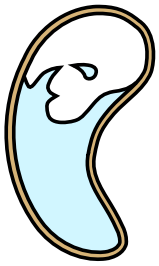

CASHEW 0.32
manual

Teemu Hynninen[1,2], Cristiano Dias[3], Mikko Karttunen[3], Adam Foster[1,2], Tapio Ala-Nissila[2]

[1]Tampere University of Technology, Finland
[2]Helsinki University of Technology, Finland
[3]University of Western Ontario, Canada

Contents

1 Introduction

This is a manual for the molecular dynamics (MD) implementation of the modified 3D Mercedes-Benz (MB) model for water [Dias et al., J. Chem. Phys. 131 (2009) 054505], CASHEW version 0.32 (Coarse Approach Simulator for Hydrogen Bonding Effects in Water). The code is written in Fortran 90 programming language. The code is provided as-it-is without technical support or guarantees. If you have comments or wish to report bugs, please send them to `teemu.hynninen@tut.fi`.

The modified 3D Mercedes-Benz model was originally developed due to the need for a coarse-grained water model for studying geometric effects in interactions of water with solutes, polymers, proteins etc. The model also reproduces well some important properties of water, such as melting at a lower temperature under increased pressure. This molecular dynamics implementation was created for studying the dynamical properties of the model.

The program is designed for MD simulations of MB water molecules, offering also the possibility to include other atomic particles with simple interactions and constraints. Both a serial and parallel versions of the program are available. The parallel version uses the Message Parsing Interface (MPI) set of communications routines, and distributes the computational load by particle pair decomposition.

This documentation is structured as follows. The physics of the model are described in section 2. The structure of the program, including compiling instructions, is summarized in section 3. User guide including the construction of the input file is given in section 4. Section 5 presents a brief version history.

2 Mercedes-Benz model

Although CASHEW can do molecular dynamics simulations of atomic particles with pair potentials, there are many more sophisticated pieces of software for that. The novel feature of the program is the implementation of the modified Mercedes-Benz model for water. The main physical features of the model are explained in this section.

The MB model of water is a coarse grained one, meaning that it drops some details in its description of water molecules in order to be computationally more efficient. Here, the water molecules are treated as rigid quasi-point-like particles. More importantly, the model does not contain long ranged electrostatic interactions. Because of this, one need not calculate as many point-to-point distances as in models which assign static charges to the water molecules. The coarse grained interactions are included via tetrahedrally coordinated dangling bonds which are assigned to the MB molecules to account for the hydrogen bonding in water. Computationally, the MB molecules are then described as points and a group of four unit vectors, as demonstrated in Figure 1. Since these unit vectors are rigid with respect to each other, it is only necessary to know the orientation of the molecule in order to determine the vectors. In CASHEW, the orientations and rotation are handled using quaternion representations.

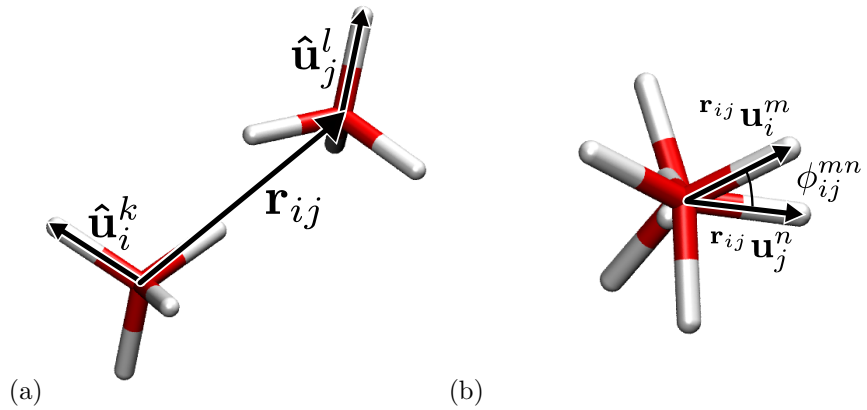

Figure 1: The Mercedes-Benz molecules are described as a point with four tetrahedrally coordinated unit vectors attached. These vectors represent dangling H-bonds. In principle two of them are H-atoms, though the model ignores the types of the dangling bonds. (a) The vector linking two molecules, i and j, is denoted $\mathbf{r}_{ij}$. The dangling bond vectors of molecule i are denoted $\hat{\mathbf{u}}_i^k, k = 1,\ldots,4$ (and similarly for molecule j). (b) The projections of vectors on the plane perpendicular to $\mathbf{r}_{ij}$ are denoted ${}^{\mathbf{r}_{ij}}\mathbf{u}$. The dihedral angles ϕ_{ij}^{mn} are defined as angles between the projections of the bond vectors of molecules i and j.

The details of the MB model are explained in the following subsections using these notations (cf. Fig. 1): indices i and j are the summation indices denoting MB molecules. The index α also refers to MB molecules, but it denotes the index of the molecule for which we are calculating the forces or torques (i.e., the molecule with respect to which we are differentiating). The relative positions of MB-particles i and j are denoted by $\mathbf{r}_{ij} = \mathbf{R}_j - \mathbf{R}_i$, the position of molecule i being $\mathbf{R}_i$ here. The dangling hydrogen bond "arms" (HB) of molecule i by are written $\hat{\mathbf{u}}_i^k$, $k = 1,\ldots,4$. The $\hat{}$ symbol is used for unit vectors, so we write the unit vector pointing from i to j as $\hat{\mathbf{r}}_{ij}$. The indices k and m are always the summation indices for the dangling bond vectors of molecule i (or α); l and n are reserved for molecule j. Projection of a unit vector $\hat{\mathbf{u}}$ on the plane defined as the normal plane to $\mathbf{r}$ is marked ${}^{\mathbf{r}}\mathbf{u}$. Such projections of the HB vectors are used for defining the dihedral angles of the molecules: The angle between the projections ${}^{\mathbf{r}_{ij}}\mathbf{u}_i^m$ and ${}^{\mathbf{r}_{ij}}\mathbf{u}_j^n$ is denoted ϕ_{ij}^{mn}.

2.1 MB potential

The potential energy of MB molecules comes from Lennard-Jones (LJ) and HB contributions:

$$U = \frac{1}{2}\sum_{\substack{i\\ j\neq i}} U_{ij} = \frac{1}{2}\sum_{\substack{i\\ j\neq i}} U_{ij}^{\mathrm{LJ}} + U_{ij}^{\mathrm{HB}} \tag{1}$$

$$U_{ij}^{\mathrm{LJ}} = 4\varepsilon_{\mathrm{LJ}}\left[\left(\frac{\sigma_{\mathrm{LJ}}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{\mathrm{LJ}}}{r_{ij}}\right)^{6}\right] \tag{2}$$

$$U_{ij}^{\mathrm{HB}} = \varepsilon_{\mathrm{HB}}\sum_{\substack{k=1\\ l=1}}^{4} b_i G_{ij}^{kl}\Phi_{ij}^{kl} \tag{3}$$

$$G_{ij}^{kl} = g\left(\frac{r_{ij}-R_{\mathrm{HB}}}{\sigma_{\mathrm{HB}}}\right) g\left(\frac{\hat{\mathbf{u}}_i^k\cdot\hat{\mathbf{r}}_{ij}-1}{\sigma_\theta}\right) g\left(\frac{\hat{\mathbf{u}}_j^l\cdot\hat{\mathbf{r}}_{ji}-1}{\sigma_\theta}\right) = g_{ij}^{\mathrm{HB}} g_{ijk}^{\theta} g_{jil}^{\theta} \tag{4}$$

$$g(x) = \exp\left(-\frac{1}{2}x^2\right) \tag{5}$$

$$\Phi_{ij}^{kl} = 1 + \frac{\varepsilon_\phi}{2}\sum_{\substack{m\neq k\\ n\neq l}}(1+\cos 3\phi_{ij}^{mn}) = 1 + \frac{\varepsilon_\phi}{2}\sum_{\substack{m\neq k\\ n\neq l}}(1+4\cos^3\phi_{ij}^{mn} - 3\cos\phi_{ij}^{mn}) \tag{6}$$

$$\begin{aligned}\cos\phi_{ij}^{mn} &= \frac{{}^{\mathbf{r}_{ij}}\mathbf{u}_i^m \cdot {}^{\mathbf{r}_{ij}}\mathbf{u}_j^n}{|{}^{\mathbf{r}_{ij}}\mathbf{u}_i^m||{}^{\mathbf{r}_{ij}}\mathbf{u}_j^n|} = \frac{\hat{\mathbf{u}}_i^m\cdot\hat{\mathbf{u}}_j^n + (\hat{\mathbf{u}}_i^m\cdot\hat{\mathbf{r}}_{ij})(\hat{\mathbf{u}}_j^n\cdot\hat{\mathbf{r}}_{ji})}{[(1-(\hat{\mathbf{u}}_i^m\cdot\hat{\mathbf{r}}_{ij})^2)(1-(\hat{\mathbf{u}}_j^n\cdot\hat{\mathbf{r}}_{ji})^2)]^{1/2}}\\ &= \frac{\hat{\mathbf{u}}_i^m\cdot\hat{\mathbf{u}}_j^n + c_{ij}^m c_{ji}^n}{[(1-(c_{ij}^m)^2)(1-(c_{ji}^n)^2)]^{1/2}} = \frac{\eta_{ij}^{mn}}{\xi_{ij}^{mn}}\end{aligned} \tag{7}$$

