SOP-GPU Documentation

Release 2.0

Artem Zhmurov **Andrey Alekseenko**
Olga Kononova **Valeri Barsegov**

September 12, 2016

Contents

The SOP-GPU package, where SOP stands for the Self Orginized Polymer Model fully implemented on a GPU, is a scientific software package designed to perform Langevin Dynamics Simulations of the mechanical or thermal unfolding, and mechanical pulling/indentation of large biomolecular systems in the experimental subsecond (millisecond-to-second) timescale. The SOP-GPU package utilizes the C_α and C_α-C_β based coarse-grained description of proteins combined with computational power of modern Graphics Processing Units (GPUs).

Download SOP-GPU package; View code in repository

Chapter 1

Self-Organized Polymer model

1.1 Method

In the structure-based Self-Organized Polymer (SOP) model, each amino acid residue is usually represented by a single interaction center described by the corresponding C_α-atom, or two interaction centers described by the corresponding C_α and C_β atoms. In the first case the protein backbone is represented by a collection of the $C_\alpha - C_\alpha$ covalent bonds. In the second case, backbone atoms are replaced by C_α bead and side-chain atoms are replaced by one C_β bead, connected covalently to the C_α bead of the same amino acid. The coarse-graining procedure with one interaction center representating each residue is illustrated on Figure 1. The potential energy function of a protein conformation U_{SOP} is specified in terms of the coordinates of the C_α and C_β-beads $\{r_i\} = r_1, r_2, \ldots, r_N$, where N is the total number of beads in coarse-grained model. U_{SOP} is given by *[Hyeon2006]*, *[Mickler2007]*:

$$U_{SOP} = U_{FENE} + U_{NB}^{ATT} + U_{NB}^{REP}. \tag{1.1}$$

Eq. (1.1), the first term is the finite extensible nonlinear elastic (FENE) potential:

$$U_{FENE} = -\sum_{covalent} \frac{kR_0}{2} \log\left(1 - \frac{(r_{ij} - r_{ij}^0)^2}{R_0^2}\right). \tag{1.2}$$

Here, $k = 14$ N/m is the spring constant, and $R_0 = 2.0$ Å is the tolerance to the change of the covalent bond length. The FENE potential describes the backbone chain connectivity ($C_\alpha - C_\alpha$), side-chain connectivity ($C_\alpha - C_\beta$) and other covalent links (e.g. disulfide bonds). The distance between the particles ($C_\alpha - C_\alpha$ or $C_\alpha - C_\beta$) i and j, is r_{ij}, and r_{ij}^0 is its value in the native structure. The summation (Σ_{bonds}) is performed over all pairs of beads i and j that are covalently bonded.

To account for the non-covalent (non-bonded) interactions that stabilize the native state, the Lennard-Jones potential is used:

$$U_{NB}^{ATT} = \sum_{native} \varepsilon_h \left[\left(\frac{r_{ij}^0}{r_{ij}}\right)^{12} - 2\left(\frac{r_{ij}^0}{r_{ij}}\right)^{6}\right]. \tag{1.3}$$

Here, the r_{ij} is the distance between two beads i and j and r_{ij}^0 is the equilibrium distance taken from the initial structure. The value of ε_h quantifies the strength of the non-bonded interactions. The summation $\sum_{native}$ goes over all beads i and j that are assumed forming native contacts. The definition of the native contact can vary, the most general definition follows. If the two be beads i and j are separated by more than 2 covalent bonds and $r_{ij} < R_C$ then they form native contact. The typical cut-off distances are $R_C = 8.0$ Å for $C_\alpha - C_\alpha$ and $C_\alpha - C_\beta$ contacts and $R_C = 5.2$ Å for $C_\beta - C_\beta$ bonds. The value of ε_h quantifies the strength of the non-bonded interactions and sets the energy scale. This parameter can be estimated based on the results of all-atom MD simulations.

The non-native (non-bonded) interactions are treated as repulsive:

$$U_{NB}^{REP} = \sum_{repulsive} \varepsilon_l \left(\frac{\sigma_l}{r_{ij}} \right)^6 \tag{1.4}$$

In Eq. (1.4), the values for the parameters are $\varepsilon_l = 1$ kcal/mol and $\sigma_l = 3.8$ Å. These define the strength and the range of the repulsion.

The dynamics of the system is obtained by solving numerically the Langevin equations of motion for each particle position r_i in the over-damped limit:

$$\xi \frac{dr_i}{dt} = -\frac{\partial U_i(r_i)}{\partial r_i} + g_i(t) \tag{1.5}$$

In Eq. (1.5), $U_i(r_i)$ is the total potential energy, which accounts for all the biomolecular interactions between the particles in the molecule (U_{SOP}; see Eq. (1.1)). It also includes interactions of particles with the indenting object (U_{tip}; see Eq. (3.7)) and surface particles in indentation regime (see Section *Force indentation simulations* below) and external force $f\Delta X$ in pulling regime. Also, in Eq. (1.5) $G_i(t)$ is the Gaussian distributed zero-average random force, and ξ is the friction coefficient *[Zhmurov2010] [Kononova2013a]* .

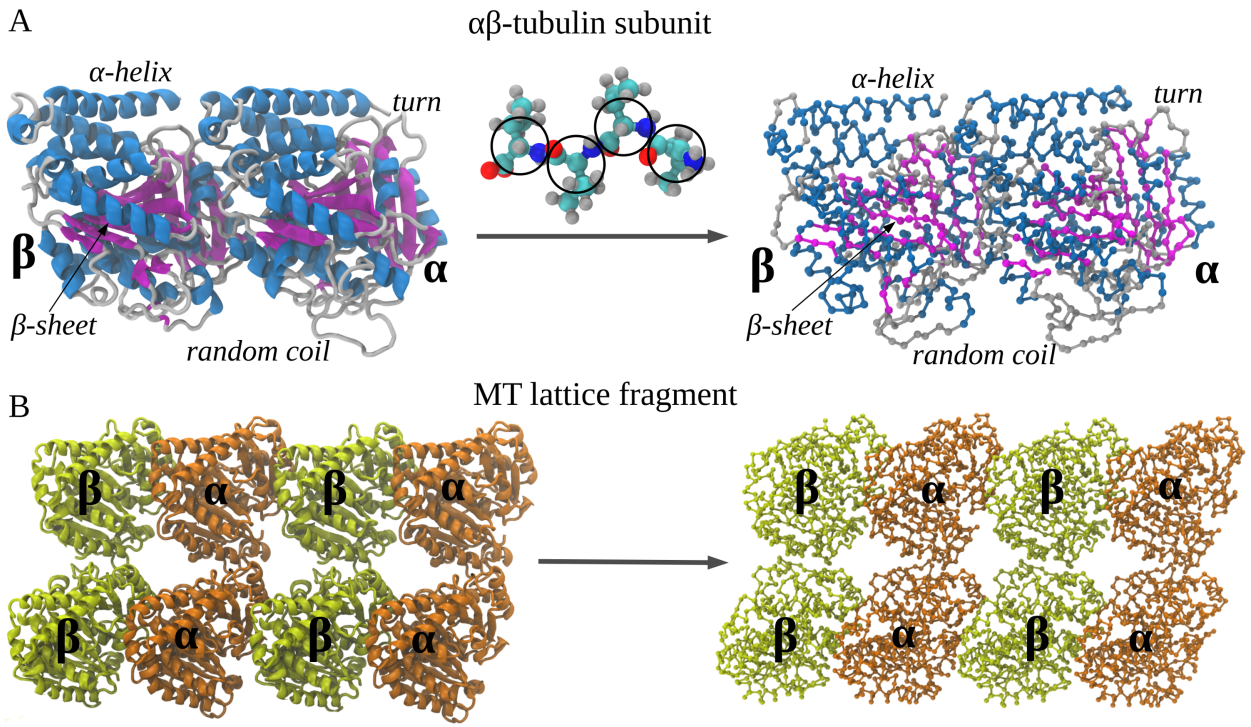

Figure 1: Coarse-graining procedure for constructing a Self Organized Polymer (SOP) model of a polypeptide chain. Panel **A** exemplifies coarse-graining of the atomic structure of the $\alpha\beta$-tubulin dimer – the structural unit of the microtubule cylinder. The amino acid residues are replaced by single interaction centers (spherical beads) with the coordinates of the C_α-atoms (represented by the black circles). Four representative circles are shown to exemplify the coarse-graining process. Consequently, the protein backbone is replaced by a collection of the $C_\alpha - C_\alpha$ covalent bonds with the bond distance of 3.8 Å. Panel **B** depicts the results of coarse-graining of a small fragment of microtubule cylinder. Four identical copies of the tubulin dimer structure, coarse-grained as described in panel **A**, form a C_α-based model of the fragment.

1.2 Benchmark simulations

We have tested the performance of the SOP-GPU package (written in CUDA - a dialect of C and C++ programming languages) on a NVIDIA GPU Tesla C1060 (MIPT), and have compared the results against the performance of the optimized C code (SOP program) on a dual Quad Core Xeon 2.83 GHz of a similar level of technology. We have analyzed the results of CPU- and GPU-based computations by comparing the force spectra, i.e. f versus X force-extension profiles, the distributions of unfolding forces (peak forces in the force spectra), and the average temperature $\langle T \rangle$, for the all-β sheet WW-domain. Aside from small deviations due to the different initial conditions, the profiles of $f(X)$ and $\langle T \rangle$, and the unfolding force histograms, obtained on the CPU and on the GPU, agree very well (Figure 2).

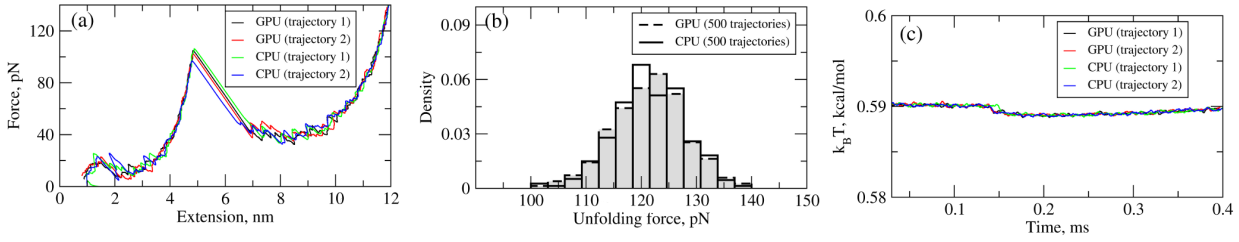

Figure 2: Comparison of the results of pulling simulations for the WW-domain obtained on a GPU and on a CPU (pulling speed $\nu_f = 2.5 \mu m/s$). Panel (a): Representative examples of the force spectrum (force-extension curves). Panel (b): The histograms of unfolding forces. Panel (c): The average temperature of the system as a function of time $\langle T(t) \rangle$.

