RESEARCH ARTICLE | NOVEMBER 29 2022

Complexes++: Efficient and versatile coarse-grained simulations of protein complexes and their dense solutions

Max Linke; Patrick K. Quoika ; Berenger Bramas ; Jürgen Köfinger ; Gerhard Hummer

J. Chem. Phys. 157, 204802 (2022)
https://doi.org/10.1063/5.0117520

Complexes++: Efficient and versatile coarse-grained simulations of protein complexes and their dense solutions

Cite as: J. Chem. Phys. **157**, 204802 (2022); doi: 10.1063/5.0117520
Submitted: 1 August 2022 • Accepted: 7 November 2022 •
Published Online: 29 November 2022

View Online Export Citation CrossMark

Max Linke,[1] Patrick K. Quoika,[1,2] Berenger Bramas,[3,4,5] Jürgen Köfinger,[1,a)] and Gerhard Hummer[1,6,b)]

AFFILIATIONS

[1] Department of Theoretical Biophysics, Max Planck Institute of Biophysics, Max-von-Laue-Str. 3, 60438 Frankfurt am Main, Germany
[2] Center for Protein Assemblies (CPA), Physics Department, Chair of Theoretical Biophysics (T38), Technical University of Munich, Ernst-Otto-Fischer-Str. 8, 85748 Garching, Germany
[3] Max Planck Computing and Data Facility, Gießenbachstraße 2, D-85748 Garching, Germany
[4] Inria Nancy - Grand Est, 615 Rue du Jardin-Botanique, 54600 Villers-lès-Nancy, France
[5] ICube, 300 bd Sébastien Brant, 67412 Illkirch, France
[6] Department of Physics, Goethe University Frankfurt, Max-von-Laue-Str. 1, D-60438 Frankfurt am Main, Germany

[a)] **Author to whom correspondence should be addressed:** juergen.koefinger@biophys.mpg.de
[b)] **Electronic mail:** gerhard.hummer@biophys.mpg.de

ABSTRACT

The interior of living cells is densely filled with proteins and their complexes, which perform multitudes of biological functions. We use coarse-grained simulations to reach the system sizes and time scales needed to study protein complexes and their dense solutions and to interpret experiments. To take full advantage of coarse-graining, the models have to be efficiently implemented in simulation engines that are easy to use, modify, and extend. Here, we introduce the Complexes++ simulation software to simulate a residue-level coarse-grained model for proteins and their complexes, applying a Markov chain Monte Carlo engine to sample configurations. We designed a parallelization scheme for the energy evaluation capable of simulating both dilute and dense systems efficiently. Additionally, we designed the software toolbox *pycomplexes* to easily set up complex topologies of multi-protein complexes and their solutions in different thermodynamic ensembles and in replica-exchange simulations, to grow flexible polypeptide structures connecting ordered protein domains, and to automatically visualize structural ensembles. Complexes++ simulations can easily be modified and they can be used for efficient explorations of different simulation systems and settings. Thus, the Complexes++ software is well suited for the integration of experimental data and for method development.

I. INTRODUCTION

Biomolecular simulations provide insight into the molecular structures and interactions underlying biological functions.[1] With limits on the available computational power,[2] one way to extend the length scales, time scales, or sampling needed to study more complex biomolecular systems is through coarse-grained simulation models.[2–6] In these models, multiple atoms are typically grouped into a single interaction site or bead. This reduction in the number of degrees of freedom and model complexity lowers the computational cost of evaluating energies and forces. The corresponding smooth potential energy surfaces speed up particle dynamics compared to fine-grained models. The overall simplicity of coarse-grained models also makes them easier to analyze and modify. However, this increase in computational efficiency and modifiability usually comes with a decrease in accuracy. The latter can be improved, for example, by using force fields with multi-body interactions for the same resolution level.[7]

To take full advantage of coarse-grained simulations and improve their predictive power, simulation engines must be highly efficient and easy to use, modify, and extend. Importantly, coarse-grained simulations can be more easily used to explore different simulation systems and settings than, for example, atomistic simulations.[8,9] The computational and memory efficiencies are increased, and we can run multiple simulations in rapid succession and in parallel, even on a single machine. Thus, coarse-grained models are ideally suited to screen vast parameter spaces by running multiple simulations. One can modify and study system properties such as stoichiometry, chemical composition, thermodynamic states, chemical functionalization, and initial structures. The possibility to try different settings makes coarse-grained simulations especially well suited for the integration of additional data, as in inferential structure determination[10] and ensemble refinement.[11,12] Moreover, the reduced number of degrees of freedom and the model simplicity allow us to more easily bias the underlying energy surfaces or force fields using enhanced sampling techniques.