$${}^{\mathbf{r}}\mathbf{u} = \hat{\mathbf{u}} - (\hat{\mathbf{u}}\cdot\hat{\mathbf{r}})\hat{\mathbf{r}} \tag{8}$$

$$c_{ij}^m = \hat{\mathbf{u}}_i^m\cdot\hat{\mathbf{r}}_{ij},\ \eta_{ij}^{mn} = \hat{\mathbf{u}}_i^m\cdot\hat{\mathbf{u}}_j^n + c_{ij}^m c_{ji}^n,\ \xi_{ij}^{mn} = [(1-(c_{ij}^m)^2)(1-(c_{ji}^n)^2)]^{1/2} \tag{9}$$

$$b_i = \begin{cases} 1 & z_i \leq 4 \\ \left(\frac{4}{z_i}\right)^v & z_i > 4 \end{cases} \tag{10}$$

$$z_i = \sum_{q\neq i} f(r_{iq}) \tag{11}$$

$$f(r_{iq}) = \begin{cases} 1 & r_{iq} < R_b - D_b \\ \frac{1}{2}\left(1-\sin\frac{\pi(r_{iq}-R_b)}{2D_b}\right) & R_b - D_b \leq r_{iq} \leq R_b + D_b \\ 0 & R_b + D_b < r_{iq} \end{cases} \tag{12}$$

where the various parts and notations are

- (1) Total potential energy
- (2) Lennard-Jones potential
- (3) Hydrogen bond energy
- (4) Gaussian part of HB
- (5) Single gaussian function (not normalized)
- (6) Dihedral angle part of HB
- (7) Dihedral cosine
- (8) Projection of $\hat{\mathbf{u}}$ on plane perpendicular to $\mathbf{r}$
- (9) Shorthands
- (10) Bond order (Tersoff) part of HB (for discouraging too dense packing)
- (11) Bond count
- (12) Proximity function

2.2 MB forces

The force acting on molecule α is obtained by differentiating the total energy with respect to displacement of the molecule (∇_α).

$$\mathbf{F}_\alpha = -\nabla_\alpha U = -\frac{1}{2}\sum_{\substack{i\\ j\neq i}} \nabla_\alpha U_{ij}^{\mathrm{LJ}} + \nabla_\alpha U_{ij}^{\mathrm{HB}} = \mathbf{F}_\alpha^{\mathrm{LJ}} + \mathbf{F}_\alpha^{\mathrm{HB}} \tag{13}$$

$$\nabla_\alpha U_{ij}^{\mathrm{LJ}} = 4\varepsilon_{\mathrm{LJ}}\left(6\sigma_{\mathrm{LJ}}^6 r_{ij}^{-7} - 12\sigma_{\mathrm{LJ}}^{12} r_{ij}^{-13}\right)\nabla_\alpha r_{ij} \tag{14}$$

$$\nabla_\alpha r_{ij} = (\delta_{j\alpha} - \delta_{i\alpha})\hat{\mathbf{r}}_{ij} = \delta_{ij\alpha}\hat{\mathbf{r}}_{ij} \tag{15}$$

$$\nabla_\alpha U_{ij}^{\mathrm{HB}} = \varepsilon_{\mathrm{HB}}\sum_{k,l=1}^{4}(\nabla_\alpha b_i)G_{ij}^{kl}\Phi_{ij}^{kl} + b_i(\nabla_\alpha G_{ij}^{kl})\Phi_{ij}^{kl} + b_i G_{ij}^{kl}(\nabla_\alpha \Phi_{ij}^{kl}) \tag{16}$$

$$\nabla_\alpha G_{ij}^{kl} = (\nabla_\alpha g_{ij}^{\mathrm{HB}})g_{ijk}^\theta g_{jil}^\theta + g_{ij}^{\mathrm{HB}}(\nabla_\alpha g_{ijk}^\theta)g_{jil}^\theta + g_{ij}^{\mathrm{HB}} g_{ijk}^\theta(\nabla_\alpha g_{jil}^\theta) \tag{17}$$

$$\nabla_\alpha g_{ij}^{\mathrm{HB}} = -\frac{r_{ij} - R_{\mathrm{HB}}}{\sigma_{\mathrm{HB}}^2} g_{ij}^{\mathrm{HB}}\nabla_\alpha r_{ij} \tag{18}$$

$$\nabla_\alpha g_{ijk}^\theta = -\frac{\hat{\mathbf{u}}_i^k\cdot\hat{\mathbf{r}}_{ij} - 1}{\sigma_\theta^2} g_{ijk}^\theta \nabla_\alpha(\hat{\mathbf{u}}_i^k\cdot\hat{\mathbf{r}}_{ij}) = -\frac{\hat{\mathbf{u}}_i^k\cdot\hat{\mathbf{r}}_{ij} - 1}{\sigma_\theta^2} g_{ijk}^\theta \delta_{ij\alpha}\frac{\mathbf{r}_{ij}\mathbf{u}_i^k}{r_{ij}} \tag{19}$$

$$\nabla_\alpha \Phi_{ij}^{kl} = \frac{\varepsilon_\phi}{2}\sum_{\substack{m\neq k\\ n\neq l}}(12\cos^2\phi_{ij}^{mn} - 3)\nabla_\alpha\cos\phi_{ij}^{mn} \tag{20}$$

$$\nabla_\alpha\cos\phi_{ij}^{mn} = \frac{\xi_{ij}^{mn}\nabla_\alpha\eta_{ij}^{mn} - \eta_{ij}^{mn}\nabla_\alpha\xi_{ij}^{mn}}{(\xi_{ij}^{mn})^2} \tag{21}$$

$$\nabla_\alpha\eta_{ij}^{mn} = \frac{\delta_{ij\alpha}}{r_{ij}}[c_{ji}^n\,^{\mathbf{r}_{ij}}\mathbf{u}_i^m - c_{ij}^m\,^{\mathbf{r}_{ij}}\mathbf{u}_j^n] \tag{22}$$

$$\nabla_\alpha\xi_{ij}^{mn} = \frac{\delta_{ij\alpha}}{\xi_{ij}^{mn} r_{ij}}[c_{ij}^m(1-(c_{ji}^n)^2)\,^{\mathbf{r}_{ij}}\mathbf{u}_i^m - c_{ji}^n(1-(c_{ij}^m)^2)\,^{\mathbf{r}_{ij}}\mathbf{u}_j^n] \tag{23}$$

$$\nabla_\alpha b_i = \begin{cases} 0 & z_i \leq 4 \\ -4^v v z_i^{-v-1}\nabla_\alpha z_i & z_i > 4 \end{cases} \tag{24}$$

$$\nabla_\alpha z_i = \begin{cases} \sum_{q\neq\alpha}\frac{\pi}{4D_b}\cos\frac{\pi(r_{\alpha q}-R_b)}{2D_b}\hat{\mathbf{r}}_{\alpha q}\delta_{r_{\alpha q}\in[R_b-D_b,R_b+D_b]} & i=\alpha \\ \frac{\pi}{4D_b}\cos\frac{\pi(r_{\alpha i}-R_b)}{2D_b}\hat{\mathbf{r}}_{\alpha i}\delta_{r_{\alpha i}\in[R_b-D_b,R_b+D_b]} & i\neq\alpha \end{cases}, \tag{25}$$

that is,[1]

$$\mathbf{F}_\alpha^{\mathrm{LJ}} = 2\varepsilon_{\mathrm{LJ}}\sum_{j\neq\alpha}\left(6\sigma_{\mathrm{LJ}}^6 r_{\alpha j}^{-7} - 12\sigma_{\mathrm{LJ}}^{12} r_{\alpha j}^{-13}\right)\hat{\mathbf{r}}_{\alpha j} \tag{26}$$

$$\mathbf{F}_\alpha^{\mathrm{HB}} = \mathbf{F}_\alpha^{\mathrm{pair}} + \mathbf{F}_\alpha^{\mathrm{many}} \tag{27}$$

$$= -\frac{\varepsilon_{\mathrm{HB}}}{2}\sum_{\substack{i\\ j\neq i}} b_i\sum_{k,l=1}^{4}(\nabla_\alpha G_{ij}^{kl})\Phi_{ij}^{kl} + G_{ij}^{kl}(\nabla_\alpha\Phi_{ij}^{kl}) - \frac{\varepsilon_{\mathrm{HB}}}{2}\sum_{\substack{i\\ j\neq i}}(\nabla_\alpha b_i)\sum_{k,l=1}^{4} G_{ij}^{kl}\Phi_{ij}^{kl} \tag{28}$$

[1] This form of $\mathbf{F}_\alpha^{\mathrm{many}}$ is nicely symmetric, but computationally it is probably better to avoid recalculating $G_{ij}^{kl}\Phi_{ij}^{kl}$'s and make the innermost sum over the cosines.