We have compared the overall performance of an end-to-end application of the SOP-GPU program with the heavily tuned CPU-based implementation (SOP program) in describing the Langevin dynamics of the WW domain at equilibrium. We profiled the computational performance of the SOP-GPU program as a function of the number of independent trajectories s running concurrently on the GPU (**many-runs-per-GPU approach**). While the single CPU core generates one trajectory at a time, the GPU device is capable of running many trajectories at the same time. The results (see Figure 3a) show that, for the WW domain ($N = 34$), the GPU accelerates computations starting from 3 independent runs, which is equivalent to a single run for a system of $N \approx 10^2$ residues (**one-run-per-GPU approach**). This is the so-called break-even point. While the simulation time on the CPU scales linearly with s (or with N), the scaling on the GPU in this regime is sublinear (nearly constant) up to $N \approx 10^4$ ($s \approx 300$ for the WW domain). At this point, the GPU shows significant performance gains relative to the CPU reaching the maximum 80-90-fold speedup (see Figure 3b). The amount of GPU on-board memory, i.e. ~4 GB (Tesla C1060), is sufficient to describe long Langevin dynamics for large biomolecular systems of $\sim 10^4$ residues.

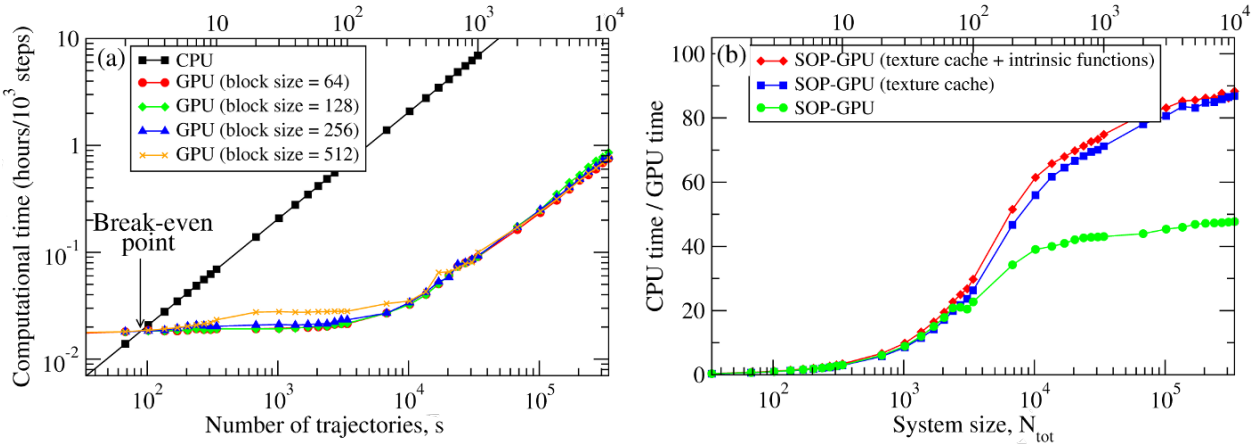

Figure 3: Panel (a): The log-log plot of the computational time per 1,000 steps of the simulations on a CPU and on a GPU versus the system size, N (**one-run-per-GPU approach**), and versus the number of independent trajectories running concurrently on a GPU s (**many-runs-per-GPU approach**), for the all-β-strand WW domain. The GPU performance is tested for the thread blocks of size B = 64, 128, 256, and 512. Panel (b): The log-linear plot of the relative CPU/GPU performance (computational speedup) as a function of N and s. The performance is compared for the SOP-GPU program, and when it is accelerated by using texture cache, and texture cache plus intrinsic mathematical functions.

Chapter 2

Generation of pseudo-random numbers on graphics processors

Pseudo-random number generators are used in many computer applications such as simulations of stochastic systems, numerical analysis, probabilistic algorithms, etc. Numerical modeling of biological systems and processes, e.g., all-atom MD simulations in implicit solvent *[Brooks1983]*, *[Haberthur2008]*, Langevin simulations *[Zhmurov2010b]*, and Monte Carlo simulations *[Press1992]*, all require generation of a large number of independent random variables at each step of a simulation run. We developed two approaches for implementation of random number generators (RNGs) on a graphics processing unit (GPU). In the **one-RNG-per-thread approach**, one RNG produces a stream of random numbers in each thread of execution, whereas the **one-RNG-for-all-threads** method builds on the ability of different threads to communicate, thus, sharing random seeds across an entire GPU device. An RNG produces a sequence of random numbers, u_i, which is supposed to imitate independent and uniformly distributed random variates from the unit interval $(0, 1)$. There are three main requirements for a numerical implementation of an RNG: (1) good statistical properties, (2) high computational speed, and (3) low memory usage. Because a deterministic sequence of random numbers comes eventually to a starting point, $u_{n+p} = u_n$, an RNG should also have a long period p *[LEcuyer2007]*. In addition, an RNG must pass rigorous statistical tests of randomness (i.e., for independence and for uniformity), and some application-based tests of randomness that offer exact solutions to the test applications *[LEcuyer2007]*, *[Marsaglia1996]*, *[Mascagni2000]*, [Soto1999]. Indeed, using random numbers of poor statistical quality might result in insufficient sampling, unphysical correlations, and even unrealistic results, which might lead to errors in practical applications. We developed the GPU-based realizations of several RNGs, which provide pseudo-random numbers of high statistical quality, using the cycle division paradigm *[Zhmurov2011b]*.

2.1 Method

Different methods are used to generate the Gaussian distributed random variates g_i from the uniformly distributed random numbers $u_i, (i = 1, 2, ..., n)$ *[Tsang2000]*, *[Marsaglia1964]*, *[Box1958]*. Here, we adopt the most commonly used Box-Mueller transformation *[Box1958]*. In the one-RNG-per-thread approach, the basic idea is to partition a single sequence of random numbers among many computational threads running concurrently across an entire GPU device, each producing a stream of random numbers. Since most RNG algorithms, including LCG, Ran2, and Hybrid Taus, are based on sequential transformations of the current state [Press1994], then the most common way of partitioning the sequence is to provide each thread with different seeds while also separating the threads along the sequence so as to avoid possible inter-stream correlations (see Figure 4, left panel). On the other hand, several generators, including the Mersenne Twister and Lagged Fibonacci algorithms, which employ recursive transformations, allow one to leap ahead in a sequence of random variates and to produce the $(n + 1)$-st random number without knowing the previous, n-th number *[Mascagni2004]*. The leap size, which, in general, depends on a choice of parameters for an RNG, can be properly adjusted to the number of threads (number of particles N), or multiples of N $(M \times N)$. Then,

all N random numbers can be obtained simultaneously, i.e. the j-th thread produces numbers $j, j+N, j+2N, ...$, etc. $(j = 1, 2, ..., n)$. At the end of each simulation step, threads of execition must be syncronized to update the current RNG state. Hence, the same RNG state can be shared by all threads, each updating just one elements of the state. We refer to this as the one-RNG-for-all-threads approach (Figure 4, right panel).

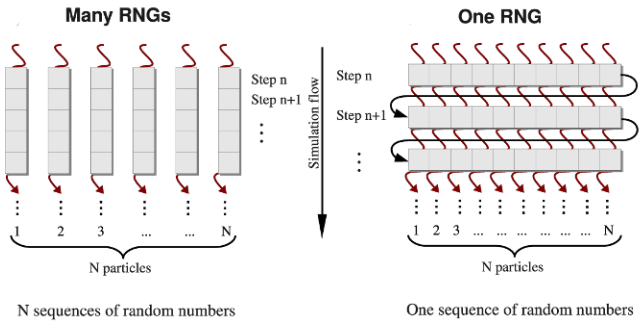

Figure 4: A simplified schematic of the one-RNG-per-thread approach (*left panel*) and the one-RNG-for-all-threads approach (*right panel*). In the one-RNG-per-thread approach, one RNG produces a stream of pseudo-random numbers in each j-th thread of execution $(j = 1, 2, ..., n)$, i.e., the same RNG algorithm (realized in many RNGs) is running in each thread generating different subsequences of the same sequence of random numbers. The one-RNG-for-all-threads approach builds on the ability of different threads to communicate, and, hence, to share the state of just one RNG across an entire GPU device.

We employed these methods to develop GPU-based implementations of the Linear Congruent Generator (LCG) *[Press1992]*, and the Ran2 *[Press1992]*, Hybrid Taus *[Press1992]*, *[Tausworthe1965]*, and additive Lagged Fibonacci algoritms *[Press1992]*, [*[Mascagni2004]*. These generators have been incorporated into the program for Langevin simulations of biomolecules fully implemented on the GPU.

2.2 Benchmark simulations

We tested RNGs implemented on a GPU in Langevin simulations of N Brownian oscillators using the Hybrid Taus and additive Lagged Fibonacci algorithms. We compared the computational time as a function of the system size N for three different implementations of Langevin simulations:

- random numbers and Langevin Dynamics are generated on a CPU;
- random numbers, obtained on the CPU, are transfered to the GPU and used to generate Langevin Dynamics on the GPU;
- random numbers and Langevin Dynamics are generated on the GPU.

The results obtained for the 2.83 GHz Intel Core i7 930 CPU, for the 1.15GHz Tesla C2050 (MIPT) show that starting from $\approx 10^2$ particles, it becomes computationally expensive to generate random numbers on the CPU and transfer them to the GPU in order to generate stochastic trajectories on the GPU (Figure 3, left panel). We observed a ~10-250-fold speedup for Langevin simulations of $N = 10^3 - 10^6$ Brownian particles on the GPU (Figure 5, right panel).

We also benchmarked the computational efficiency of the GPU-based realizations of the Ran2, Hybrid Taus, and Lagged Fibonacci algorithms using Langevin simulations of N Brownian oscillators in three dimensions. For each system size N, we ran one trajectory for 10^6 simulation steps. All N threads were synchronized at the end of each step to emulate an LD simulation run of a biomolecule on a GPU. The associated execution time and memory usage are profiled in Figure 6 below.