Here, we present an efficient open-source implementation of the KH model,[13] which combines knowledge-based information about amino acid interactions[14] with physical interaction laws to model protein complexes[15] and even intrinsically disordered proteins.[16] The KH model was developed for protein complexes containing ordered protein domains, possibly connected by flexible and disordered polytpeptides. Such systems are common in biological systems. For example, the KH model has been applied to infer the dynamic solution ensembles of the ESCRT–I complex,[17] the Atg1 complex,[15] and a module of a polyketide megasynthase.[18] To tackle such systems, where we often have only limited or coarse initial structural information, the simulation software has to be computationally efficient and easy to use to be able to explore systems and settings efficiently. In the development of our simulation software, we addressed these two main challenges of efficiency and usability.

We optimized computational and memory efficiency at different levels of our software. For example, we developed and implemented a parallelization scheme designed to handle diverse systems ranging from dense to dilute protein solutions. Moreover, the simulation of protein complexes, where ordered protein domains modeled as rigid bodies are linked by disordered polypeptides, is computationally highly expensive. The presence of explicitly modeled peptide linkers between ordered protein domains hinders their diffusion and is detrimental to sampling. We thus model disordered linkers as harmonic restraining potentials in simulations and regrow explicit linker structures in post-processing.

To make our software easy to use, modify, and extend, we followed a twofold strategy. We developed an efficient and versatile simulation engine, Complexes++, using C++17.[19] Additionally, we provide the *pycomplexes* toolbox to set up, modify, process, and visualize Complexes++ simulations. It enables the user to, for example, prepare coarse-grained simulations from atomistic structure PDB-files from the Protein Data Bank (https://rcsb.org)[20] or efficiently add explicit polypeptide linker configurations by post-processing sampled simulation frames. The versatility and extendability of the Complexes++ simulation engine and the *pycomplexes* toolbox facilitate the systematic and comprehensive study of complex biomolecular systems where, for example, only limited initial structural information is available or the interplay of a large number of protein domains is of interest.

This article is organized as follows: We first introduce the Complexes++ simulation engine, its architecture, and its functionality and present the toolbox *pycomplexes* in Sec. II. We present method details in Sec. III and performance results in Sec. IV. We end with a discussion and conclusions in Sec. V.

II. THE COMPLEXES++ SOFTWARE

We first give a brief overview of the Complexes++ simulation engine. Then, we discuss implementation details and afterward the *pycomplexes* toolbox.

A. Overview

In the KH model, one represents amino acids of proteins with single interaction beads centered at the positions of C_α atoms.[13] In this model, the knowledge-based Miyazawa–Jernigan potentials[14] acting between beads are complemented by electrostatic interactions approximated in terms of the Debye–Hückel potential for an implicit solvent with different salt concentrations. Folded proteins and multi-protein complexes are represented as rigid bodies. These rigid domains can be connected by disordered polypeptides, modeled as flexible polymer chains, or as effective potentials. We call the full set of connected domains a topology. Potential energies are evaluated at the bead level, and position updates are done at the domain and topology levels. In addition to the Lennard–Jones-like potential of the KH model, we also implemented the Weeks–Chandler–Anderson (WCA) repulsive and attractive potentials,[21] a softcore potential,[22] and interactions of Gaussian charge distributions.[23]

To generate configurations according to the underlying statistical ensemble, Complexes++ uses a Markov chain Monte Carlo algorithm.[24] We implemented the canonical ensemble (NVT) and the NPT ensemble, where P is here the osmotic pressure. For enhanced sampling, Complexes++ provides different replica-exchange algorithms, such as temperature replica exchange,[25] pressure replica exchange,[26] and Hamiltonian replica exchange.[27] As an exchange scheme, we use the odd–even pair scheme.[27]

For the rigid domains, we implemented random translation and rotation Monte Carlo moves. Each component of a trial translation vector is chosen uniformly from a user-defined interval. The rotations are generated by choosing a random rotation axis and uniformly selecting a rotation angle in a user-defined interval.[28] We define the direction of the rotation axis by uniformly selecting a point in the unit cube. Even though rotation axes along the edges are more likely, they do not affect the generated ensembles due to detailed balance. Translation and rotation moves are selected with equal probability.