$$
\begin{aligned}
\mathbf{F}_\alpha^{\text{pair}} &= -\frac{\varepsilon_{\text{HB}}}{2}\sum_{j\neq\alpha}(b_\alpha+b_j)\sum_{k,l=1}^{4}G_{\alpha j}^{kl}\times \\
&\left[\Phi_{\alpha j}^{kl}\left(-\frac{r_{\alpha j}-R_{\text{HB}}}{\sigma_{\text{HB}}^2}\hat{\mathbf{r}}_{\alpha j}+\frac{1}{\sigma_\theta^2 r_{\alpha j}}[(c_{\alpha j}^k-1)^{\,\mathbf{r}_{\alpha j}}\mathbf{u}_\alpha^k+(c_{j\alpha}^l-1)^{\,\mathbf{r}_{\alpha j}}\mathbf{u}_j^l]\right)-\right. \\
&\frac{\varepsilon_\phi}{2}\sum_{\substack{m\neq k\\ n\neq l}}\frac{\left(12\cos^2\phi_{\alpha j}^{mn}-3\right)}{r_{\alpha j}(\xi_{\alpha j}^{mn})^3}\times \qquad (29)\\
&\left.\left([c_{j\alpha}^n(\xi_{\alpha j}^{mn})^2+c_{\alpha j}^m(1-(c_{j\alpha}^n)^2)\eta_{\alpha j}^{mn}]^{\,\mathbf{r}_{\alpha j}}\mathbf{u}_\alpha^m-[c_{\alpha j}^m(\xi_{\alpha j}^{mn})^2+c_{j\alpha}^n(1-(c_{\alpha j}^m)^2)\eta_{\alpha j}^{mn}]^{\,\mathbf{r}_{\alpha j}}\mathbf{u}_j^n\right)\right] \\
\mathbf{F}_\alpha^{\text{many}} &= 4^{v-1}v\frac{\pi\varepsilon_{\text{HB}}}{2D_b}\sum_{q\neq\alpha}\cos\frac{\pi(r_{\alpha q}-R_b)}{2D_b}\delta_{r_{\alpha q}\in[R_b-D_b,R_b+D_b]}\hat{\mathbf{r}}_{\alpha q}\times \\
&\left[z_\alpha^{-v-1}\delta_{z_\alpha>4}\sum_{j\neq\alpha}\sum_{k,l=1}^{4}G_{\alpha j}^{kl}\Phi_{\alpha j}^{kl}+z_q^{-v-1}\delta_{z_q>4}\sum_{j\neq q}\sum_{k,l=1}^{4}G_{qj}^{kl}\Phi_{qj}^{kl}\right]. \qquad (30)
\end{aligned}
$$

2.3 MB torques

As the MB molecules are treated as rigid particles that can also rotate in 3D space, one needs to know the torques acting on them due to the MB potential. Similarly to the forces, the torque acting on molecule α is obtained by differentiating the potential with respect to rotation of the particle (∇_α^φ). Note though, that $\mathbf{r}_{ij}$ is constant with respect to rotations, and only terms including HB vectors $\hat{\mathbf{u}}_i^k$ etc. give non-zero derivatives. Also note that the torques are not symmetric: the interaction between two particles will in general result in different torques for the two. This is due to the asymmetric $\hat{\mathbf{u}}\cdot\hat{\mathbf{r}}$ terms.

$$
\begin{aligned}
\mathbf{T}_\alpha &= -\nabla_\alpha^\varphi U=-\frac{1}{2}\sum_{\substack{i\\ j\neq i}}\nabla_\alpha^\varphi U_{ij}^{\text{HB}} && (31)\\
\nabla_\alpha^\varphi U_{ij}^{\text{HB}} &= \varepsilon_{\text{HB}}\sum_{k,l=1}^{4}b_i(\nabla_\alpha^\varphi G_{ij}^{kl})\Phi_{ij}^{kl}+b_iG_{ij}^{kl}(\nabla_\alpha^\varphi\Phi_{ij}^{kl}) && (32)\\
\nabla_\alpha^\varphi G_{ij}^{kl} &= g_{ij}^{\text{HB}}(\nabla_\alpha^\varphi g_{ijk}^\theta)g_{jil}^\theta+g_{ij}^{\text{HB}}g_{ijk}^\theta(\nabla_\alpha^\varphi g_{jil}^\theta) && (33)\\
\nabla_\alpha^\varphi g_{ijk}^\theta &= -\frac{\hat{\mathbf{u}}_i^k\cdot\hat{\mathbf{r}}_{ij}-1}{\sigma_\theta^2}g_{ijk}^\theta\nabla_\alpha^\varphi(\hat{\mathbf{u}}_i^k\cdot\hat{\mathbf{r}}_{ij})=-\frac{\hat{\mathbf{u}}_i^k\cdot\hat{\mathbf{r}}_{ij}-1}{\sigma_\theta^2}g_{ijk}^\theta\delta_{i\alpha}(\hat{\mathbf{u}}_i^k\times\hat{\mathbf{r}}_{ij}) && (34)\\
\nabla_\alpha^\varphi\Phi_{ij}^{kl} &= \frac{\varepsilon_\phi}{2}\sum_{\substack{m\neq k\\ n\neq l}}(12\cos^2\phi_{ij}^{mn}-3)\nabla_\alpha^\varphi\cos\phi_{ij}^{mn} && (35)\\
\nabla_\alpha^\varphi\cos\phi_{ij}^{mn} &= \frac{\xi_{ij}^{mn}\nabla_\alpha^\varphi\eta_{ij}^{mn}-\eta_{ij}^{mn}\nabla_\alpha^\varphi\xi_{ij}^{mn}}{(\xi_{ij}^{mn})^2} && (36)\\
\nabla_\alpha^\varphi\eta_{ij}^{mn} &= \delta_{ij\alpha}(\hat{\mathbf{u}}_i^m\times\hat{\mathbf{u}}_j^n)-\delta_{i\alpha}c_{ji}^n(\hat{\mathbf{u}}_i^m\times\hat{\mathbf{r}}_{ij})-\delta_{j\alpha}c_{ij}^m(\hat{\mathbf{u}}_j^n\times\hat{\mathbf{r}}_{ji}) && (37)\\
\nabla_\alpha^\varphi\xi_{ij}^{mn} &= -\frac{\delta_{i\alpha}}{\xi_{ij}^{mn}}(1-(c_{ji}^n)^2)c_{ij}^m(\hat{\mathbf{u}}_i^m\times\hat{\mathbf{r}}_{ij})-\frac{\delta_{j\alpha}}{\xi_{ij}^{mn}}(1-(c_{ij}^m)^2)c_{ji}^n(\hat{\mathbf{u}}_j^n\times\hat{\mathbf{r}}_{ji}) && (38)
\end{aligned}
$$

Simplifying, this can be written

$$
\begin{aligned}
\mathbf{T}_\alpha &= -\frac{\varepsilon_{\text{HB}}}{2}\sum_{j\neq\alpha}(b_\alpha+b_j)\sum_{k,l=1}^{4}G_{\alpha j}^{kl}\left[\frac{1}{\sigma_\theta^2}\Phi_{\alpha j}^{kl}(c_{\alpha j}^k-1)(\hat{\mathbf{r}}_{\alpha j}\times\hat{\mathbf{u}}_\alpha^k)+\right. \\
&\frac{\varepsilon_\phi}{2}\sum_{\substack{m\neq k\\ n\neq l}}\left(3-12\cos^2\phi_{\alpha j}^{mn}\right)\times \\
&\left.\left(\left(\frac{c_{j\alpha}^n}{\xi_{\alpha j}^{mn}}+\frac{c_{\alpha j}^m}{1-(c_{\alpha j}^m)^2}\cos\phi_{\alpha j}^{mn}\right)(\hat{\mathbf{r}}_{\alpha j}\times\hat{\mathbf{u}}_\alpha^m)+\frac{1}{\xi_{\alpha j}^{mn}}(\hat{\mathbf{u}}_j^n\times\hat{\mathbf{u}}_\alpha^m)\right)\right]. \qquad (39)
\end{aligned}
$$

3 Program files and compiling

3.1 Program files

The source code is split to the following 8 files:

- `Cashew.F90` - main program, includes MD routines (`PROGRAM cashew`)
- `IO.F90` - module for input and output (`MODULE file_handler`)
- `Model.F90` - module for physics of the MB model (`MODULE mb_model`)
- `Functions.F90` - module for some commonly used functions (`MODULE functions`)
- `Quaternions.F90` - module for quaternion and vector algebra, representing 3D rotations (`MODULE quaternions`)
- `Parallel.F90` - module for MPI-related variables and functions (`MODULE mpi_mod`)
- `Params.F90` - module for parameters and constants (`MODULE parameters`)
- `Mersenne.F90` - module for random numbers (`MODULE mt95`)

3.2 Input and output files

Every simulation has a name provided as a command line parameter (see Section 4). Let the name of the simulation be, say `system`. Then the i/o-files will be named as follows:

- `system.mb` - main input
- `system.out` - main output
- `system_end.mb` - continuation input (also `system_start.mb` or `system_mid*.mb`)
- `system.xyz` - xyz geometry
- `system_stats.out` - general statistics (also `system_stats_*.out`)
- `system_rdf.out` - radial distribution function
- `system_adf.out` - axial distribution function

3.3 Compiling

The source code contains the necessary elements for both serial and parallel versions of the program. By default, a serial version will be compiled. To compile in parallel mode, one must provide the preprocessor option `MPI`, as explained below.