On a GPU Ran2 is the most demanding generator as compared to the Hybrid Taus, and Lagged Fibonacci RNGs (Figure 6, left panel). Using Ran2 in Langevin simulations to obtain a single trajectory over 10^9 steps for a system

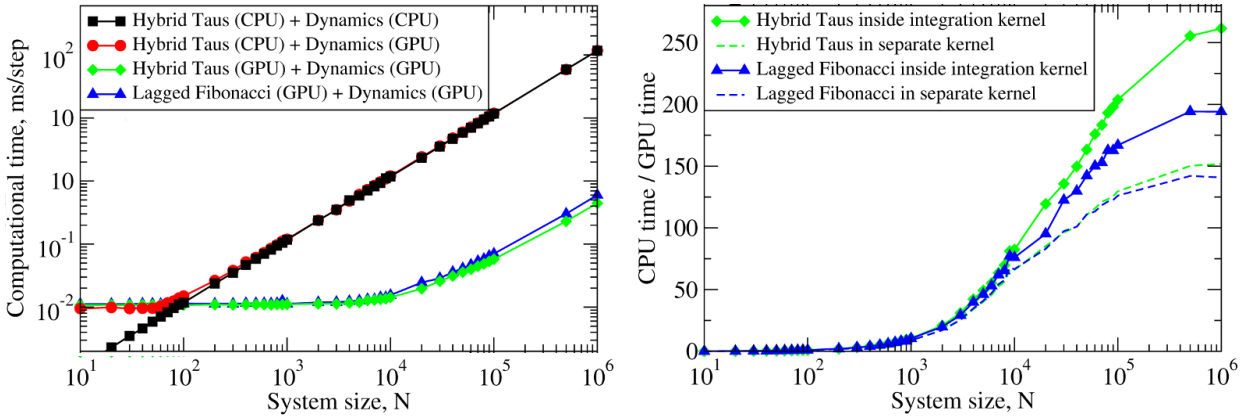

Figure 5: *Left panel:* The computational time for Langevin Dynamics (LD) of N Brownian oscillators with the Hybrid Taus and additive Lagged Fibonacci RNGs. Considered are three implementations, where random numbers and LD are generated on the CPU (Hybrid Taus (CPU) + Dynamics (CPU)), random numbers are obtained on the CPU, transfered to the GPU and used to propagate LD on the GPU (Hybrid Taus (CPU) + Dynamics (GPU)), and random numbers and LD are generated on the GPU (Hybrid Taus (GPU) + Dynamics (GPU) and Lagged Fibonacci (GPU) + Dynamics (GPU)). *Right panel:* The computational speedup (CPU time/GPU time) for LD simulations fully implemented on the GPU and on the single CPU core. Compared are two options when an RNG (Hybrid Taus or Lagged Fibonacci) is organized in a separate kernel or is inside the main (integration) kernel.

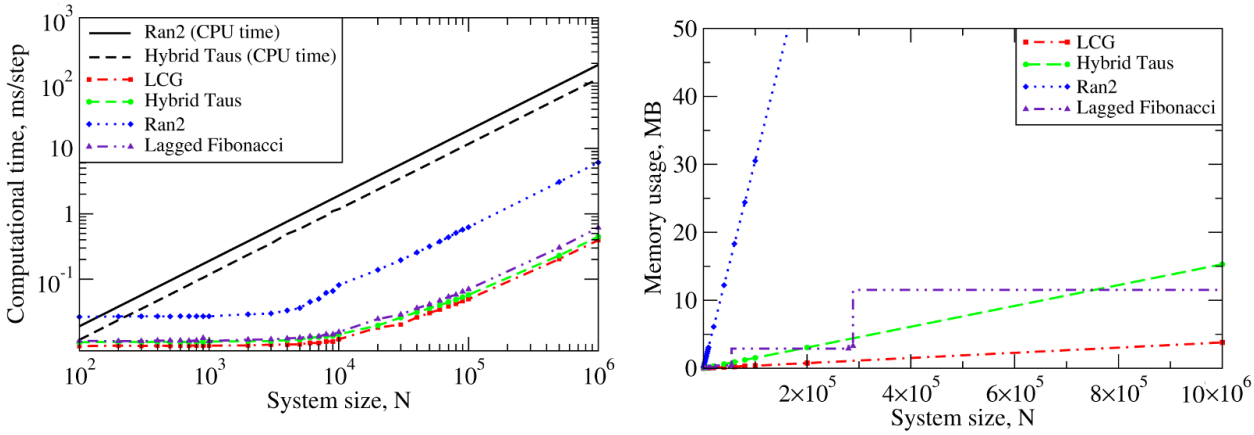

Figure 6: The computational performance of LCG, and the Ran2, Hybrid Taus, and Lagged Fibonacci algorithms in Langevin simulations of N Brownian oscillators on the GPU device. *Left panel:* The execution time (CPU time for Langevin simulations with Ran2 and Hybrid Taus RNGs is shown for comparison). *Right panel:* The memory demand, i.e. the amount of memory needed for an RNG to store its current state. Step-wise increases in the memory usage for Lagged Fibonacci are due to the change of constant parameters for this RNG.

of $N = 10^4$ particles requires additional ~264 hours of wall-clock time. The associated memory demand for Ran2 RNG is quite high, i.e. >250MB for $N = 10^6$ (Figure 6, right panel). Because in biomolecular simulations a large memory area is needed to store parameters of the force field, Verlet lists, interparticle distances, etc., the high memory demand might prevent one from using Ran2 in the simulations of a large system. Also, implementing the Ran2 RNG in Langevin simulations on the GPU does not lead to a substantial speedup (Figure 6, left panel). By contrast, the Hybrid Taus and Lagged Fibonacci RNGs are both light and fast in terms of the memory usage and the execution time (Figure 6). These generators require a small amount of memory, i.e. <15-20MB, even for a large system of as many as $N = 10^6$ particles.

Chapter 3

Using SOP-GPU program

Running SOP-GPU program requires specification of a configuration file (regular text file), containing information about the system of interest and parameters of the simulation:

```
sop-gpu config_file.conf
```

All the information about the simulation protocol and current process is printed out in terminal screen as well as in separate files specified in configuration file.

There are six regimes of simulation available in SOP-GPU package: minimization simulation, equilibrium simulation, point-/plane-pulling simulation, force indentation and heating simulation. Also, SOP-GPU package has implemented hydrodynamic interactions, which can by optionally included in calculation. Parameters and output files for each of these regimes are described in sections below.

3.1 General output

The general output files for any regime of simulation are following:

- Energy output file (usual format *.dat*).
- Trajectory coordinates file (format *.dcd*).
- Restart coordinates file (format *.pdb*).
- Reference coordinates file (first frame of the trajectory, format *.pdb*).
- Final coordinates file (format *.pdb*).

The columns of standard energy output file are:

1. Current simulation step.
2. Average Maxwell-Boltzmann temperature (T, in kcal/mol).
3. Potential energy of covalent bonds (U_{FENE}, in kcal/mol).
4. Potential energy of native interactions (U_{NB}^{ATT}, in kcal/mol).
5. Potential energy of repulsive (long range) interactions (U_{NB}^{REP}, in kcal/mol).
6. Number of native contacts not ruptured (Q).
7. Total potential energy (U_{SOP}, in kcal/mol).
8. Gyration radius (R_{gyr}, optional).

9. Deviation of hydrodynamic tensor from diagonal form (ϵ (see Eq. (3.6), optional).

3.2 Hydrodynamic interactions

In Langevin Dynamics simulations in the overdamped limit, equations of motion for particles of the system are propagated forward in time (see Eq. (1.5) and Eq. (4.2) below) with the amplitude of random force $\rho = \sqrt{2k_BT\zeta/h} = k_BT\sqrt{2/D_{\alpha\alpha}h}$, where α runs over all degrees of freedom. In this approach, which ignores the hydrodynamic coupling of degrees of freedom, all particles are described by the same diffusion coefficient $D = D_{\alpha\alpha} = k_BT/\zeta$.

To account for solvent-mediated many-body effects, one can use an approach proposed originally by Ermak and McCammon *[Ermak1978]* . In this approach, the equation of motion (4.2) is transformed (in absence of external flow) into the following equation:

$$\Delta r_\alpha = \sum_{\beta=1}^{3N} \frac{D_{\alpha\beta}}{kT} F_\beta h + \sqrt{2h} \sum_{\beta=1}^{3N} B_{\alpha\beta} g_\beta \tag{3.1}$$

The first term on the right-hand side is a hydrodynamic tensor $\mathbf{D}$ — a real $3N \times 3N$ matrix, in which an entry $D_{\alpha\beta}$ is a contribution to the diffusion of α-th degree of freedom from the β-th degree of freedom. Alternatively, tensor $\mathbf{D}$ can be represented by an $N \times N$ matrix of 3×3 submatrices $\mathbf{D}_{ij}$, each corresponding to a pair of particles i and j. Also, for the correct distribution of random forces, in the second term in equation (3.1) a real $3N \times 3N$ matrix $\mathbf{B}$ must satisfy the condition $\mathbf{B}^\intercal\mathbf{B} = \mathbf{D}$, where the superscript $^\intercal$ represents the transpose of a matrix. It is easy to show that when in equation (3.1) $\mathbf{D}$ is a diagonal matrix with the identical matrix elements $D_{\alpha\alpha} = kT/\zeta$, we recover equation (3.1).

In SOP-GPU program, we use the Rotne-Prager-Yamakawa (RPY) form of the hydrodynamic tensor $\mathbf{D}$ *[Rotne1969] [Yamakawa1970]*, which is a positive-definite quantity. The submatrices $\mathbf{D}_{ij}$ of RPY tensor are given by the following expressions:

$$\mathbf{D}_{ij} = \frac{kT}{\zeta} \begin{cases} \mathbf{I} & \text{, if } i = j, \\ \left(1 - \frac{9|\mathbf{r}_{ij}|}{32a}\right)\mathbf{I} + \left(\frac{3|\mathbf{r}_{ij}|}{32a}\right)\hat{\mathbf{r}}_{ij} \times \hat{\mathbf{r}}_{ij} & \text{, if } i \neq j \text{ and } |\mathbf{r}_{ij}| < 2a_{HD}, \\ \left(1 + \frac{2a^2}{3|\mathbf{r}_{ij}|^2}\right)\mathbf{I} + \left(1 - \frac{2a^2}{|\mathbf{r}_{ij}|^2}\right)\hat{\mathbf{r}}_{ij} \times \hat{\mathbf{r}}_{ij} & \text{, if } i \neq j \text{ and } |\mathbf{r}_{ij}| \geq 2a_{HD}. \end{cases} \tag{3.2}$$

In equation (3.2), $\mathbf{I}$ is the identity matrix of rank 3, a_{HD} is the hydrodynamic radius of the particle (we assume that a_{HD} is same for all particles, the denotation "$\times$" is used to define the tensor product.

In SOP-GPU program, we utilized an exact approach of computing $\mathbf{B}$ using Cholesky decomposition of $\mathbf{D}$, as well as fast Truncated Expansion approximation (TEA) approach *[Geyer2009]*. In the TEA-based approach, the matrix elements of $\mathbf{B}$ can be rewritten as $B_{\alpha\beta} = C_\alpha b_{\alpha\beta} D_{\alpha\beta}$, and equation (3.1) can be recast as

$$\Delta r_\alpha = \frac{h}{\zeta} \sum_{\beta=1}^{3N} \frac{D_{\alpha\beta}}{D_{\alpha\alpha}} \left(F_\beta + C_\alpha b_{\alpha\beta} \cdot \rho g_\beta\right), \tag{3.3}$$

where

$$b_{\alpha\beta} = \begin{cases} 1 & \text{if } \alpha = \beta, \\ b' & \text{if } \alpha \neq \beta. \end{cases} \tag{3.4}$$

In Eqs. (3.3) and (3.4), C_α and b' are given by

$$C_\alpha = \left(1 + \sum_{\beta \neq \alpha} b'^2 \frac{D_{\alpha\beta}}{D_{\alpha\alpha} D_{\beta\beta}}\right)^{\frac{1}{2}}, \tag{3.5}$$

$$b' = \frac{1 - \sqrt{1 - [(N-1)\epsilon^2 - (N-2)\epsilon]}}{\sqrt{(N-1)\epsilon^2 - (N-2)\epsilon}}, \tag{3.6}$$

where $\epsilon = \langle D_{\alpha\beta}/D_{\alpha\alpha} \rangle$. This linearization procedure allows us to efficiently parallelize the integration algorithm on a GPU.