We model disordered polypeptides connecting rigid domains using a Gaussian polymer model during simulations and grow explicit linker configurations for sampled frames in post-processing. In the original simulation code,[13] disordered polypeptides connecting rigid domains could also be simulated using a bead-chain model using bond, angle, and dihedral potentials similar to molecular dynamics force fields.[29] The advantage of this model is that amino acids are modeled explicitly as beads, which can interact with the rigid domains. However, these interactions are difficult to model accurately. Moreover, configurations of explicit rigid domains linked by disordered polypeptides are hard to sample, especially with single particle Monte Carlo moves. As a result, the chain diffuses slowly through configuration space and strongly restrains the diffusive motion of attached domains. While this explicit model can be applied to smaller protein complexes,[30] it makes simulations of larger complexes[15,18] inefficient.

To efficiently sample the configuration space of protein domains connected by flexible polypetide linkers, we use a linker model that avoids entropic bottlenecks in the sampling and constrains protein movement the least. For a polypetide linker, its potential mean force (PMF) is mainly determined by its length and not by its amino acid sequence.[31] Thus, we assume that the main function of the linker is to restrain two connected domains accordingly. We replace the explicit peptide chain with a PMF that only depends on the distance between the two rigid domains and the number of amino acids in the linker, i.e., its length. This PMF acts as a restraining potential and ensures that the distance distribution between two rigid domains is physically meaningful. The diffusion of the rigid domains is least hindered by this linker model because no explicit linker beads are involved.

In Complexes++, we use the PMF of the Gaussian chain polymer model.[32,33] Our PMF acts between two beads, one on each of the two connected domains. These two beads have to be added to the length N of the linker. The final PMF for a linker of length N is

$$\mathrm{PMF}(\vec{r}_0, \vec{r}_{N+1}) = \frac{3}{2b^2}\frac{1}{N+1}(\vec{r}_0 - \vec{r}_{N+1})^2, \tag{1}$$

with $\vec{r}_0$ and $\vec{r}_{N+1}$ being the positions of the two beads of the rigid domains that are connected by the linker. Beads on the linker are separated by an average bond length b such that the average end-to-end distance of the linker is $\sqrt{N}b$. In the following, we use $b = 3.81$ Å.[34] After the simulation, we add explicit linker configurations to the sampled structures using our *pycomplexes* toolbox (see Sec. II C).

B. Complexes++ architecture and functionalities

We designed the Complexes++ software to give the modeler freedom when setting up simulations. To increase flexibility, domain classes are defined on-the-fly during runtime using domain templates. A domain is composed of move types, their parameters, and interaction kernels with other domains. Move types and interaction kernels are implemented separately. Domains have unique names. For each simulation, one can specify the exact domains, set their Monte Carlo move parameters, and define interactions with the rest of the system.

Complexes++ is composed of several abstraction layers (Fig. 1). These abstraction layers are based on the main components of the Monte Carlo algorithm and ensure a high degree of modularity.

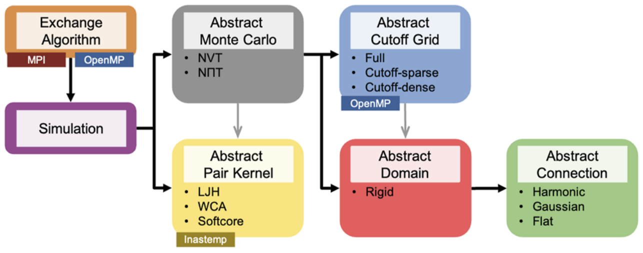

FIG. 1. Architecture of the Complexes++ simulation software. In colored boxes, we display different classes (rectangular boxes). Some of these classes are abstract classes, which serve as templates. Derived classes are shown as bullet points within the same colored box. As indicated in the small rectangular boxes, some of these classes are parallelized with Message Passing Interface (MPI) or OpenMP, or use the vectorization library Inastemp[36] for improved performance. Black arrows with full arrow heads indicate that a class (start of the arrow) owns another class object (full arrow head). Gray arrows indicate that a class (start of the arrow) contains a smart pointer to another class (empty arrow head). These pointers are needed such that the class holding the pointer can automatically determine if the other class has changed. The interaction kernel "LJH" includes Lennard–Jones ("LJ" in "LJH") and Debye–Hückel ("H" in "LJH") potentials. "WCA" stands for Weeks–Chandler–Andersen.[21] The "softcore" interaction kernel comprises a softcore Lennard–Jones potential[22] and interactions of Gaussian charge distributions.[23]

This design makes it possible to add new components with only minimal changes. One can add a new domain type or a new type of Monte Carlo sweep in a few implementation steps. The abstraction layers are implemented as abstract classes in C++ and use the template method pattern,[35] which is language-independent. For example, the abstract Monte Carlo class uses the template method pattern for the sweep operation that each sub-class must have. As a result, Complexes++ can be used not only to perform simulations of the KH model but also to develop and test other molecular models.