Compile the code with the command `f90 -o cashew_serial Mersenne.F90 Params.F90 Parallel.F90 Quaternions.F90 Functions.F90 Model.F90 IO.F90 Cashew.F90` to create a serial executable, called `cashew_serial` here. Replace `f90` above with the name of your Fortran 90 compiler of choice. Optimization options depend on the compiler, please test them to enhance the code performance.

The parallel version of the program must be compiled in an MPI environment, in which case the compilation command is `f90 -D MPI -o cashew_parallel Mersenne.F90 Params.F90 Parallel.F90 Quaternions.F90 Functions.F90 Model.F90 IO.F90 Cashew.F90`.

Known issues:

- high-level optimization with the PGI compiler breaks file reading routines in `IO.F90` — GNU is known to work at least with -O3

4 User guide

4.1 Running

If you created an executable named `cashew_executable`, run the program in the directory where the executable is located via `./cashew_executable system` where `system` is the name of your simulation. The program will then read the file `system.mb` (in the same directory) for input and write main output to `system.out`. Other output files, written if demanded, include `system_stats.out`, `system_rdf.out` and `system.xyz`.

4.2 Input file

The program reads one file which should contain all necessary information for setting up and running the simulation. This file should in general be in the directory from which you launch the program. The name of the simulation must be given as a command-line parameter, and this decides also the name of the input file: the file name must be the name of the simulation, given as parameter, followed by the extension `.mb`.

The input is not case sensitive, but it must follow a certain predefined, though flexible, structure. The file must be divided in blocks, defined by tags. A block starts with a tag such as `<particles>` and ends with a tag such as `</particles>`. Anything outside such tags is ignored. One can also include comments anywhere in the input file using the comment character "`!`". The names of the blocks recognized in the input file are:

- `<control>` – control parameters, required
- `<mb-model>` – parameters of the MB model, required
- `<cell>` – supercell parameters, optional
- `<elements>` – names and masses of particles other than MB molecules, optional
- `<particles>` – types and starting positions of particles, required
- `<velocities>` – initial velocities of particles, optional
- `<constraints>` – constraints, optional
- `<potentials>` – atom-atom and MB-atom interactions, optional
- `<statistics>` – statistics to be recorded, optional

In the following subsections, the formats of these blocks are explained. In the formatting examples, expected real values are marked `R*`, integers `I*` and keywords `K*`. If a value is optional or its necessity depends on the preceding options, it is put in square brackets `[ ]`.

4.2.1 `<control>` block

Format:
```
K* R*/I*/K*
parameter value
```

Example:
```
<control>
 temperature 300.0
 algo verlet
 ...
</control>
```

Default: -

The block `<control>` defines control variables and physical parameters not directly related to the MB model. To define a parameter, one must simply write its name and its value on the same line, separated by one or several spaces.
Names of parameters[2]:

- `temperature` – target temperature [K]

[2] CASHEW can currently use two sets of units, real and reduced, and the choice affects the numerical value of the Boltzmann constant used. For parameters with a dimension, the expected units for the real unit scale are given.

- `pressure` – target pressure [eV/Å^3]
- `tsimu` – total simulation time [fs]
- `tstep` – simulation time step [fs]
- `algo` – simulation algorithm
- `cgtol` – convergence criterion for conjugate gradients
- `cgsteps` – maximum number of steps for conjugate gradients
- `thermo` – thermostat
- `baro` – barostat
- `run` – type of run (test or full)
- `rndseed` – seed for the random number generator
- `verbose` – control for the amount of output
- `writexyz` – control for writing xyz-files
- `writeinp` – control for writing input files for continuation simulations
- `units` – switch for reduced and real units

Temperature

Format:	`temperature R*`
Example:	`temperature 300.00`
Default:	`temperature 273.15`

The target temperature of the system. Unless initial velocities are explicitly given, they are randomly generated to approximately match this temperature. If a thermostat is applied, it will drive the system towards this temperature. For real units, the temperature should be given in Kelvins. Although there is a default value, the user should always specify the temperature.

Pressure

Format:	`pressure R*`
Example:	`pressure 1.0E-5`
Default:	`pressure 1.0E-6`

The target pressure of the system. This value has meaning only if a barostat is applied. Currently, only the Berendsen barostat is available. For real units, the pressure should be given in eV/Å^3.

Algorithm

	Format:	`algo K*`
	Example:	`algo verlet`
	Default:	`algo verlet`
Options:	`verlet`	velocity Verlet
	`leapfrog`	leap-frog (limited implementation)
	`cg`	conjugate gradients (limited implementation)

The algorithm for molecular dynamics or conjugate gradients. For MD, the algorithm of choice is velocity Verlet (`verlet`) for which the positions ($\mathbf{r}$) and velocities ($\mathbf{v}$) are updated using

$$\mathbf{r}(t+\Delta t) = \mathbf{r}(t) + \mathbf{v}(t)\Delta t + \frac{1}{2}\frac{\mathbf{F}(t)}{m}\Delta t^2 \quad (40)$$

$$\mathbf{v}(t+\Delta t) = \mathbf{v}(t) + \frac{1}{2m}(\mathbf{F}(t) + \mathbf{F}(t+\Delta t))\Delta t, \quad (41)$$

where $\mathbf{F}$ is force, m is mass and t is time. The time step Δt is specified by the parameter `tstep`. Leap-frog (`leapfrog`) is a variation of the Verlet scheme, but the current implementation of this

algorithm does not support e.g. thermostats. The program does not warn the user if the use of leap-frog overrides some other options, so it is always recommended to use Verlet!

Besides MD, one can also do simple conjugate gradients (`cg`) minimum energy searches. Whereas MD simulations are controlled by parameters `tstep` and `tsimu`, CG obeys `cgtol` and `cgsteps`. The implementation for conjugate gradients is experimental and untested. We do not guarantee that it works!

MD simulation time

Format:	`tsimu R*`
Example:	`tsimu 10000.0`
Default:	`-`

The total length of the simulation in system timescale. This parameter determines how long the simulation will last: the program will complete `tsimu`/`tstep` time steps before ending the simulation. For real units, the simulation time should be given in femtoseconds.

MD timestep

Format:	`tstep R*`
Example:	`tstep 1.0`
Default:	`-`

The time step used for numerical integration of the equations of motion (in system timescale). I.e., the parameter specifies how much the system time advances in one MD step. For real units, the time step should be given in femtoseconds.

CG tolerance

Format:	`cgtol R*`
Example:	`cgtol -0.02`
Default:	`cgtol 0.0001`

Note: CG implementation is limited! The convergence criterion for conjugate gradients searches. If a positive real is given, it will be read as the energy difference tolerance between two configurations: when the CG step finds a new configuration whose energy differs from that of the previous configuration by less than the specified tolerance, the search is considered converged. If a negative real is given, its absolute value will be read as the maximum force tolerance: when a configuration is found where all forces are below this tolerance, the search is considered converged. For real units, the energies should be given in eV and the forces in eV/Å.

CG maximum iterations

Format:	`cgsteps I*`
Example:	`cgsteps 30`
Default:	`cgsteps 100`

Note: CG implementation is limited! The maximum number of iterations allowed for a CG search before the search is aborted.

Thermostat

Format:	`thermo K* [R*]`
Example:	`thermo langevin 0.01`
Default:	`thermo none`

Options:	`none`	no thermostat
	`langevin` *friction*	Langevin thermostat
	`cooler` *friction*	friction

The thermostat for MD simulations. By default, simulations are carried out in the microcanonical ensemble with no thermostats (`none`). For constant temperature simulations, the Langevin thermostat is available (`langevin`). This thermostat changes the Newtonian equations of motion of the particles to the Langevin equations of motion

$$m\mathbf{a} = \mathbf{F} - \gamma m\mathbf{v} + \mathbf{f}, \tag{42}$$

by adding a friction coefficient γ and a random force $\mathbf{f}$ whose effects balance out on average at a given temperature. (Here, $\mathbf{a}$ is acceleration.) The value of the friction coefficient must be given by the user. For real units, the value should be in 1/fs.

A third option called `cooler` is also provided. This will add the friction coefficient but leave out the random force in the Langevin equation. As a result, the system loses energy and cools down. This may be used for minimum energy searches although it is not a very efficient scheme for that purpose.

Barostat

Format:	`baro K* [R* [I*]]`
Example:	`baro berendsen 10.0`
Default:	`baro none`

Options:	`none`	no barostat
	`berendsen` *`tau`* `[`*`interval`*`]`	Berendsen barostat

The barostat for MD simulations. By default, no barostat is used (`none`). One also can use the simple Berendsen barostat (`berendsen`) for pressure control. This barostat scales the system volume and coordinates according to the measured pressure (P) by the factor

$$\mu = \left[1 - \frac{\Delta t}{\tau_P}(P - P_0)\right]^{1/3}, \tag{43}$$

where P_0 is the target pressure, Δt is the time interval between the scalings, and τ_P is the barostat time constant. The time constant must be given by the user. Optionally, the user may also specify the time interval (as an integer — the program multiplies it by the value of the MD time step) for the application of the barostat. By default, the barostat is applied every 10 steps and the average pressure for the interval is used for scaling. The time constant should be given in fs, when real units are used.