Cholesky algorithm is implemented by-the-book, i.e. straightforward computation of lower-left-triangular matrix B is carried out with $O(N^3)$ complexity. Due to implementation design, the single trajectory can not contain more than 128 particles is Cholesky factorization is applied.

There is no agreement regarding the value of the hydrodynamic radius a_{HD}. The proposed values vary between $a_{HD} = 1.5 - 5.3$ Å *[Cieplak2009] [Frembgen-Kesner2009]*. However, one must keep in mind that, although the TEA handles overlaps correctly, the RPY tensor is better suited for description of non-overlapping beads. Since the inter-bead $C_\alpha - C_\alpha$-distance in a polypeptide chain is about 3.8 Å, which corresponds to the length of a peptide bond, a_{HD} should not exceed 1.9 Å.

For hydrodynamic interactions parameters see Section *Hydrodynamic interactions parameters*.

3.3 Pulling simulations

Pulling simulations were designed to mimic force-ramp and force-clamp AFM experiments. In this regime, cantilever base is represented by the virtual particle, connected by a harmonic spring to a specified ("pulled") amino acid, mimicking adsorption of residues on the cantilever tip. The system particles specified as "fixed" will be firmly constrained mimicking molecule absorption on the surface. The cantilever base moving with constant velocity (ν_f) extends the cantilever spring, translating into the molecule extension, with the time-dependent force (force-ramp) $\mathbf{f}(t) = f(t)\mathbf{n}$ in the pulling direction $\mathbf{n}$. The force magnitude, $f(t) = r_f t$, applied to cantilever tip, i.e. to the pulled end of the molecule, increases linearly in time t with the force-loading rate $r_f = \kappa\nu_f$ *[Zhmurov2011]*.

For pulling simulation parameters see Section *Pulling parameters*. When pulling is enabled, program will save additional output file (usual format *.dat*) with pulling data. This file has following columns:

1. Current simulation step.
2. Absolute value of the end-to-end distance (X, in Å).
3. Projection of the end-to-end distance on pulling vector (X_{proj}, in Å).
4. Absolute value of the cantilever spring force ($\kappa\Delta x$, in kcal/molÅ).
5. Force vector component (F_x, in kcal/molÅ).
6. Force vector component (F_y, in kcal/molÅ).
7. Force vector component (F_z, in kcal/molÅ).

3.4 Plane-pulling simulations

3.5 Force indentation simulations

Nanoindentation regime adds to the system a cantilever and surface models. In this regime, the cantilever base is represented by the virtual particle, connected to the spherical bead of radius R_{tip}, mimicking the cantilever tip (indentor),

by a harmonic spring. The tip interacts with the particles via the Lennard-Jones potential

$$U_{tip} = \sum_{i=1}^{N} \varepsilon_{tip} \left[A_{tip} \left(\frac{\sigma_{tip}}{|r_i - r_{tip}| - R_{tip}} \right)^{12} + B_{tip} \left(\frac{\sigma_{tip}}{|r_i - r_{tip}| - R_{tip}} \right)^{6} \right] \tag{3.7}$$

thereby producing an indentation on the particle's outer surface. In Eq. (3.7), r_i and r_{tip} are coordinates of the i-th particle and the center of the tip, respectively, ε_{tip} and σ_{tip} are the parameters of interaction, and the summation is performed over all the particles under the tip. The factors A_{tip} and B_{tip} define the attractive and repulsive contributions into the indentor-particle interactions, respectively. For the standard Lennard-Jones potential $A_{tip} = 1$ and $B_{tip} = -2$. If $A_{tip} = 0$ and $B_{tip} = 1$ the interactions are repulsive only. For the cantilever tip, we solve numerically the following Langevin equation of motion:

$$\xi_{tip} \frac{dr_{tip}}{dt} = -\frac{\partial U_{tip}(r_{tip})}{\partial r_{tip}} + \kappa((r^0_{tip} - \nu_f t) - r_{tip}) \tag{3.8}$$

where r^0_{tip} is the initial position of spherical tip center (ν_f is the cantilever base velocity; κ is the cantilever spring constant), and ξ_{tip} is the friction coefficient for a spherical particle of radius R_{tip} in water. To generate the dynamics of the biological particle of interest tested mechanically, the Eqs. (1.1) — (1.5) for the particle (see above) and Eqs. (3.7) and (3.8) for the indentor (spherical tip) should be solved numerically.

The substrate surface is also modeled using Lennard-Jones potential with parameters of interactions ε_{surf} and σ_{surf} and factors A_{surf} and B_{surf} (see Eq. (3.7)). The surface is represented by a number of particles and interaction potential is calculated between each particle in system and particles on the surface.

The cantilever base moving with constant velocity (ν_f) exerts (through the tip) the time-dependent force (force-ramp) $\mathbf{f}(t) = f(t)\mathbf{n}$ in the direction $\mathbf{n}$ perpendicular to the particle surface. The force magnitude, $f(t) = r_f t$, exerted on the particle increases linearly in time t with the force-loading rate $r_f = \kappa\nu_f$ *[Kononova2013b] [Kononova2014]* .

For force indentation simulation parameters see Section *Force indentation parameters*. The results of indentation will be saved in additional output file (usual format *.dat*) with the following columns:

1. Current simulation step.
2. Distance traveled by cantilever base (Z, in Å).
3. Average molecular force acting on a cantilever tip projected onto chip movement direction (F_{proj}, in kcal/molÅ).
4. Average absolute value of a molecular force, acting on a cantilever tip, (F, in kcal/molÅ).
5. Absolute value of the cantilever spring force at a given step ($\kappa\Delta x$, in kcal/molÅ).
6. Absolute value of the cantilever spring force average ($\overline{\kappa\Delta x}$, in kcal/molÅ).
7. Molecular force vector component (F_x, in kcal/molÅ).
8. Molecular force vector component (F_y, in kcal/molÅ).
9. Molecular force vector component (F_z, in kcal/molÅ).
10. Current cantilever tip coordinate (X_x, in Å).
11. Current cantilever tip coordinate (X_y, in Å).
12. Current cantilever tip coordinate (X_z, in Å).
13. Current cantilever base coordinates (Z_x, in Å).
14. Current cantilever base coordinates (Z_y, in Å).
15. Current cantilever base coordinates (Z_z, in Å).

3.6 Heating simulations

Although coarse-grained models are known to be not very accurate in describing heat-induced unfolding of molecules, SOP-model still can provide good qualitative results. When heating option is on, temperature of the water bath (i.e. strength of random force, see Eq. (4.2) below) increases gradually during the simulation process. Heating parameters are described in Section *Heating parameters*.

Chapter 4

Units

For numerical evaluation of the Eq. (1.5) in time, it can be written in form

$$\xi \frac{r_i^{t+1} - r_i^t}{\Delta t} = F_i^t + G_i^t \tag{4.1}$$

When divide both sides of Eq. (4.1) by particle mass m and express the change of coordinates $\Delta r_i^t = r_i^{t+1} - r_i^t$ arrive to

$$\Delta r_i^t = \frac{\Delta t}{\xi/m} \frac{1}{m} (F_i^t + G_i^t)$$

From the equation for harmonic oscillator, $\xi/m = \zeta/\tau_L$ is damping coefficient. Here ζ is dimensionless damping ratio and $\tau_L = \sqrt{ma^2/\varepsilon_h}$ is characteristic time for underdamped motion of spherical particle of mass m and radius a with energy scale ε_h. According to Langevin equation, the random force $G_i^t = g_i^t \sqrt{2\zeta k_B T/h}$, where g_i^t is random number from the interval $[0, 1]$. Hence

$$\Delta r_i^t = \frac{\Delta t \tau_L}{\zeta m} (F_i^t + g_i^t \sqrt{2\zeta k_B T/h}) \tag{4.2}$$

From the Stokes-Einstein friction theory $\xi = 6\pi\eta a$ for a spherical particle of radius a in a liquid with viscosity η. Therefore $\zeta = 6\pi\eta a^2/\sqrt{m\varepsilon_h}$. In the program $\zeta = 50$. This was obtained for $a \sim 5$ Å, $m \sim 3 \times 10^{-22}$ g (mass of a residue) and the bulk water viscosity $\eta = 0.01$ gs $^{-1}$ cm $^{-1}$.

In general, a varies between 3.8 Å to 5 Å, while m varies between 3×10^{-22} g to 5×10^{-22} g. In the simulations $a = 3.8$ Å. Because of the fact that ζ depends on ε_h, every time when ε_h was changed, valid m value should be calculated, which gives the value $\zeta = 50$.

Example: for $\varepsilon_h = 1$ kcal/mol from the above equation for ζ we find that $m = 4.3 \times 10^{-22}$ g which is a valid value. For $\varepsilon_h = 1.5$ kcal/mol, we get $m = 3 \times 10^{-22}$ g which is still a valid value. After finding the mass m, we can go back to the expression for τ_L and get its value. For example, for $\varepsilon_h = 1$ kcal/mol we get $\tau_L = 3$ ps while for $\varepsilon_h = 1.5$ kcal/mol, we get $\tau_L =$ ps.

For the overdamped Langevin dynamics the characteristic time is $\tau_H = \zeta\varepsilon_h\tau_L/kT = 6\pi\eta a^3/kT$. In order to get it in units of ps, both ε_h and k_BT need to be of the same units. Since ε_h is in kcal/mol, k_BT should be also in kcal/mol (at $T = 300$ K $k_BT = 0.6$ kcal/mol). Therefore the simulation time step $\Delta t = h \cdot \tau_H$ is also in units of ps. With the standard parameters ($\eta = 0.01$ gs $^{-1}$ cm $^{-1}$, $T = 300$ K and $a = 3.8$ Å), $\tau_H = 248$ ps. The parameter h can be specified in configuration file.

In the pulling/indentation simulation, cantilever velocity is defined as $\nu_f = \Delta x/(n_{av} \cdot h \cdot \tau_H)$ where Δx is displacement of virtual bead, representing cantilever base, during n_{av} steps, it is given in Å. The force is calculated in kcal/(molÅ), to get the force in pN, one need to multiplied by 70. Therefore, the cantilever spring constant κ should be also specified in the units of kcal/(mol ^{2}).

