., Ed.; Reidel: Dordrecht, 1981, p. 331.
137. Baron, R.; de Vries, A. H.; Hünenberger, P. H.; van Gunsteren, W. F. J Phys Chem B, to be submitted.
138. Voordijk, S.; Hansson, T.; Hilvert, D.; van Gunsteren, W. F. J Mol Biol 2000, 300, 963.
139. Pitera, J. W.; Falta, M.; van Gunsteren, W. F. Biophys J 2001, 80, 2546.
140. Schäfer, H.; Smith, L. J.; Mark, A. E.; van Gunsteren, W. F. Proteins 2002, 46, 215.
141. Bakowies, D.; van Gunsteren, W. F. J Mol Biol 2003, 315, 713.
142. Antes, I.; Thiel, W.; van Gunsteren, W. F. Eur Biophys J 2002, 31, 504.
143. Smith, L. J.; Jones, R. M.; van Gunsteren, W. F. Proteins 2005, 58, 439.
144. van den Bosch, M.; Swart, M.; Snijders, J. G.; Berensen, H. J. C.; Mark, A. E.; Oostenbrink, C.; van Gunsteren, W. F.; Canters, G. W. ChemBioChem 2005, 6, 738.
145. Bonvin, A. M. J. J.; van Gunsteren, W. F. J Mol Biol 2000, 296, 255.
146. Peter, C.; Daura, X.; van Gunsteren, W. F. J Am Chem Soc 2000, 122, 7461.
147. Daura, X.; Gademann, K.; Schäfer, H.; Juan, B.; Seebach, D.; van Gunsteren, W. F. J Am Chem Soc 2001, 123, 2393.
148. Gee, P. J.; Hamprecht, F. A.; Schuler, L. D.; van Gunsteren, W. F.; Duchardt, E.; Schwalbe, H.; Albert, M.; Seebach, D. Helv Chim Acta 2002, 85, 618.
149. Yu, H.; Daura, X.; van Gunsteren, W. F. Proteins 2004, 54, 116.
150. Soares, T.; Christen, M.; Hu, K.; van Gunsteren, W. F. Tetrahedron 2004, 60, 7775.
151. Santiveri, C. M.; Jimènez, M. A.; Rico, M.; van Gunsteren, W. F.; Daura, X. J Peptide Sci 2004, 10, 546.
152. Glättli, A.; Seebach, D.; van Gunsteren, W. F. Helv Chem Acta 2004, 87, 2487.
153. Kony, D.; Damm, W.; Stoll, S.; Hünenberger, P. H. J Phys Chem B 2004, 108, 5815.
154. Czechtizky, W.; Daura, X.; Vasella, A.; van Gunsteren, W. F. Helv Chim Acta 2001, 84, 2132.
155. Soares, T. A.; Hünenberger, P. H.; Kastenholz, M. A.; Kräutler, V.; Lenz, T.; Lins, R. D.; Oostenbrink, C.; van Gunsteren, W. F. J Comp Chem 2005, 26, 725.
156. van der Vegt, N. F. A.; van Gunsteren, W. F. J Phys Chem B 2004, 108, 1056.

157. Trzesniak, D.; van der Vegt, N. F. A.; van Gunsteren, W. F. Phys Chem Chem Phys 2004, 6, 697.
158. van der Vegt, N. F. A.; Trzesniak, D.; Kasumaj, B.; van Gunsteren, W. F. Chem Phys Chem 2004, 5, 144.
159. Oostenbrink, C.; van Gunsteren, W. F. Pys Chem Chem Phys 2005, 7, 53.
160. Daura, X.; Haaksma, E.; van Gunsteren, W. F. J Comput Aided Mol Design 2000, 14, 507.
161. Walser, R.; van Gunsteren, W. F. Proteins 2001, 42, 414.
162. Peter, C.; Daura, X.; van Gunsteren, W. F. J Biomol NMR 2001, 20, 297.
163. Stocker, U.; van Gunsteren, W. F. Proteins 2000, 40, 145.
164. Glättli, A.; Daura, X.; Seebach, D.; van Gunsteren, W. F. J Am Chem Soc 2002, 124, 12972.
165. Daura, X.; Bakowies, D.; Seebach, D.; Fleischhauer, J.; van Gunsteren, W. F.; Krüger, P. Eur Biophys J 2003, 32, 661.