Note that the Berendsen barostat may lead to rapidly oscillating system volume. Therefore, it is recommended to not use the barostat for production simulations.

Run type

Format:	`run K*`
Example:	`run test`
Default:	`run full`

Options:	`full`	full simulation
	`test`	initialization only

A switch for full simulations (`full`) and test runs (`test`). In a test run, the system is initialized and any requested pre-run output is produced, but the actual simulation is not started. Test runs can be used for checking the validity of input files and producing quick analyses of given systems.

Random seed

Format:	`rndseed I*`
Example:	`rndseed 5801`
Default:	*`random`*

Random seed for the random number generator. Randomness is used for the random force in the Langevin thermostat as well as for random system initialization (orientation, velocities etc.). If no value is given, a seed is extracted from the current time and date. Note, that even if a seed was picked this way, the program will write the obtained seed in any continuation input files it is asked to generate. This will result in the continuation runs getting the same random number sequence as the initial run, if the parameter is not removed by hand. Also note that if you use a continuation file written at the start or in the middle of a run, you will not, in general, reproduce the same simulation as in the initial run, since the random number sequence starts from the beginning, not from the point at which the continuation file was written.

Verbosity

Format: `verbose K*/R* [I*]`
Example: `verbose 100.0`
Default: `verbose` *`tsimu`*`/20 2`

Options:		
	`none`	no basic output during run
	`interval` `[`*`level`*`]`	output interval

A switch for controlling the amount of basic output during a simulation. For no output, use `verbose none`. Otherwise, one can specify the time interval (in simulation time, fs for real units) between outputs. During an output event, the program prints out the system time, wall clock time, system temperature and system energy (divided into components). If the interval is given as a positive real, the data is printed both to standard output (screen) and the main output file — if negative, the data is only printed to the file.

Optionally, one may also set the verbosity level as an integer to control what data is written. If, for instance the level is set to 0, only the used parameters are written at the beginning of the main output file, whereas forces acting on the particles and nearest neighbor lists are also written for level 2.

XYZ-files

Format: `writexyz K* [R*]`
Example: `writexyz interval 200.0`
Default: `writexyz none`

Options:		
	`none`	no xyz file is written
	`start`	an xyz file is written at the start of the run
	`end`	an xyz file is written at the end of the run
	`startend`	an xyz file is written at the start and end of the run
	`interval` *`interval`*	an xyz file is written and appended every interval

A switch for controlling the amount of output in xyz-format. (In the output, the centers of MB molecules are labeled O, while the tips of HB vectors are labeled H, even though they do not represent H atoms.) If so specified, the program writes a file, with the name of the system and an extension `.xyz`, where the geometry of the system is given in xyz-format. If asked to do so periodically (`interval`), only one file is written and the geometries at different times are written one after another in the file. In this case, the interval for output must be given in simulation time — fs when real units are used.

Continuation files

Format: `writeinp K* [R*]`
Example: `writeinp none`
Default: `writeinp end`

Options:		
	`none`	no continuation file is written
	`start`	a cont. file is written at the start of the run
	`end`	a cont. file is written at the end of the run
	`startend`	a cont. file is written at the start and end of the run
	`interval` *`interval`*	a cont. file is written every interval

A switch for controlling when continuation files are written. The available options are the same as for `writexyz`. By default, a continuation file is written at the end of the run (distinguished by the extension `_end.mb`), but one can also ask for one at the beginning of the run (`start`, extension`_start.mb`) or periodically during the run (`interval`, extension `_mid`*`time`*`.mb`).

Units

Format: `units K*`
Example: `units reduced`
Default: `units real`

Options:	`real`	real units (K, eV, Å, fs)
	`reduced`	reduced units ($k_B = 1$)

A switch for different unit scales. If reduced units are used (`reduced`), the Boltzmann constant is set to 1 and all numeric values given by the user are taken as they are given. If real units are used, the Boltzmann constant is given the numeric value $8.617 \cdot 10^{-5}$ (its value in eV/K) and the masses given are multiplied by 103.6. This is because for everything else, the real unit scale uses Kelvins, electron volts, Ångströms and femtoseconds, but masses are expected in g/mol, which must be converted to eV fs^2 Å^{-2}.

4.2.2 `<mb-model>` block

Format:
```
K* R*
parameter value
```

Example:
```
<mb-model>
 elj 0.90
 ...
</mb-model>
```

Default: -

The block `<mb-model>` defines parameters of the MB model. It has similar format to `<control>`, i.e., each parameter is defined by its name followed by the value. The meaning of these parameters is given in section 2.
Names of parameters:

- `elj` – Lennard-Jones energy ($\varepsilon_{\mathrm{LJ}}$) [eV]
- `ehb` – H-bond energy ($\varepsilon_{\mathrm{HB}}$) [eV]
- `efi` – Dihedral angle energy (ε_{ϕ}) [1]
- `rhb` – H-bond equilibrium length (R_{HB}) [Å]
- `sigmalj` – Lennard-Jones sigma (σ_{LJ}) [Å]
- `sigmahb` – H-bond distance part sigma (σ_{HB}) [Å]
- `sigmath` – H-bond angle part sigma (σ_{θ}) [1]
- `expbond` – Exponent of bond order coefficient (v) [1]
- `rbond` – Bonding distance of bond order coefficient (R_b) [Å]
- `dbond` – Bonding cut width of bond order coefficient (D_b) [Å]
- `cutlj` – Cutoff for Lennard-Jones [Å]
- `cuthb` – Cutoff for H-bond [Å]
- `cutverl` – Cutoff increment (added to the potential cutoff) for list of neighbors [Å]
- `cutatom` – Cutoff for atom-atom and MB-atom interactions (optional) [Å]
- `mmb` – MB mass [g/mol]
- `imb` – MB moment of inertia [g/mol Å^2]
- `loh` – Length of "O-H arms" of MB molecules – relevant if there are interactions between atoms and the "H's" at the ends of the arms; see `<potentials>`. (Affects also the xyz-files written.) [Å]

4.2.3 `<cell>` block

Format:
```
R* [I*] [K* [R*]]
length [multiplier] [boundary [wall]]
```

Example:
```
<cell>
 10.00 periodic
 10.00 free
 12.34 periodic
</cell>
```

Default: `1.0 free`

The block `<cell>` defines the simulation supercell. This block should contain exactly 3 lines (ignoring empty lines and comments), specifying the cell in x, y and z directions. One line should have (i) the length of the cell [Å], (ii) an optional integer multiplier and (iii) an optional boundary type tag. If the multiplier is given and is greater than 1, the cell (and also the particles in it) will be copied the specified amount of times in the direction in question. The boundary conditions may be specified to be `periodic`, `free` (default) or `wall` with these keywords. The "wall" boundary means a harmonic force is applied to any particle that moves outside the supercell; the keyword `wall` must be followed by the spring constant of this force (in eV/Å^2 for real units). If the block is empty or missing, free boundaries are assumed (in which case the size of the cell is irrelevant).

4.2.4 `<elements>` block

Format:
```
K* R*
label mass
```

Example:
```
<elements>
 C 12.00
 atomB 123.45
</elements>
```

Default: `-`

The block `<elements>` defines the names and masses of particles other than MB molecules. Each non-empty non-comment line should specify the name and mass (in g/mol for real units) for one class of particles, separated by spaces. The length of the names is limited to 5 characters at the moment, so one can store a bit more information than just the chemical label. The keywords `MB` and `HB` are reserved though and cannot be used. The block is optional — if it is empty or missing, only MB molecules enter the simulation.

4.2.5 `<particles>` block

Format:
```
I* K* R* R* R* [R* R* R* R*]
index label x y z [qw qx qy qz]
```

Example:
```
<particles>
 1 MB 0.00 0.00 0.00
 2 MB 2.00 3.00 1.00 1.0 0.0 0.0 0.0
</particles>
```

Default: `-`

The block `<particles>` defines the particles entering the simulation. Each non-empty non-commented line should specify the type and initial position of a single particle. A line should contain (i) an integer index (ii) particle label (iii) x, y and z coordinates of the particle and (iv) optionally four components of a quaternion specifying the orientation of a MB molecule. The indices can be any positive integers, as long as they are unique. The purpose is to uniquely identify each particle, but the indices have no other meaning. Especially, the indices need not enter in order and one can leave some integers unused. (So removing a particle and an index from the list does not affect validity of the rest of the input, unless that particle also has a pre-defined velocity or a constraint which must also be removed. Extra velocities or constraints lead to aborting the

program, since these may be a sign of an error in the input.) Note however, that the program will reindex particles in output so that MB molecules are listed first and no gaps are left in the set of used index values. Particle labels should be either `MB`, if the line defines a MB molecule, or one of the names entered in `<elements>` block. If the particle is a MB molecule, one can also specify its orientation in quaternion representation ($[\cos\phi/2, x\sin\phi/2, y\sin\phi/2, z\sin\phi/2]$, ϕ being the angle or rotation and $[x, y, z]$ the unit vector defining the axis of rotation). The "no rotation" case, `1.0 0.0 0.0 0.0`, is defined to be an orientation where the H-bond unit vectors of the MB molecule are $[0, 0, 1]$, $[2\sqrt{2}/3, 0, -1/3]$, $[-\sqrt{2}/3, \sqrt{2}/\sqrt{3}, -1/3]$ and $[-\sqrt{2}/3, -\sqrt{2}/\sqrt{3}, -1/3]$, and other orientations are given with respect to this one. If the given quaternion does not have a unit norm, it is first scaled to unity. If no orientation is given, it is randomly generated. For an atomic particle, no orientation is expected anyway.