Chapter 5

Topology

The SOP-GPU package includes two utilities for coarse-graining the system. The old one, `sop-top` can only create C_α-based model, but there is a functionality to make tandems out of the monomer. The new utility `sop-top2` uses a flexible coarse-graining configuration config, which allows one to create various coarse-grained models (e.g. C_α or $C_\alpha - C_\beta$).

5.1 Old sop-top utility

Creating of coarse-grained structure with corresponding topology file can be performed running `sop-top` utility as follow:

```
sop-top top_config_file.top
```

As with the main program, configuration file should be passed as the first parameter to `sop-top`. Executing the command above will generate new, coarse-grained PDB file and the topology file in Gromacs TOP format. The PDB file is used only to store coordinates of the particles and all the connectivities are described in TOP file. This configuration file can use the same features as configuration file for SOP-GPU, as described in Section *Input parameters file*. Topology is created from the original (full-atomic) PDB file using its `ATOM` and `SSBOND` entries. All C_α atoms are added into `[ atoms ]` section of topology file generated. Backbone connectivity and disulfide bonds along with their equilibrium (PDB) distances are collected into `[ bonds]` section. Native contacts are determined based on two cut-off distances. The first one relates to a maximum $C_\alpha - C_\alpha$ distance for two amino-acids in native contact (*simple Go definition*), the second one is the cut-off for the minimal distance of two heavy atoms in corresponding amino-acids side-chains (*full Go definition*). Along with the indexes of amino-acids i and j, PDB distance r_{ij}^0 and value of ε_h are saved for each pair qualify. ε_h can be specified as constant value for all native pairs or can be taken from occupancy of beta columns of original PDB. In later case, geometric average of two values listed for amino-acids i and j are taken.

5.2 New sop-top (sop-top2) utility

In some cases, the C_α-representation is not just sufficient. The `sop-top2` utility allows for the custom coarse-graining of the initial full-atomic system. The coarse-graining in this case relies on the coarse-graining configuration file, in which one can find a description on how to coarse-grain a particular amino-acid. For the convinience, two configs are supplied with the SOP-GPU package: one to get the C_α representation and the other — to get the $C_\alpha - C_\beta$ representation of the protein system. The `sop-top2` program takes the path to the configuration file as an argument. In this file, one should specify the following parameters as an input: the path to the initial (all-atom) PDB file and the path to the coarse-grained configuration file.

5.2.1 The coarse-graining configuration: $C_\alpha - C_\beta$ model

The coarse-graining configuration file starts with the list of masses for all the atoms present in the system:

```
MASS     1 H      1.00800  ! Hydrogen
MASS     2 C     12.01100  ! Carbon
MASS     3 N     14.00700  ! Nitrogen
MASS     4 O     15.99900  ! Oxygen
MASS     5 S     32.06000  ! Sulphur
```

The description of the coarse-graining for each amino-acid follows. For instance, in the $C_\alpha - C_\beta$ approach, the alanine entry will be:

```
RESI ALA
BEAD CA CA
REPR N HN CA HA C O
COOR CA
CHAR 0.0
CONN +CA CB
ENDBEAD
BEAD CB SC
REPR CB HB1 HB2 HB3
COOR CB
CHAR 0.0
ENDBEAD
ENDRESI
```

Here, `RESI` and `ENDRESI` keywords encapsulate the description of the residue, which contain two beads entries: one for the C_α-bead and one for the side-chain (C_β) bead. Each bead starts with the keyword *BEAD* followed by the name and type of the bead. For each bead, the following information should be provided: (1) Which atoms this bead represents (their names as they are in the initial PDB file are listed after `REPR` keyword). (2) Where the created bead should be placed (the name (or names) for the positioning atoms should be provided after `COOR` keyword). (3) The charge (`CHAR`) is the charge assigned to the bead (not used in SOP model). The entry `CONN` lists the covalent bonds that should be added for a particular bead. After this keyword listed are the names of the beads with which the current bead is connected to. The syntax resembles the CHARMM forcefield topology file: the + sign means that the connection is with the next residue in the polypeptide chain, – — with the preceeding. Each covalent bond should be added once (i.e. if $C_\alpha - C_\beta$ bond is added for the C_α-atom, there is no necessity to add this bond for the C_β-atom, as the alanine entry above illustrates). In the entry above, the C_α-bead (`CA`) is connected to the C_α-bead of the next residue (`+CA`) and to the C_β-bead of the same residue (`CB`). There is no `CONN` entry for the C_β-bead, since the $C_\alpha - C_\beta$ is already listed in the the C_α bead section.

In other words, the entry for the alanine above, reads: In the residue `ALA`, first bead is the `CA` (C_α) bead of the type `CA` (`BEAD CA CA`). It represents the atoms of the backbone (`REPR N HN CA HA C O`) and should be placed on the position of the C_α-atom of the alanine residue (`COOR CA`). Its charge is zero (`CHAR 0.0`). It is covalently connected to the C_α-bead of the next residue and the side-chain (C_β) bead of the same residue. The second bead of the alanine residue is the `CB` (C_β) bead of type `SC` (`BEAD CB SC`). It represents the side-chain atoms (`REPR CB HB1 HB2 HB3`) and should be placed on the position of the C_β-atom from the initial all-atom PDB (`COOR CB`). Its charge is also zero (`CHAR 0.0`). The description of the bead and residue ends here.

The corse-graining configuration file should provide similar description for all the residues in the initial PDB file (in general the description of the coarse-graining for all 20 essential amino-acids should be suffitient). If you system has some non-standart residues, sugars, nucleic acids, etc., you will need to add the coarse-graining description to the coarse-graining config file you use. To do so, you need to decide, how many beads for the residue you want to add, where you want to place them, which atoms they represent, what is the total charge of these atoms. The connection entry for each bead should include the covalent connectivity within the residue and(or) the connectivity to the next (preceding) residues, marked with + (−) sign.

The provided with SOP-GPU $C_\alpha - C_\beta$ coarse-graining configuration file is called `aa_to_cg.inp` and includes the

following description for the side chains of 20 essential amino-acids: (i) there is no side-chain for GLY; (ii) for the aliphatic amino acids (ALA, VAL, LEU, and ILE), the C_β-bead is placed at the position of the center of mass of the side-chain; (iii) for residues THR and SER, the C_β-atom is placed at the position of the hydroxyl oxygen; (iv) the side-chain of the acidic amino acids (ASP and GLU) is placed at the center of mass of the COO-group; (v) the side-chain of the basic amino acids (LYS and ARG) is placed at the center of mass of the NH_3+-group; (vi) for ASN and GLN, the C_β-atom is placed at the position of the center of mass of the group $CO - NH_2$; (vii) aromatic side-chains in PHE and TYR are represented by a single C_β-bead placed at the geometrical center of the rings (for TYR, the bead representing the OH-group is also added); (viii) TRP side-chain having a double-ring structure is represented by two beads placed in the geometrical centers of the rings; (ix) HIS is represented by a single bead placed at the geometrical center of the five-member ring forming the side-chain; (x) sulfur-containing amino acids (MET, CYS) are represented by a side-chain bead, placed at the position of the sulfur atom; and (xi) the C_γ-atom in PRO is represented by the C_β-bead linked to its C_α bead and to the C_α bead of the residue before, thus forming a cyclic bond structure.

5.2.2 The coarse-graining configuration: C_α model

The coarse-graining configuration file C_α-based model is also provided with the package. It is called `aa_to_cg_ca.inp`. The entry for Alanine residue in this file is:

```
RESI ALA
BEAD CA CA
REPR CA
COOR CA
CHAR 0.0
CONN +CA
ENDBEAD
ENDRESI
```

Here, only one C_α bead of type `CA` is added (`BEAD CA CA`) on the position of the C_α atom (`COOR CA`). It has zero charge (`CHAR 0.0`) and connected to the C_α bead of the next residue in the polypeptide chain (`CONN +CA`).

5.3 Additional covalent bonds

In many proteins, the covalent bonding is not limited by the polypeptide backbone. The most common example is the disilfide bonding. In the `sop-top2` utility, these bonds can be added by providing additional file, that contain the list of additional bonds to be added. The path to this file can be specified by the parameter **additional_bonds**. If this parameter is absent, no additional bonds will be added. In this file, each line correspond to one bond and starts with the `CONN` keyword followed by the chain-residue-name triplet for two beads to be connected:

```
CONN A 49 SG B 76 SG
```

The line above tell the programm to add the disulfide bond between the residue 49 from the chain A and residue 76 of the chain B. The names `SG` for both of these residues are the names for the side-chain atoms of the cystene residues in the $C_\alpha - C_\beta$ coarse-graining approach. Since there are no SG beads in the C_α model, the same disulfide bond would be:

```
CONN A 49 CA B 76 CA
```

5.4 Topology file

There are three types of interactions in SOP model: covalent interactions, native interactions and repulsive pairs. All these should be listed in Gromacs-style topology file (*.top*). SOP topology file has four sections: `[ atoms ]`, that lists all the particles (C_α and C_β atoms) in the system (including information about residue ID, residue and chain name,

etc.), and three sections that correspond to three types of interactions: `[ bonds ]` for covalent bonds, `[ native ]` for native interactions and `[ pairs ]` for repulsive pairs. The `[ atoms ]` section follows the Gromacs-style atom description:

```
[atoms]
;   nr       type  resnr residue  atom   cgnr     charge       mass
     0         CA      1     LEU     CA      A       0.00     56.044
     1         CG      1     LEU     CG      A       0.00     57.116
     2         CA      2     ILE     CA      A       0.00     56.044
     3         CD      2     ILE     CD      A       0.00     57.116
...
```

Here, the columns correspond to particle ID, particle type, number of residue, particle atom name, charge and mass.

The last three sections consist of the list of interacting particles IDs, function type and set of specific parameters. Particle IDs correspond to internal program indexes and start from 0, function type column is set to 1 for all pairs and ignored, parameters are specific for each interaction type as described below. More details on file format can be found in Gromacs Manual.

5.4.1 `[ bonds ]` section

Typical `[ bonds ]` section includes lines similar to the following:

```
[ bonds ]
;  ai    aj funct            c0            c1            c2            c3
    0     1     1    3.81188
    1     2     1    3.77232
    2     3     1    3.79319
...
```

Covalent bonds include backbone interactions and disulfide S-S bonds. Potential energy function term that corresponds to covalent bonds interaction is described by U_{FENE} (Eq. (1.2)) in Eq. (1.1), where summation is made over all lines in `[ bonds ]` section of the topology file, i and j correspond to the particles IDs listed in the line, distance r_{ij} is computed from particles coordinates, r_{ij}^0 is the distance between two corresponding C_α atoms in native state (PDB file), listed as the first parameter in the line (column `c0`, see sample listing above).