Complexes++ is parallelized using MPI and OpenMP. MPI is optional and can be disabled during the compilation stage. In replica-exchange simulations using MPI, the replicas are statically distributed to the processes, and there can be at most as many processes as there are replicas. We avoid global communication and synchronization by using a point-to-point exchange scheme between processes when replicas might be exchanged. The MPI-related code is localized in a single file and is transparent to the rest of the application. For instance, any class that could be exchanged together with a replica only has to provide serialize/deserialize methods. We use the same principle for shared-memory parallelization, which is done over the different replicas such that one replica is managed by a single thread for a given number of sweeps.

In replica simulations using only OpenMP, a parallel loop is performed over the replicas, leading to at most one thread per replica but with potentially several replicas for some of the threads. In order to manage the cases where the number of available threads is greater than the number of replicas still to be processed or the workload per replica is highly heterogeneous, we use a pool of tasks to dynamically share the work and adapt the execution. However, the tasks are created only when at least one thread becomes idle, which happens immediately when there are fewer replicas than threads or after a delay when there is no more replica to assign to a thread. After this point, all threads that were computing their own replica alone begin to create tasks. With this approach, we minimize the possible overhead coming from the task-based mechanism itself or from the division of the replicas. When all replicas have progressed and performed their sweeps, a point-to-point synchronization between the threads is used to test and potentially exchange some replicas.

We vectorize the interaction kernels using the Inastemp library,[36] which was originally created specifically for Complexes++. Since then, it has been used in multiple other applications. One example is ScalFMM,[37] which implements a fast multipole method. Applying Inastemp, the kernels are written in C++ using a hardware-independent data type. During the compilation stage, this data type is replaced with an Inastemp class selected by the compiler according to the CPU's capabilities. Inastemp supports common instruction sets (SSE{3/4.1}, AVX{1,2-512}, and VMX) and ensures that even kernels with condition statements are vectorized. This mechanism ensures pre-compilation portability and high performance while being easy to maintain.

We developed and implemented a variant of the cell-list algorithm to efficiently evaluate the interaction energies. We adapted the algorithm for sparse as well as dense systems, respectively. The interaction kernels used in Complexes++ are generally short-ranged, such that one can apply a cutoff distance. In the original cell-list method,[28] we partition the simulation box into a 3D cell grid and compute the interactions between beads in neighboring cells only.

In our version of the cell-list algorithm, we work on the beads and domains in different ways to compute the global energy and the interaction energy of a single domain with all the others, and to move a single domain and update the grid accordingly. To do so efficiently, we do not store the beads directly in the cells. Instead, we use the notion of intervals, where an interval is composed of a domain identification number (id), a starting bead index, and an ending bead index. We store beads contiguously in memory, no matter their distribution in the grid. One can then efficiently access all the bead coordinates of a domain for further computations. Moreover, for each domain, we use a list of interval-index/cell-id pairs, which allows us to iterate over the cells crossed by a domain and to compute the energy related to a single domain efficiently. Finally, the update of the grid when a domain has moved can be done efficiently, as it consists of updating the cells' interval vectors by removing the old intervals and inserting the new ones. The overall computational complexity of these different stages is linear with respect to the numbers of beads, intervals, and domains.

For this cell-list algorithm, we provide two different data structures or containers to store the cell-grid, depending on the sparsity and size of the system. The first container, called "dense data structure" in the following, is a 3D array for dense molecular systems. The second container, called "sparse data structure," is an unordered hash-map with the linearized 3D indices as keys for sparse or dilute molecular systems. Whereas the memory consumption of the former increases with the number of cells and thus with the volume for a given cell size, the memory consumption of the latter is independent of the number of cells. The sparse data structure enables us to run multiple replicas on a single node, where memory can become the limiting factor. Such situations arise, for example, when running replica-exchange simulations. For both data structures, the time complexity of the cell-list algorithm scales linearly with the number of beads. See the user manual for further details on our implementation of the cell-list algorithm.

C. Pycomplexes: A helper toolbox

The *pycomplexes* toolbox provides diverse functionality to efficiently prepare, modify, process, visualize, and analyze simulations. It was designed for convenience and ease-of-use. *Pycomplexes* is written in Python and harnesses widely used libraries, such as Numpy[38] and MDAnalysis.[39] Thus, it is cross-platform compatible and easily extensible. It may be used from the command line, but is also easily integrated in interactive Python environments such as Jupyter notebooks.[40] Along with the code, we provide numerous tutorials.