Alternatively, it is possible to read the geometry from an xyz file. The file should have structure similar to those written by CASHEW: Each MB molecule should be represented by one O and four H atoms, defined on five consecutive lines. The position of the O atom is read as the position of the MB and the vectors from the O to the first two H atoms are used for determining the orientation of the molecule. In order to read the geometry from a file, the block should contain only one line: `xyz` *`filename`* `[`*`config`*`]` Where *`filename`* is the name of the file and *`config`* is the number of the configuration to be read (in case the file contains several geometries, only one of them will be used and the user needs to tell which one). Note however, that in the current implementation file reading routines are slow since input is parsed as read, and trying to read a large file will take a very long time. It is never recommended to read xyz files containing several geometries.

4.2.6 `<velocities>` block

Format:
```
I* [to I*] R* R* R* [R* R* R*]
index [to index] vx vy vz [ox oy oz]
```

Example:
```
<velocities>
 1 to 3 0.00 0.00 0.00
 5 2.00 3.00 1.00 1.0 0.0 0.0
</velocities>
```

Default: *`random`*

The block `<velocities>` defines initial velocities of the particles and also initial angular velocities of the MB molecules. The block is optional: if it is not included, all particles will be assigned Maxwell-Boltzmann distributed random velocities. Naturally, one can also give the velocities of some particles and also then the rest will get random velocities. One line in the block should contain (i) the index or index range for particles, (ii) the x, y and z components of the velocity and (iii) optionally the x, y and z components of the angular velocity. The index is the unique index of a particle defined in `<particles>` block, and it specifies the particle for which the velocity is being defined. To define the velocities of several particles at once, one can specify a range of indices by giving the first and last index separated by the keyword `to`. For an atomic particle, no angular velocity is read. If one defines the velocity of a MB particle but no angular velocity is given, the angular velocity will be set to zero.

4.2.7 `<constraints>` block

Format:
```
I* [to I*] K* [K*/R*] K* [K*/R*] K* [K*/R*]
index [to index] x-constr [value]
          y-constr [value] z-constr [value]
```

Example:
```
<constraints>
 1 to 3 frozen all frozen all free
 5 frozen pos well 1.0 well 1.0
</constraints>
```

Default: `free`

Options:

`frozen all`	completely frozen
`frozen pos`	frozen position
`frozen vel`	constant velocity
`force` *force*	external force
`well` *spring*	harmonic well
`free`	no constraints

The block `<constraints>` defines possible constraints to be applied to the motion of the particles. It is naturally optional. A line in the block should contain (i) index or index range of particles and (ii) the constraints to be applied in x, y and z directions. The index works as in the block `velocities`, and `to` can be used to specify a range. A constraint must be defined by one of the following keywords: `free`, `frozen`, `well` or `force`. If `free`, no constraints are applied to the coordinate. Using `free` one can leave some degrees of freedom unchanged while restricting the movement in the direction of other axes. The keyword `frozen` must be followed by `pos`, `vel` or `all`. A `frozen pos` (frozen position) means the coordinate is kept fixed at all times, but a MB molecule is still allowed to rotate. Specifying `frozen all` will also freeze the rotational degree of freedom. So, if one applies `frozen all` to, say, the x coordinate, this will always keep the x component of the angular velocity vector zero. The option `frozen vel` will freeze the velocity (i.e., the component of acceleration will be kept zero). This will allow the simulation of dragging etc. The tag `well` stands for harmonic well, and it will apply an extra harmonic potential $U = \frac{1}{2}k(x - x_0)^2$ to the particle. It must be followed by the value of the spring constant of the potential, k (eV/Å^2, in real units). Finally, `force` adds a constant external force to the particle. It must be followed by the strength of the force (in eV/Å).

4.2.8 `<potentials>` block

Format:
```
K* K* [R*]
 K* R* R* [R*]
label1 label2 [cut]
 potential param1 param2 [param3]
```

Example:
```
<potentials>
 MB atom 10.0
 LJ 0.10 2.50
 pow 0.20 4.50
</potentials>
```

Default: -

Options:

`LJ` *eps sigma*	Lennard-Jones	$4\varepsilon[(\sigma/r)^{12} - (\sigma/r)^6]$
`exp` *eps rho*	exponent	$\varepsilon \exp(-r/\rho)$
`pow` *eps nu*	power	$\varepsilon r^{-\nu}$
`spr` *k r0*	spring	$\frac{1}{2}k(r - r_0)^2$
`fene` *k R0 r0*	FENE	$-\frac{1}{2}kR_0^2 \ln[1 - ((r - r_0)/R_0)^2]$
`hard` *eps sigma r0*	hardsphere LJ	$4\varepsilon[(\sigma/(r - r_0))^{12} - (\sigma/(r - r_0))^6]$
`hardrep` *eps sigma r0*	repulsive HS	$4\varepsilon[(\sigma/(r - r_0))^{12} - (\sigma/(r - r_0))^6] + \varepsilon$, $r < r_0 + 2^{1/6}\sigma$

The block `<potentials>` defines interactions for atomic particles. The definition of a potential begins with a line containing the labels of the two types of atoms for which the potential acts (as defined in block `<elements>`) and optionally the cutoff range of the potential. (If no cutoff is given, the `cutatom` value from `<mb-model>` is used — one or the other must be given for all types of potentials.) In addition to atomic labels, *two* other labels can be used: `MB` denotes the center of a MB molecule (just like in the `<positions>` block) and `HB` denotes the tips of the hydrogen bond "arms" of the molecules. That is, a potential of, say, type `HB Na` will act between atoms called Na and the points in space separated from the MB centers of mass by the MB H-bond vectors. The length of these vectors is defined in the block `<mb-model>`, using tag `loh` (l_{OH}). MB molecules cannot have extra interactions between them, so using `MB MB` or `MB HB` is forbidden.

Having defined a pair of particle types, the following lines define the potentials acting between the particles, each line defining a single potential term. One line should contain (i) a keyword specifying the type of potential and (ii) the parameters of the potential. Possible potentials are Lennard-Jones (`LJ`), exponential (`exp`), spring (`spr`), power-law (`pow`), hardsphere+LJ (`hard`), repulsive hardsphere (`hardrep`), and FENE (`fene`). The potential terms are simply summed together until a new pair of atomic labels or the end of the block is found.

4.2.9 `<statistics>` block

Format: `K* [R*/I* [to I*]]`
parameter [*value* [to *value*]]

Example:

```
<statistics>
writestat
 start 0.0
 interval 50.0
 time
 temperature
 energy
 forcesum 1 to 3
writerdf average 100.0 500.0 all 10.0
</statistics>
```

Default: -

The block `<statistics>` defines the physical parameters whose values are to be recorded. There are two types of monitors: functions (e.g. the radial distribution function) and values (e.g. energy, temperature). Functions have their own switches defining how and when they are measured, and they are written in files of their own. A group of values to be recorded must be defined first by the keyword `writestat` followed by the names of the values to be monitored. If only one `writestat` command is given, the values are written in a file with the extension `_stats.out` in the order in which they appear in the block. If several `writestat` commands are given, several files will be created, distinguished by the extensions `_stats_001.out` etc., each containing the group of values following the respective `writestat` command in the block. By default, these values are recorded as often as defined by the tag `verbose` in `<control>` block. The starting point and interval can be adjusted though, using tags `start` and `interval`. When several files are written, it is possible to set different recording intervals for them. One can also record the same value several times. That is, one can record, say, values A, B and C in one file every 10 fs, and values A and D in another file every 1000 fs. The values measured at the same time are written in the file on a single line, and the values at different times are separated to different lines.
Names of functions and values:

- `writebnd` – Switch for controlling when bonding analysis is written to file
- `writerdf` – Switch for controlling when a radial distribution function (RDF) is written to file
- `writeadf` – Switch for controlling when an axial distribution function (ADF) is written to file
- `writestat` – Switch for controlling what values are written to file or files