5.4.2 `[ native ]` section

```
[ native ]
;  ai    aj funct            c0            c1            c2            c3
    5     9     1    5.85792    1.50000
    5    10     1    7.06482    1.50000
    5    35     1    6.64479    1.50000
...
```

In SOP model, native interactions (U_{NB}^{ATT}, see Eq. (1.1)) are described by full Lennard-Jones potential (Eq. (1.3)). Each term in the sum corresponds to one line in `[ native ]` section. Apart from IDs of interacting particles, equilibrium distance r_{ij}^0 (column `c0`) and the strength of non-bonded energy interaction, ε_h (column `c1`), are listed. r_{ij}^0 is the distance between C_α atoms in native state (PDB file), value of ε_h is usually between 1.0 and 1.5 kcal/mol and can be obtained from att-atom MD simulations.

5.4.3 `[ pairs ]` section

```
[ pairs ]
;  ai    aj funct            c0            c1            c2            c3
    0     2     1
    0     3     1
    0     4     1
...
```

Pairs section correspond the third term in Eq. (1.1) (Eq. (1.4)). There is no pair-specific parameters in this section, only indexes are listed. Note, that this list scales as $\sim N^2$ with the system size N, and saving all possible repulsive pairs in the topology file would lead to very large files. In SOP-GPU program, this section is used only for small systems and when trajectory massive-production is employed. When large system is simulated, dual-range cut-off algorithm is utilized and only pairs withing bigger cut-off are kept (pairlist). Pairlist is updated using exclusion principle: only those pairs that are withing cut-off distance but not in the list of excluded pairs added. Verlet list is built from this pairlist based on smaller cut-off distance and used when potential function and forces are computed. Excluded pairs are those already listed in `[ bonds ]` and `[ native ]` sections.

5.5 Parameter for the topology creation

Both `sop-top` and `sop-top2` use the parameters file, path to which is passed as a first argument. The parameters one can use are:

- **structure** <*filename*>

  Type: Path to the file.

  Status: Required.

  Purpose: Path to the initial (all-atomic) PDB file.

- **additional_bonds** <*filename*>

  Type: Path to the file.

  Status: Optional.

  Purpose: Path to the file with the list of additional bonds (e.g. S-S bonds)

- **topology** <*filename*>

  Type: Path to the file.

  Status: Required.

  Purpose: Path to the output topology (*.top*) file.

- **coordinates** <*filename*>

  Type: Path to the file.

  Status: Required.

  Purpose: Path to the output coarse-grained *.pdb* file.

- **topology_psf** <*filename*>

  Type: Path to the file.

  Status: Optional. Uses with `sop-top2` only.

  Purpose: The path to save the topology in *.psf* (NAMD) format (for VMD visualisation).

- **topology_natpsf** <*filename*>

  Type: Path to the file.

  Status: Optional. Uses with `sop-top2` only.

  Purpose: The path to save the topology in *.psf* (NAMD) format (for VMD visualisation). Native contacts will be saved instead of covalent bonds in the corresponding section. Convinient for the native contacts inspection.

- **R_limit_bond** <*cut-off distance*>

  Type: Float.

  Status: Required.

  Purpose: The cut-off value for the distance between beads. If two beads are within this distance in the provided structure, they considered to form native contact.

- **SC_limit_bond** <*cut-off distance*>

  Type: Float.

  Status: Optional.

  Default value:

  Purpose: The cut-off value for the distance between side-chain beads. If two atoms listed in the `REPR` section of the amino acid are within this distance in the provided structure, the beads considered to form native contact.

- **eh** <*native energy scale*>

  Type: Float or `O`/`B`.

  Status: Required.

  Purpose: The value for the ε_h parameter. If the float value is given, the value is the same for all contacts. If the `O` or `B` is specified, the value is taken as a geometric average of the beta or occupancy column value.

Chapter 6

Input parameters file

6.1 General features

Input parameters file contains all the simulation parameters listed as tab or space separated pairs of name and value. Remarks are allowed using "#" character. To simplify creation of multiple configuration/output files, parameters values support macroses. This can be use full in order to avoid overwriting of the output files if multiple trajectories are running in parallel, for example when many-runs-per-GPU approach is used. Any parameter name in the file can be used as macros, additional macroses can be added using same name-value syntax as for regular parameters. To use macros, parameter name included in any other parameter value should be surrounded with "<" and ">" characters. For example, the following lines:

```
run 3
DCDfile <run>.dcd
```

result in the value for the output file name "*3.dcd*".

6.2 Device parameters

- **device** <*device ID*>

  Type: Integer.

  Status: Required.

  Default value: 0.

  Purpose: ID of NVidia card to run simulations on. Use "nvidia-smi" or "deviceQuery" from NVidia SDK to check devices.

- **block_size** <*integer*>

  Type: Integer.

  Status: Optional.

  Default value: 256.

  Purpose: Set the number of threads per block. Can be specified for every potential individually, using **block_size_covalent**, **block_size_native**, **block_size_pairs**, **block_size_pairlist** and **block_size_possiblepairs**.

- **max_covalent**: <*integer*>

Type: Integer.

Status: Optional.

Default value: 8.

Purpose: Set the maximum number of pairs per residue for covalent interactions.

- **max_native** <*integer*>

  Type: Integer.

  Status: Optional.

  Default value: 128.

  Purpose: Set the maximum number of pairs per residue for native interactions.

- **max_pairs** <*integer*>

  Type: Integer.

  Status: Optional.

  Default value: 512.

  Purpose: Set the maximum number of pairs per residue for pairs list.

- **max_possiblePairs** <*integer*>

  Type: Integer.

  Status: Optional.

  Default value: 4096.

  Purpose: Set the maximum number of pairs per residue for possible pairs list.

6.3 Structure parameters

- **name** <*protein name*>

  Type: String.

  Status: Required.

  Purpose: Name, assigned to the structure. Used mostly for files naming.

- **topology** <*filename*>

  Type: Path to the file.

  Format: .top

  Status: Required.

  Purpose: Path to the structure topology file (see Section *Topology*).

- **coordinates** <*filename*>

  Type: Path to the file.

  Format: .pdb

  Status: Required.

  Purpose: Path to the structure initial coordinates file.

6.4 General simulation parameters

- **numsteps** <*steps count*>

  Type: Long integer.

  Status: Required.

  Purpose: Number of simulation steps.

- **timestep** <*time*>

  Type: Float.

  Units: τ_H (see Section *Units*).

  Status: Required.

  Purpose: Time-scale of one simulation step.

- **seed** <*random seed*>

  Type: Integer.

  Status: Optional.

  Default value: Taken from current date and time.

  Purpose: Initial random seed used for random force. Actual seed is computed by adding **run** or **firstrun** (whichever is defined) to this value.

- **run** <*trajectory number*>

  Type: Integer.

  Status: Optional.

  Default value: -1

  Purpose: Trajectory number when running only one trajectory per GPU ("one-run-per-GPU approach"). Usually used for files naming. Alternatively, **firstrun** and **runnum** can be used.

- **firstrun** <*first trajectory number*>

  Type: Integer.

  Status: Required if **run** is not specified.

  Purpose: Number of first trajectory when "using many-runs-per-GPU" approach.

- **runnum** <*number of trajectories*>

  Type: Integer.

  Status: Required if **firstrun** is specified.

  Purpose: Total amount of trajectories for running in parallel on one GPU when using "many-runs-per-GPU" approach. Trajectories from **firstrun** to **firstrun** + **runnum** will be started. Note, that in this case all output files require "<run>" macros, so that the output data will be saved into different files for different trajectories.

6.5 Force-field parameters

- **temperature** <*temperature value*>

Type: Float.

Units: kcal/mol.

Status: Optional.

Default value: 0.6.

Purpose: Set the temperature to heat bath (random force). Default value 0.6 kcal/mol $\approx$ 300 K.

- **zeta** <*:math:'zeta' value*>

  Type: Float.

  Units: Dimensionless.

  Status: Optional.

  Default value: 50.0.

  Purpose: Friction coefficient for amino acid in viscous environment. For a spherical particle: $\zeta = 6\pi\eta a^2/\sqrt{m\varepsilon_h}$, where $\eta = 0.01$ gs $^{-1}$ cm $^{-1}$ is a bulk water viscosity, $m \sim 3 \times 10^{-22}$ g is an average mass of an amino acid residue, $a = 3.8$ Å is length of amino acid amide bond, ε_h is an average strength (hydrophobicity) of native interactions, it is taken from topology file and usually between 0.9 and 1.5.

- **kspring_cov** <*spring constant*>

  Type: Float.

  Units: kcal/molÅ.

  Status: Optional.

  Default value: 20.0.

  Purpose: Spring constant k of covalent interactions in FENE potential (Eq. (1.2)).

- **R_limit** <*tolerance in distance change*>

  Type: Float.

  Units: Å.

  Status: Optional.

  Default value: 2.0.

  Purpose: The tolerance in the change of the covalent bond distance R_0 parameter in FENE potential (Eq. (1.2)).

- **a** <*covalent bond length*>

  Type: Float.

  Units: Å.

  Status: Optional.

  Default value: 3.8.

  Purpose: Default distance between C_α-atoms in polypeptide chain. Amino acid size parameter σ_l in repulsive Lennard-Jones potential as an a (Eq. (1.4)).

- **el** <*repulsive energy factor*>

Type: Float.

Units: kcal/mol.

Status: Optional.

Default value: 1.0.

Purpose: Energy factor ε_l of repulsive interactions (Eq. (1.4)).

6.6 Pairs lists parameters

- **pairs_cutoff** <*pairs cut-off distance value*>

  Type: Float.

  Units: Å.

  Status: Optional.

  Default value: 20 Å.

  Purpose: Cut-off distance for a pair of amino acids from a pair list defining whether repulsive interactions between these particles will be taken into account or not. If distance between two particles is larger then this value, force is not computed.

- **pairlist_cutoff** <*pairs (Verlet) list cut-off distance value*>

  Type: Float.

  Units: Å.

  Status: Optional.

  Default value: 20 Å.

  Purpose: Cut-off distance for a pair of amino acids defining whether this pair will be added to pairs (Verlet) list or not. If the distance between two particles is less then this value, pair is added into pairs (Verlet) list.

- **pairs_threshold** <*possible pairs cut-off distance value*>

  Type: Float.

  Units: Å.

  Status: Optional.

  Default value: 200 Å.

  Purpose: Cut-off distance using to generate the list of possible pairs. This list is generated based on exclusion principle: if a pair of amino acids does not belong to covalent bond or native bond and distance between them is less than the threshold value, then the pair is added into possible pairs list.

- **pairs_freq** <*number of steps*>

  Type: Float.

  Status: Optional.

  Default value: 1000.

  Purpose: Frequency of the pairs (Verlet) list update.

- **possiblepairs_freq** <*number of steps*>

Type: Float.

Status: Optional.

Default value: 100000.

Purpose: Frequency of the possible pairs list update.

6.7 Hydrodynamic interactions parameters

- **hi_on** <*on/off*>

  Type: Boolean.