166. Soares, T. A.; Daura, X.; Oostenbrink, C.; Smith, L. J.; van Gunsteren, W. F. J Biomol NMR 2004, 30, 407.
167. Oostenbrink, C.; Soares, T. A.; van der Vegt, N. F. A.; van Gunsteren, W. F. Eur Biophys J 2005, 34, 273.
168. Stocker, U.; Juchli, D.; van Gunsteren, W. F. Mol Simulat 2003, 29, 123.
169. Peter, C.; Rüping, M.; Wörner, H. J.; Jaun, B.; Seebach, D.; van Gunsteren, W. F. Chem Eur J 2003, 9, 5838.
170. Glättli, A.; van Gunsteren, W. F. Angew Chem Int Ed Engl 2004, 43, 6312; Angew Chem 2004, 116, 6472.
171. Oostenbrink, B. C.; Pitera, J. W.; van Lipzig, M. M. H.; Meerman, J. H. N.; van Gunsteren, W. F. J Med Chem 2000, 43, 4594.
172. Dolenc, J.; Oostenbrink, C.; Koller, J.; van Gunsteren, W. F. Nucleic Acids Res 2005, 33, 725.
173. Hsu, S.-T. D.; Peter, C.; van Gunsteren, W. F.; Bonvin, A. M. J. J. Biophys J 2005, 88, 15.
174. van Gunsteren, W. F.; Bürgi, R.; Peter, C.; Daura, X. Angew Chemie Int Ed 2001, 40, 351.
175. Schäfer, H.; Daura, X.; Mark, A. E.; van Gunsteren, W. F. Proteins 2001, 43, 45.
176. Daura, X.; Glättli, A.; Gee, P.; Peter, C.; van Gunsteren, W. F. Adv Protein Chem 2002, 62, 341.
177. Baron, R.; Bakowies, D.; van Gunsteren, W. F.; Daura, X. Helv Chim Acta 2002, 85, 3872.
178. Baron, R.; Bakowies, D.; van Gunsteren, W. F. Angew Chem Int Ed Engl 2004, 43, 4055; Angew Chem 2004, 116, 4147.
179. Baron, R.; Bakowies, D.; van Gunsteren, W. F. J Peptide Sci 2005, 11, 74.
180. Billeter, S. R.; van Gunsteren, W. F. J Phys Chem A 2000, 104, 3276.
181. Berweger, C. D.; Thiel, W.; van Gunsteren, W. F. Proteins 2000, 41, 299.
182. Billeter, S. R.; Hanser, C. F. W.; Mordasini, T. Z.; Scholten, M.; Thiel, W.; van Gunsteren, W. F. Phys Chem Chem Phys 2001, 3, 688.
183. Glättli, A.; Daura, X.; van Gunsteren, W. F. J Chem Phys 2002, 116, 9811.
184. Glättli, A.; Daura, X.; van Gunsteren, W. F. J Comput Chem 2003, 24, 1087.
185. Smith, L. J.; Berendsen, H. J. C.; van Gunsteren, W. F. J Phys Chem A 2004, 108, 1065.
186. Geerke, D. P.; Oostenbrink, C.; van der Vegt, N. F. A.; van Gunsteren, W. F. J Phys Chem B 2004, 108, 1436.
187. Schuler, L. D.; Walde, P.; Luisi, P. L.; van Gunsteren, W. F. Eur Biophys J 2001, 30, 330.
188. Chandrasekhar, I.; van Gunsteren, W. F. Curr Sci 2001, 81, 1325.
189. Chandrasekhar, I.; Kastenholz, M.; Iins, R. D.; Oostenbrink, C.; Schuler, L. D.; Tieleman, D. P.; van Gunsteren, W. F. Eur Biophys J 2003, 32, 67.
190. Pereira, C. S.; Lins, R. D.; Chandrasekhar, I.; Freitas, L. C. G.; Hünenberger, P. H. Biophysical Journal 2004, 86, 2273.
191. Chandrasekhar, I.; Oostenbrink, C.; van Gunsteren, W. F. Soft Mater 2004, 2, 27.
192. de Vries, A. H.; Chandrasekhar, I.; van Gunsteren, W. F.; Hünenberger, P. H. J Phys Chem 2005, 109, 11643.