- `start` – Switch for controlling when statistics measuring starts. Must be followed by the virtual time [fs] after which the recording of values begins. By default, the recording starts at the beginning of the simulation.
- `interval` – Switch for controlling how often statistics are measured. Must be followed by the virtual time interval [fs] for the value recording. By default, the interval defined by the tag `verbose` in the block `<control>` is used.
- `time` – virtual time [fs]
- `wallclock` – actual simulation time in "d hh:mm:ss.xxx" format
- `temperature` – temperature [K]
- `pressure` – pressure [eV/Å^3]
- `energy` – total energy [eV]
- `kinetic` – kinetic energy [eV]
- `potential` – potential energy [eV]
- `linear` – linear movement contribution of kinetic energy [eV]
- `rotational` – rotational contribution of kinetic energy [eV]
- `lenjon` – Lennard-Jones contribution of potential energy [eV]
- `hbond` – hydrogen bond contribution of potential energy [eV]
- `mbatom` – MB-atom contribution of potential energy [eV]
- `atomatom` – atom-atom contribution of potential energy [eV]
- `virial` – virial [eV]
- `position` – particle coordinates: must be followed by a particle index or range (defined using `to`)
- `orientation` – MB molecule orientation in quaternion representation: must be followed by a particle index or range (defined using `to`)
- `velocity` – particle velocity: must be followed by a particle index or range (defined using `to`)
- `angular` – MB molecule angular velocity: must be followed by a particle index or range (defined using `to`)
- `arms` – MB molecule "arm" vectors: must be followed by a particle index or range (defined using `to`)
- `force` – force acting on a particle: must be followed by a particle index or range (defined using `to`)
- `torque` – torque acting on a particle: must be followed by a particle index or range (defined using `to`)
- `forcesum` – sum of forces acting on a group of particle: must be followed by a list of particle indices or ranges (separated by spaces). Example: `forcesum 1 3 to 6 10` will sum the forces on particles 1, 3, 4, 5, 6, and 10.

Bonding analysis

Format:	`writebnd K* [R* [R*]] [K* [R*]]`
	`writerdf` *timing*[1] `[`*interval*`]`
Example:	`writebnd interval 100.0`
Default:	`writebnd none`

[1]Options:

`none`	no info is written
`start`	info is written at the start of the run
`end`	info is written at the end of the run
`startend`	info is written at the start and end
`interval` *interval*	info is written every interval

A switch for controlling H-bonding analysis calculation and output. The possible choices for the output interval are the same as for `xyzwrite`, the default being `none`. When the bonding analysis is called, the program writes two files with the extensions `_mb.xyz` and `_bonds.out`. The first one is an xyz geometry file with just the centers of MB molecules included (for visualization purposes). The second file contains the actual bonding data. Each row in the file lists the bond counts for all the MB molecules (in the order they appear in the xyz file) at a point in time, and the rows appear in chronological order.

The bond count for a molecule is defined using the Gaussian part of the MB potential (4): for MB i, the number of H-bonds is

$$n_i = \sum_j \sum_{\substack{k=1 \\ l=1}}^{4} G_{ij}^{kl}. \tag{44}$$

For a molecule with ideal coordination, this is 4, and for a molecule with no H-bonding, 0.

Radial distribution function

Format:	`writerdf K* [R* [R*]] [K* [R*]]`
	`writerdf` *timing*[1] `[`*interval* `[`*start*`]] [`*type*[2] `[`*range*`]]`
Example:	`writerdf average 100.0 500.0 all 10.0`
Default:	`writerdf none`

[1]Options:

`none`	no RDF is written
`start`	an RDF is written at the start of the run
`end`	an RDF is written at the end of the run
`startend`	an RDF is written at the start and end
`interval` *interval*	an RDF is written every interval
`average` *interval* `[`*start*`]`	an average RDF is measured and written

[2]Options:

`all`	RDFs are measured for MB-MB, atom-atom, MB-atom, and any-any
`mb`	RDF is measured only for MB-MB
`atom`	RDF is measured only for atom-atom

A switch for controlling radial distribution function calculation and output. The possible choices for the output interval are the same as for `xyzwrite`, the default being `none`, with the addition of the option `average` which must be followed by the measuring interval and, optionally, the starting time for the averaging [fs]. (If no start time is given, the averaging starts from the beginning of the run.) This option will start measuring the RDF at the specified starting time and then as often as the specified interval. In the end of the run the results will be averaged and the average and standard error of the average are written. Optionally, one can also specify the particle species and the plot range for the function, after the interval. The options for the types are `mb`, `atom` and `all`. The first two tags result in the RDF being computed only for MB molecules or atomic particles, respectively. For the last keyword, both of these are recorded along with the MB-atom RDF and total RDF. The RDF will be written in column format, with columns containing (i) separation, (ii) MB-MB RDF, (iii) atom-atom RDF, (iv) MB-atom RDF, (v) total RDF. Consecutive RDFs are written in the same file, one after another.

Axial distribution function

Format: `writeadf K* [R* [R*]] K* [K*] K* I*/(R* R* R*) [K* [R*]]`
`writeadf` *timing*[1] `[`*interval* `[`*start*`]] `*axis*[2] `[`*angle*[3]`]`
axispos[4] *index/(x y z)* `[`*type*[5] `[`*range*`]]`
Example: `writeadf average 100.0 500.0 zaxis noangle follow 1 all 10.0`
Default: `writeadf none`

[1]Options:

`none`	no ADF is written
`start`	an ADF is written at the start of the run
`end`	an ADF is written at the end of the run
`startend`	an ADF is written at the start and end
`interval` *interval*	an ADF is written every interval
`average` *interval* `[`*start*`]`	an average ADF is measured and written

[2]Options:

`xaxis`	ADF is measured in the yz-plane
`yaxis`	ADF is measured in the xz-plane
`zaxis`	ADF is measured in the xy-plane

[3]Options:

`angle`	angle dependent ADF, $g(r, \varphi)$
`noangle`	angle averaged ADF, $g(r)$

[4]Options:

`follow` *index*	measuring axis follows the given particle
`fixed` *x y z*	measuring axis fixed at a position

[5]Options:

`all`	ADFs are measured for MB, atom, and any
`mb`	ADF is measured only for MBs
`atom`	ADF is measured only for atoms

A switch for controlling axial distribution function calculation and output. The possible choices for the interval are the same as for `rdfwrite`, the default being `none`. The ADF is the distribution of particle positions measured from a given axis, which must point in the direction of one of the supercell axes.

After the interval, one must specify this axis around which the measuring is done. The direction is specified by one of the keywords `xaxis`, `yaxis` or `zaxis`, for x-, y- and z-axis, respectively. After this, one can optionally specify whether the function is averaged over the axial angle (ADF = $g(r)$) or whether a 2D function of both the distance and angle should be measured (ADF=$g(r, \varphi)$). This can be done with the keywords `angle` and `noangle`, respectively. By default, the angle is averaged out. (If the angle is not averaged, the angle is measured from the x-axis if the main axis is the z-axis, and from the z-axis otherwise. Note that in this case the function tracks the positions of individual particles, and you should take an average over a very long simulation to get decent statistics.) This should be followed by the definition of a point through which the axis should go. This point can be fixed, in which case one must give the keyword `fixed` followed by the coordinates of the point. It can also be set to follow a particle by the keyword `follow` and the index of the particle. Finally, one can specify the types of particles to be measured, similarly to RDF.

4.3 Input example

Below is an example input file for a system with 360 water molecules in an ice lattice with 45 (generic) interstitial atoms. Red text is commented and would be ignored by CASHEW.

```
!
! quite small ice crystal with foreign interstitials
!
<control>
 temp              273.15
 tsimu            1000.00
 tstep               1.00
 writexyz  interval  5.00
 !writeinp  interval 50.0
 verbose           -10.00
 algo              verlet
 thermo     langevin 0.01
 run                 test
 ! comments can be added
</control>
<mb-model>
 loh      0.9      ! [Å]
 ehb     -0.181    ! [eV]
 elj      0.00905  ! [eV]
 efi      0.01     !
 rhb      2.78     ! [Å]
 sigmalj  2.58     ! [Å]
 sigmahb  0.278    ! [Å]
 sigmath  0.08     !
 expbond  0.5      !
 rbond    3.61     ! [Å]
 dbond    0.556    ! [Å]
 cuthb    6.0      ! [Å]
 cutlj    7.0      ! [Å]
 cutverl  2.0      ! [Å]
 mmb      18.00    ! [g/mol]
 imb      1.333    ! [g/mol Å^2]
</mb-model>
<cell>
 7.8631 3 periodic
 4.5397 5 periodic
 7.4133 3 periodic
</cell>
<particles>
1   MB  0.0000  0.0000  0.0000   0.0 0.0 1.0 0.0
2   MB  3.9315  2.2699  0.0000   0.0 0.0 1.0 0.0
3   MB  1.3105  2.2699  0.9267   1.0 0.0 0.0 0.0
4   MB  5.2420  0.0000  0.9267   1.0 0.0 0.0 0.0
5   MB  1.3105  2.2699  3.7067   0.0 1.0 0.0 0.0
9   MB  5.2420  0.0000  3.7067   0.0 1.0 0.0 0.0
7   MB  0.0000  0.0000  4.6333   0.0 0.0 0.0 1.0
8   MB  3.9315  2.2699  4.6333   0.0 0.0 0.0 1.0
10  atom  2.0000  5.0000  2.0000
</particles>
<elements>
    atom   1.0
</elements>
<potentials>
 atom  MB  7.50
     LJ  0.01  3.00
</potentials>
<constraints>
!2  free         free         force -0.5
!2  frozen pos   frozen pos   frozen pos
 2  well  0.1    well  0.1    well  0.1
</constraints>
<velocities>
2 to 4  0.0 0.0 0.5   0.0 0.1 0.0
</velocities>
<statistics>
 writestat
  interval   100.0
  time
  temperature
 writestat
  interval   10.0
  time
  wallclock
 writerdf  interval 100.0 all 8.0
</statistics>
```

temperature in K

simulation time in fs

timestep in fs

will append the `system.xyz` file after every 5 fs of simulation

if uncommented, will write an input file for continuation runs every 50 fs - now (as default) writes one only at the end

will write system state info only to file every 10 fs

velocity verlet

langevin thermostat with friction coefficient 0.01 1/fs

test run, use `full` or remove for a full simulation

parameters for the water model

periodic cell in x direction with a length of 3 x 7.8631 Å

similarly for y and z - use free for free boundaries or wall for harmonic walls

Each line defines one particle: index, type, coordinates, [orientation]

Orientation [0 0 1 0] means the MB molecu is rotated 180 degrees around the y axis.