  Status: Optional.

  Default value: off.

  Purpose: Switch on calculation of hydrodynamic interactions (see Section *Hydrodynamic interactions*).

- **hi_exact** <*on/off*>

  Type: Boolean.

  Status: Optional.

  Default value: off.

  Purpose: Use Cholesky-based method of the hydrodynamic tensor calculation, which is exact approach (see Section *Hydrodynamic interactions*). If disabled, TEA approach is used.

- **hi_a** <*hydrodynamic radius value*>

  Type: Float.

  Units: Å.

  Status: Optional.

  Default value: 1.8.

  Purpose: Hydrodynamic radius a_{HD} of a particle.

- **hi_epsilon_freq** <*number of steps*>

  Type: Integer.

  Status: Required, if **hi_on** is on and **hi_exact** is off.

  Purpose: Frequency of updating ersatz coefficients for TEA method (ϵ in Eq. (3.6)). Recommended value are in range 1–10.

- **hi_capricious** <*on/off*>

  Type: Boolean.

  Status: Optional.

  Default value: on.

  Purpose: Whether to abort execution on weird values of the hydrodynamic tensor in TEA approach. See also **hi_epsmax**.

- **hi_unlisted** <*on/off*>

Type: Boolean.

Status: Optional.

Default value: on.

Purpose: Whether to calculate all particle-particle interactions, or use the pairs (Verlet) list. Using pairs list is heavily discouraged. If **hi_exact** is on, this parameter is ignored and all particle-particle interactions are always computed.

- **hi_epsmax** <*accuracy value*>

  Type: Float.

  Status: Optional.

  Default value: 999.0.

  Purpose: Abort simulation if ϵ (see Eq. (3.6)) reaches this value and **hi_capricious** is on; since ϵ will never exceed 1, the default parameter value will never trigger abortion.

6.8 Pulling parameters

- **pulling** <*on/off*>

  Type: Boolean.

  Status: Optional.

  Default value: off.

  Purpose: Switch on the pulling regime with pulling parameters (see Section *Pulling simulations*).

- **k_trans** <*cantilever spring constant*>

  Type: Float.

  Units: kcal/mol ^{2}.

  Status: Optional.

  Default value: 0.05.

  Purpose: The value of cantilever spring constant κ.

- **fconst** <*pulling force*>

  Type: Float.

  Units: kcal/molÅ.

  Status: Required, if **deltax** is not specified.

  Default value: 0.0.

  Purpose: The value of applied external force, using to run pulling simulations with force-clamp protocol.

- **deltax** <*pulling speed*>

  Type: Float.

  Units: Å.

  Status: Required, if **fconst** is not specified.

  Default value: 0.0.

Purpose: The value defining the cantilever base velocity in simulations with force-ramp protocol. Position of the cantilever base will be displaced by **deltax** every **pullFreq** steps. Actual pulling speed can be calculated as **deltax**/(**pullFreq** · **timestep**) (see Section *Units*).

- **pullFreq** <*number of steps*>

  Type: Integer.

  Status: Optional.

  Default value: **nav**.

  Purpose: The frequency of cantilever base displacement by **deltax**.

- **pullDirection** <*string*>

  Type: "endToEnd" / "vector"

  Status: Required.

  Default value: endToEnd

  Purpose: Direction in which external force if applied. If "endToEnd", cantilever base will move along end-to-end vector, which is obtained from positions of **fixedEnd** and **pulledEnd** residues. If "vector" is chosen, it also requires specification of **pullVector**.

- **pullVector** < *x, y, z normalized coordinates*>

  Type: Vector.

  Status: Required, if **pullDirection** is "vector".

  Purpose: Direction vector of external force application.

- **fixedEnd**, **pulledEnd** <*residue ID* >

  Type: Integer.

  Status: Required.

  Purpose: The residue IDs, which will be used to calculate end-to-end distance.

- **fixed** <*list of residue IDs*>

  Type: List of integers.

  Status: Required.

  Purpose: List of amino acids, which will be fixed during the pulling simulations. The values should be space-separated, interval of the values can be specified as "*value_1* to *value_N*".

- **pulled** <*list of residue IDs*>

  Type: List of integers.

  Status: Required.

  Purpose: List of amino acids to which external force **fconst** will be applied (force-clamp protocol) or which will be displaced by **deltax** (force-ramp protocol). The values should be space-separated, interval of the values can be specified as "*value_1* to *value_N*".

- **pullOutput** <*filename*>

  Type: Path to the file.

  Status: Optional.

  Default value: "pull.<name>_<author><run>.dat"

Purpose: Path to output file of pulling simulations (see Section *Pulling simulations*).

6.9 Force indentation parameters

- **indentation** <*on/off*>

  Type: Boolean.

  Status: Optional.

  Default value: off.

  Purpose: Switch on the force indentation regime with indentation parameters (see Section *Force indentation simulations*). Virtual particles, corresponding to cantilever tip, cantilever base and substrate surface will be added to the coordinates output files.

- **indentationChip** <*position vector x, y, z*>

  Type: Vector.

  Units: Å.

  Status: Required.

  Purpose: Initial position of the virtual particle representing cantilever base (i.e. cantilever "chip").

- **indentationTip** <*position vector x, y, z*>

  Type: Vector.

  Units: Å.

  Status: Optional.

  Default value: **indentationChip**.

  Purpose: Initial position of the center of virtual sphere representing cantilever tip.

- **indentationDirection** <*direction vector x, y, z*>

  Type: Vector.

  Status: Required.

  Purpose: Direction of the cantilever base movement.

- **indentationTipR** <*radius value*>

  Type: Float.

  Units: Å.

  Status: Required.

  Purpose: Radius of the virtual sphere representing cantilever tip.

- **indentationTipKs** <*spring constant value*>

  Type: Float.

  Units: kcal/mol ^{2}.

  Status: Required.

  Purpose: Spring constant of the cantilever.

- **indentationDeltaX** <*cantilever base velocity*>

Type: Float.

Units: Å.

Status: Required.

Purpose: The value define the displacement of the virtual particle, representing cantilever base, every **indentationFreq** steps. Actual cantilever base velocity can be calculated as **indentationDeltaX/(indentationFreq** · **timestep**) (see Section *Units*).

- **indentationSigma** <*range of LJ interactions*>

  Type: Float.

  Units: Å.

  Status: Optional.

  Default value: 1.0.

  Purpose: Repulsive distance for the Lennard-Jones potential σ_{tip} (see Eq. (3.7)). Note that potential is shifted to the surface of the cantilever tip sphere.

- **indentationEl** <*energy factor of LJ interactions*>

  Type: Float.

  Units: kcal/mol.

  Status: Optional.

  Default value: 1.0.

  Purpose: Repulsive energy factor ε_{tip} for Lennard-Jones potential (see Eq. (3.7)).

- **indentationShowTipSurf** <*yes/no*>

  Type: Boolean.

  Status: Optional.

  Default value: no.

  Purpose: Define whether the program should save coordinates of the cantilever tip and base as well as all the points representing substrate surface in *.dcd* file together with coordinates of the modeled system during indentation simulation. Useful for representation purposes. Tip will be represented as two particles (particle for the cantilever base and particle for the cantilever tip) with chain identificator "T" in *.pdb* file, surface particles will have chain identificator "M".

- **indentationTipA / indentationTipB** <*dimensionless constants*>

  Type: Float.

  Status: Optional.

  Default value: 0 and 1, respectively.

  Purpose: Shape of the Lennard-Jones potential for the cantilever tip A_{tip} and B_{tip} (see Eq. (3.7), Section *Force indentation simulations*).

- **indentationTipSigma** <*range of LJ interactions*>

  Type: Float.

  Units: Å.

  Status: Optional.

  Default value: **indentationSigma**.

Purpose: Repulsive distance for the cantilever tip Lennard-Jones potential σ_{tip} (see Eq. (3.7)). Will override **indentationSigma**.

- **indentationTipEl** <*energy factor of LJ interactions*>

  Type: Float.

  Units: kcal/mol.

  Status: Optional.

  Default value: **indentationEl**.

  Purpose: Repulsive energy factor ε_{tip} for the cantilever tip Lennard-Jones potential (see Eq. (3.7)). Will override **indentationEl**.

- **indentationTipZeta** < ζ *value for the cantilever tip*>

  Type: Float.

  Status: Optional.

  Default value: 5000.0.

  Purpose: Friction coefficient for the cantilever tip in viscous environment (see Eq. (3.8) and also section *Units*).

- **indentationFixTrans** <*yes/no*>

  Type: Boolean.

  Status: Optional.

  Default value:

  Purpose: Define if movement of the cantilever tip should be constrained for movement just along the indentation direction. All the transversal motions will be suppressed.

- **indentationCantLength** <*distance*>

  Type: Float.

  Units: Å.

  Status: Optional.

  Default value: 500.0 Å.

  Purpose: Length of the cantilever for its representation. Makes any difference only if **indentationShowTipSurf** is enabled.

- **indentationDiscreteSurf** <*yes/no*>

  Type: Boolean.

  Status: Optional.

  Default value: no.

  Purpose: If enabled, substrate surface will be represented as a set of interacting beads, positioned according to the surface representation (parameters **indentationSurfaceSize** and **indentationSurfaceStep**). Otherwise, potential will be continuous (the function fill be computed using the normal vector).

- **indentationSurfaceR0** <*position vector*>

  Type: Vector.

  Units: Å.

Status: Required.

Purpose: Position of the substrate surface.

- **indentationSurfaceN** <*direction vector x, y, z*>

  Type: Vector.

  Status: Required.

  Purpose: Substrate surface normal vector.

- **indentationSurfA / indentationSurfB** <*dimentionless constants*>

  Type: Float.

  Status: Optional.

  Default value: 0 and 1, respectively.

  Purpose: Shape of the Lennard-Jones potential for the substrate surface A_{surf} and B_{surf}, same as in Eq. (3.7) for the cantilever tip (see Section *Force indentation simulations*).

- **indentationSurfSigma** <*range of LJ interactions*>

  Type: Float.

  Units: Å.

  Status: Optional.

  Default value: **indentationSigma**.

  Purpose: Repulsive distance for the surface Lennard-Jones potential σ_{surf}. Will override **indentationSigma**.

- **indentationSurfEl** <*energy factor of LJ interactions*>

  Type: Float.

  Units: kcal/mol.

  Status: Optional.

  Default value: **indentationEl**.

  Purpose: Repulsive energy factor ε_{surf} for the surface Lennard-Jones potential. Will override **indentationEl**.

- **indentationSurfaceSize** <*number of points*>

  Type: Integer.

  Status: Optional.

  Default value: 51.

  Purpose: Number of points in length to represent square substrate surface. Total number of points saved will be a square value of this.

- **indentationSurfaceSizeX / indentationSurfaceSizeY** <*number of points*>

  Type: Integer.

  Status: Optional.