Here, we have 8 MB molecules and one atomic particle - however, since the cell will be multiplied 3 x 5 x 3 times, the total number of particles will be 360 + 45.

one type of atomic particles defined, name is atom and mass is 1.0 g/mol

Atoms interact with MB's via Lennard-Jones potential up to 7.50 Å distance.

MB molecule "2" is trapped in a harmonic well, as are the image particles created when multiplying the cell.

MB molecules "2", "3" and "4" have the initial velocity [0 0 0.5] and angular velocity [0 0.1 0]

writes the virtual time and system temperature every 100 fs to `system_stats_001.out`

writes the virtual time and real simulation time (wall clock) every 10 fs to `system_stats_002.out`

write the radial distribution function for all particles, up to a distance of 8 Å, every 100 fs

5 Version history

5.1 Version 0.32

Updates from version 0.31:

- various bug fixes

5.2 Version 0.31

Updates from version 0.3:

- added preprocessing options to the previously parallel only code to enable compiling in serial mode: there is no separate serial code anymore
- the names of the source files were changed upon combining the parallel and serial versions
- added separate subcell division for finding pair of MB molecules and pairs involving atoms

5.3 Version 0.3

Updates from version 0.22:

- added exact torques (previous versions calculated MB-MB torques using an approximate scheme)
- added possibility to read geometry from an xyz file
- added bonding analysis and bond count output
- added possibility to write several `stats.out` files with different output intervals
- minor bug fixes

5.4 Version 0.22 p (parallel cashew)

A parallel version of the simulator. MPI-parallelization has been implemented for:

- force calculation, which consumes roughly 90% of the computational effort and is the most important part to optimize
- energy calculation, which consumes considerable cpu time if statistics are written often
- bond number calculation, which is quite fast but must be done at each step and may become comparable to force calculation if only the forces are calculated in parallel
- neighbor list determination, which is quite time consuming but usually done seldomly; however, it is not as straightforward to parallelize efficiently as the routines above

One should also parallelize e.g. RDF calculation, but that remains to be implemented. As of now, analysis and output routines may become an efficiency bottleneck if the program is set to output a lot of data. However, if there is only moderate output during run, the scaling is excellent, as shown in Figure 2. Position and velocity updating have not been parallelized, but these routines are very fast and a parallel updating scheme would require massive information transfer between the processors to set up the force calculation, likely making the parallel scheme inefficient.

By far the most important routine to parallelize is the force calculation, and in this implementation the task is split according to particles. The program calculates the force terms incorporating particle pairs (i, j), and the task is split according to one of the particle indices, i. That is, one processor calculates forces due to pairs (1,2), (1,3), (1,4), ..., (2,3), (2,4), (2,5), ..., another one calculates (i,$i+1$), (i,$i+2$), (i,$i+3$), ... etc. To prevent double counting (and double effort), only pairs $i < j$ are considered when analyzing a given i. Therefore, the indices are not split evenly, but the processors handling the smaller i indices are also given shorter index ranges to analyze, since the j index will be given more values than for large i. In addition, active load balancing shifts the load between the processors, aiming at minimizing the idle time of all processors due to having to wait for the slowest one to finish.

Known issues: The order of summation of force terms in force calculation affects the final decimals of the outcome due to numeric errors in floating point number operations. This means that if one reruns a simulation with the parallel version, particle trajectories will start to deviate in a thousand time steps. This should not affect the physics!

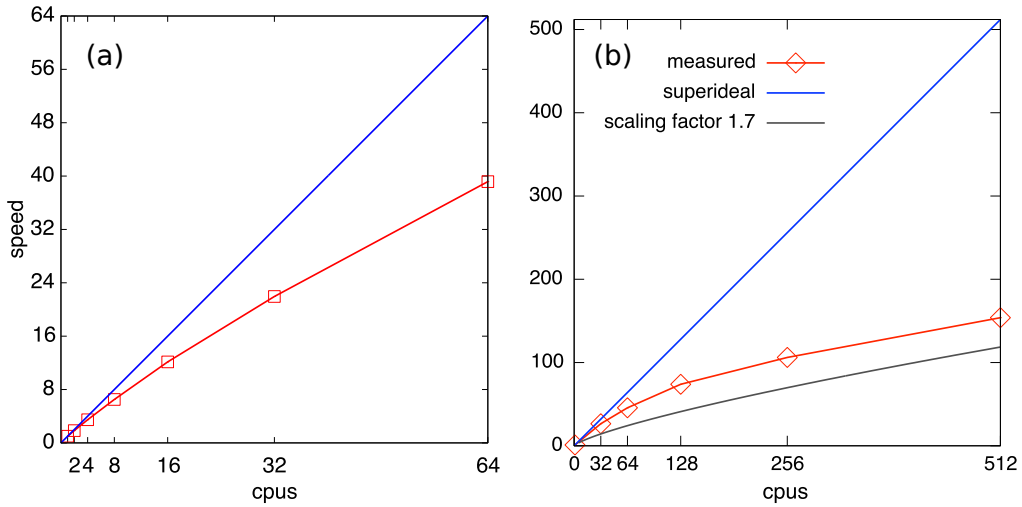

Figure 2: Parallel scaling for a system of (a) 9720 and (b) 34200 MB molecules ran for 1000 and 500 time steps, respectively, with little output (red line) vs. the superideal case (blue line). For systems of this size and range, the scaling is adequate: doubling the number of cpus reduces the wall clock time by a factor of more than 1.7 (gray line). In this case, one can use about $n_{\text{particles}}/100$ cpus. (The number of processors can be any sufficiently small positive integer, not necessarily a power of 2.)

5.5 Version 0.22

Updates and fixes from version 0.21:

- moved the `writerdf` (RDF) command from the block `<control>` to `<statistics>`
- added axial distribution function calculation
- added force sum stat type
- made it so that the lists of neighbors for MB molecules and atoms are created according to their respective interaction cutoffs instead of a global maximum cutoff
- added a hard-sphere + LJ type pair potential
- fixed a memory leak
- added a FENE type pair potential
- added a second verbosity switch
- fixed handling of tab characters in the input file (previously tabs were not recognized, now they are treated as normal spaces)
- fixed a bug in the subcell division of the supercell (for neighbor finding) where particles outside the predefined simulation volume in open boundary systems were treated wrong
- fixed other small bugs

5.6 Version 0.21

Updates and fixes from version 0.2:

- improved efficiency of the force calculation routine (may still be further optimized)
- changed the neighbor-finding routine to an $\mathcal{O}(n)$ algorithm (by decomposing the simulation cell to subcells) — replaced the brute-force $\mathcal{O}(n^2)$ algorithm
- changed the RDF calculation routine to an $\mathcal{O}(n)$ algorithm for short plotting ranges (by using the neighbor lists)
- added a switch for changing between real [eV, Å, K, fs] and reduced [$k_B = 1$] units

- fixed issue with Langevin thermostat where friction force at $t + \Delta t$ in velocity Verlet was evaluated using $v(t + \Delta t/2)$
- added a spring type pair potential

5.7 Version 0.2

Updates and fixes from version 0.1:

- added conjugate gradient minimum energy search
- added radial distribution function calculation
- added pressure monitoring
- added Berendsen barostat
- added flexible statistics output
- fixed bug that forced neighbor list update after each step
- fixed bug that updated bond order terms only with neighbor list update (together with the bug listed above, the simulator worked correctly but was inefficient)
- fixed issue with cooler thermostat where friction force at $t+\Delta t$ in velocity Verlet was evaluated using $v(t + \Delta t/2)$ (Langevin still open)
- the coefficient for changing units of masses had an inverse value (0.01 instead of 100!) which screwed the timescale, fixed this
- improved efficiency of the force calculation routine (still needs optimization)
- fixed other small bugs

5.8 Version 0.1

The first version of the program, containing the basic physics of the MB model and the MD algorithm. The options provided include

- velocity Verlet MD
- Langevin thermostat
- handling for input files in a flexible format.