  Default value: 51 and 51.

  Purpose: Number of points in length/width to represent rectangular substrate surface. Total number of points saved will be equal to **indentationSurfaceSizeX** $\times$ **indentationSurfaceSizeY**.

- **indentationSurfaceStep** <*distance*>

  Type: Float.

  Units: Å.

  Status: Optional.

  Default value: 10 Å.

  Purpose: Distance between points representing substrate surface.

- **indentationMoveSurface** <*yes/no*>

  Type: Boolean.

  Status: Optional.

  Default value: no.

  Purpose: Define whether the substrate surface will be moving along Artem direction, rather than cantilever.

- **indentationSurfConnectFile** <*filename*>

  Type: Path to the file.

  Format: .vmd

  Status: Optional.

  Default value: connect_mica.vmd.

  Purpose: Filename of a dump "connect" script that can be used in VMD to show the mica as a surface rather than set of points.

- **indentationPairsCutoff** <*distance value*>

  Type: Float.

  Units: Å.

  Status: Optional.

  Default value: 40.0 Å.

  Purpose: Cut-off distance for the pairs list if the surface is represented as a set of discreet beads.

- **indentationOutput** <*filename*>

  Type: Path to the file.

  Format: *.dat*

  Status: Optional.

  Default value: "indentation.<name>_<author><run>.dat".

  Purpose: Filename for indentation output file.

- **indentationOutputFreq** <*number of steps*>

  Type: Integer.

  Status: Optional.

  Default value: 1000.

  Purpose: Frequency of writing output of indentation process in the **indentationOutput** file and on the terminal screen.

- **indentationRetractionStep** <*number of a step*>

  Type: Integer.

  Status: Optional.

  Default value: -1.

  Purpose: If specified, direction of indentation will be reversed on this step.

6.10 Heating parameters

- **heating** <*on/off*>

  Type: Boolean.

  Status: Optional.

  Default value: off.

  Purpose: Switching on the heating regime with heating parameters.

- **initialT** <*initial temperature*>

  Type: Float.

  Units: kcal/mol.

  Status: Required.

  Purpose: Initial system temperature.

- **deltaT** <*temperature increment*>

  Type: Float.

  Units: kcal/mol.

  Status: Required.

  Purpose: Value of the temperature increment that will be added to the initial temperature every **tempFreq** steps.

- **tempFreq** <*number of steps*>

  Type: Integer.

  Status: Required.

  Purpose: Frequency of updating the temperature.

6.11 Output parameters

- **reffilename** <*filename*>

  Type: Path to the file.

  Format: *.pdb*.

  Status: Optional.

  Default value: “<name>.ref.pdb”.

Purpose: Name of the reference output file with the coordinated of modeled system as well as cantilever tip, base and substrate surface if **indentation** is "on". This can be used to load structure into VMD.

- **outputtiming** <*number of steps*>

  Type: Integer.

  Status: Optional.

  Default value: 10000.

  Purpose: Frequency of writing out energy output of simulation process (see Section *General output*}).

- **outputname** <*filename*>

  Type: Path to the file.

  Format: *.dat*.

  Status: Optional.

  Default value: "energy.<name>_<author><run>.dat".

  Purpose: Name of the output file to save resulted energy. If file exists, it will be overwritten.

- **outputcolwidth** <*number of characters*>

  Type: Integer.

  Status: Optional.

  Default value: 16.

  Purpose: Width of one column in output file, specified in amount of characters.

- **printruns** <*number of trajectories*>

  Type: Integer.

  Status: Optional.

  Default value: 10.

  Purpose: Number of trajectories for which output energies will be printed out in terminal screen when many-runs-per-GPU approach is utilized.

- **computeRg** <*yes/no*>

  Type: Boolean.

  Status: Optional.

  Default value: no.

  Purpose: Specified if program should calculate and print in output file radius of gyration of the modeled system.

- **R_limit_bond** <*cut-off distance*>

  Type: Float.

  Units: Å.

  Status: Optional.

  Default value: 8.0 Å.

  Purpose: Cut-off radius to calculate the number of survived native contacts in during simulation.

- **dcdfreq** <*number of steps*>

Type: Integer.

Status: Optional.

Default value: 10000.

Purpose: Frequency of writing out structure coordinates in .dcd output file in course of simulation.

- **DCDfile** <*filename*>

  Type: Path to dcd file.

  Status: Optional.

  Default value: "<name>_<author><run>.dcd".

  Purpose: Name of dcd file to write coordinates output in. If file exists, it will be overwritten.

- **restartfreq** <*number of steps*>

  Type: Integer.

  Status: Optional.

  Default value: 100000.

  Purpose: Frequency to save current structure coordinates in *.pdb* file.

- **restartname** <*filename*>

  Type: Path to the file.

  Format: *.pdb*.

  Status: Optional.

  Default value: "<name>_<author><run>_restart".

  Purpose: Extensionless name of the restart files. Only particle coordinates are saved.

- **finalcoord** <*filename*>

  Type: Path to the file.

  Format: *.pdb*.

  Status: Optional.

  Default value: "<name>_<author><run>_final.pdb".

  Purpose: Filename for the final coordinates.

Examples of SOP-GPU configurational files can be found here.

Bibliography

[Hyeon2006] C. Hyeon, R. I. Dima, and D. Thirumalai (2006) "Pathways and kinetic barriers in mechanical unfolding and refolding of RNA and proteins", *Structure* **14** (11): 1633-1645.

[Mickler2007] M. Mickler, R. I. Dima, H. Dietz, C. Hyeon, D. Thirumalai, and M. Rief (2007) "Revealing the bifurcation in the unfolding pathways of GFP using single molecule experiments and simulations", *Proc. Natl. Acad. Sci. USA* **104** (51): 20268–20273.

[Zhmurov2010] A. Zhmurov, R. I. Dima, and V. Barsegov (2010) "Order statistics theory of unfolding of multimeric proteins", *Biophys. J.* **99**: 1959.

[Kononova2013a] O. Kononova, L. Jones, and V. Barsegov (2013) "Order statistics inference for describing topological coupling and mechanical symmetry breaking in multidomain proteins", *J. Chem. Phys.* **139** (12): 121913.

[Ermak1978] D. Ermak and J. A. McCammon (1978) "Brownian dynamics with hydrodynamic interactions", *J. Chem. Phys.* **69** (4): 1352.

[Rotne1969] J. Rotne and S. Prager (1969) "Variational Treatment of Hydrodynamic Interaction in Polymers", *J. Chem. Phys.* **50** (11): 4831-4837.

[Yamakawa1970] H. Yamakawa (1970) "Transport Properties of Polymer Chains in Dilute Solution: Hydrodynamic Interaction", *J. Chem. Phys.* **53** (1): 436-443.

[Geyer2009] T. Geyer and U. Winter (2009) "An $O(N^2)$ approximation for hydrodynamic interactions in Brownian dynamics simulations", *J. Chem. Phys.* **130** : 114905.

[Cieplak2009] M. Cieplak and S. Niewieczerzal (2009) "Hydrodynamic interactions in protein folding", *J. Chem. Phys.* **130** : 124906.

[Frembgen-Kesner2009] T. Frembgen-Kesner and A. H. Elcock (2009) "Striking Effects of Hydrodynamic Interactions on the Simulated Diffusion and Folding of Proteins", *J. Chem. Theory. Comput.* **5** : 242-256.

[Zhmurov2011] A. Zhmurov, A. E. X. Brown, R. I. Litvinov, R. I. Dima, J. W. Weisel, and V. Barsegov (2011) "Mechanism of fibrin(ogen) forced unfolding", *Structure* **19** (11): 1615-1624.

[Kononova2013b] O. Kononova, J. Snijder, M. Brasch, J. Cornelissen, R. I. Dima, K. A. Marx, G. J. L. Wuite, W. H. Roos, and V. Barsegov (2013) "Structural transitions and energy landscape for cowpea chlorotic mottle virus capsid mechanics from nanomanipulation *in vitro* and *in silico*", *Biophys. J.* **105** (8): 1893-1903.

[Kononova2014] O. Kononova, Y. Kholodov, K. E. Theisen, K. A. Marx, R. I. Dima, F. I. Ataullakhanov, E. L. Grishchuk, and V. Barsegov (2014) "Tubulin bond energies and microtubule biomechanics determined from nanoindentation *in silico*", *J. Am. Chem. Soc.* **136** (49): 17036-17045.

[Brooks1983] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan and M. Karplus (1983) "CHARMM: A program for macromolecular energy, minimization, and dynamics calculations", *J. Comput. Chem.* **4**: 187-217.

[Haberthur2008] U. Haberthür, A. Caflisch (2008) "FACTS: Fast analytical continuum treatment of solvation", *J. Comput. Chem.* **29**: 701-715.

[Zhmurov2010b] A. Zhmurov, R. I. Dima, Y. Kholodov and V. Barsegov (2010) "SOP-GPU: Accelerating biomolecular simulations in the centisecond timescale using graphics processors", *Proteins* **78**: 2984-2999.

[Press1992] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery. "Numerical Recipes in C", 2nd ed. *The Art of Scientific Computing*, Cambridge University Press, 1992.

[LEcuyer2007] P. L'Ecuyer and R. Simard (2007) "TestU01: A C library for empirical testing of random number generators", *ACM T. Math. Software.* **33**: 22.

[Marsaglia1996] G. Marsaglia (1996) "DIEHARD: A battery of tests of Randomness" (http://stat.fsu.edu/geo/diehard.html).

[Mascagni2000] M. Mascagni and A. Srinivasan (2000) "Algorithm 806: SPRNG: A scalable library for pseudorandom number generation", *ACM T. Math. Software.* **26**: 436-461.

[Soto1999] J. Soto (1999) "Statistical testing of random number generators" (http://csrc.nist.gov/rng/).

[Zhmurov2011b] A. Zhmurov, K. Rybnikov, Y. Kholodov and V. Barsegov (2011) "Generation of random numbers on graphics processors: Forced indentation *in silico* of the bacteriophage *HK97*", *J. Phys. Chem. B* **115**: 5278-5288.

[Tsang2000] W. W. Tsang and G. Marsaglia (2000) "The Ziggurat Method for Generating Random Variables", *J. Stat. Softw.* **5**.

[Marsaglia1964] G. Marsaglia and T. A. Bray (1964) "A convenient method for generating normal variables", *SIAM Rev.* **6**: 260-264.

[Box1958] G. E. P. Box and M. E. Mueller (1958) "A note on the generation of normal random deviates", *Ann. Math. Stat.* **29**: 610-611.

[Mascagni2004] M. Mascagni and A. Srinivasan (2004) "Parameterizing parallel multiplicative lagged Fibonacci generators", *Parallel Comput.* **30**: 899-916.

[Tausworthe1965] R. C. Tausworthe (1965) "Random numbers generated by linear recurrence modulo two", *Math. Comput.* **19**: 201-209.