Pycomplexes may be regarded as a set of tools, as included, e.g., in the AMBER toolbox,[41] the GROMACS simulation suite,[42] or HOOMD-blue.[43] To start with, the user may use *pycomplexes* to generate coarse-grained structures and topologies from atomistic PDB files.[20] Complexes++ stores the full information needed to run simulations in our newly developed CPLX file format using YAML.[44] CPLX files are human-readable such that they can be manually modified. For example, we can set larger values of the bond length b in simulations to make the linkers more extended.[45] We can also set different average bond length values b for different linkers.

CPLX files contain the topologies, coordinates, and force fields and their parameters, making it easy to share complete simulation details. Such CPLX files can be set up efficiently using the "`pycomplexes convert`" command and our newly developed TOP files as input. The latter contain information about the topologies of the different complexes, i.e., about connectivities and their multiplicities. In these files, the user may specify an arbitrary number of copies of a certain protein domain of interest, for example. The copies are then automatically placed in the simulation box without further input from the user.

Additionally, *pycomplexes* can be used to update coordinates contained in the CPLX files or to adjust the topology according to the pH value, for example. It can automatically create Visual Molecular Dynamics (VMD)[46] input scripts for visualization, generate demuxed Monte Carlo sampling series from replica exchange simulations, which correspond to time-continuous trajectories in molecular dynamics simulations, and add linker positions in post-processing (see below). Furthermore, *pycomplexes* may be used to shift and scale the Miyazawa–Jernigan force field parameters.[14] Hence, the coarse-grained force field may be tuned according to experimental reference data while conserving the relative residue–residue affinity.[13,47] Such fine-tuning of the force field is an essential feature of coarse-grained simulations and has been applied to Martini simulations,[48,49] for example.

We construct explicit linker configurations by post-processing trajectories in two steps. First, we grow linker bead positions between connected domains according to the Gaussian chain model. However, the distribution of bead distances is non-physical. Thus, we equilibrate the Gaussian chains using the original linker model[13]

TABLE I. Protein structures for benchmarks. *N* is the number of amino acids, and *D* is an estimate for the largest extension.

Protein (complex)	PDB ID	N	D (Å)
Lysozyme	6LYZ[50]	129	44
Ras G12V-PI 3-kinase gamma	1HE8[51]	749	96
Atg17-Atg31-Atg29	4HPQ[52]	1282	334

with an average bond length $b = 3.81$ Å. We explain these two steps in more detail in Appendix A.

III. METHODS

To assess the performance of Complexes++, we ran dense protein simulations with three different proteins and protein complexes (Table I). The structures differ in size, from small and compact to large and elongated. We use the original KH model for the interaction potentials. For all benchmarks, if not further specified, the protein volume fraction is set to 0.1. To calculate this volume fraction, we assign each protein a volume given by the volume of a sphere with a diameter equal to the protein's maximum extension (see Table I), for which we use the dense data structure for the cell-list algorithm. The cell size is set to 12 Å, the number

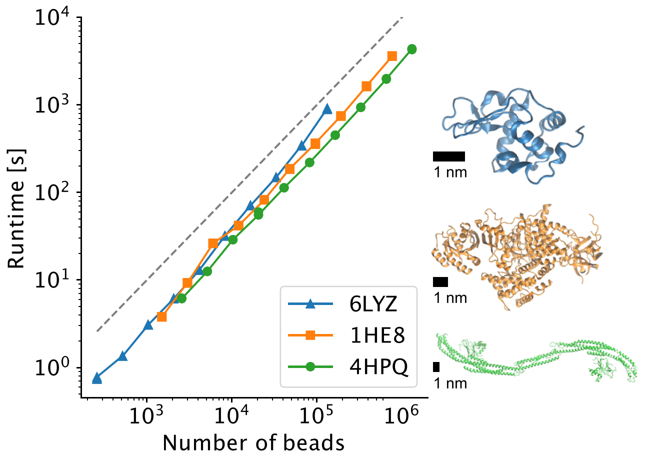

FIG. 2. Scaling of the runtime of Complexes++ simulations for the solutions of the three different proteins (colored symbols with lines as a guide to the eye, see Table I) with the number of beads. We show images of the PDB structures of the three proteins (same colors as symbols) on the right with scale bars of 1 nm length, emphasizing their different shapes and sizes. Protein numbers were varied between 2 and 512 in steps of the power of 2 (blue triangles-6LYZ, orange squares-1HE8, green disks-4HPQ). The slope of the dashed gray line indicates ideal linear scaling. All simulations were performed using a single thread. Note the double-logarithmic scale.

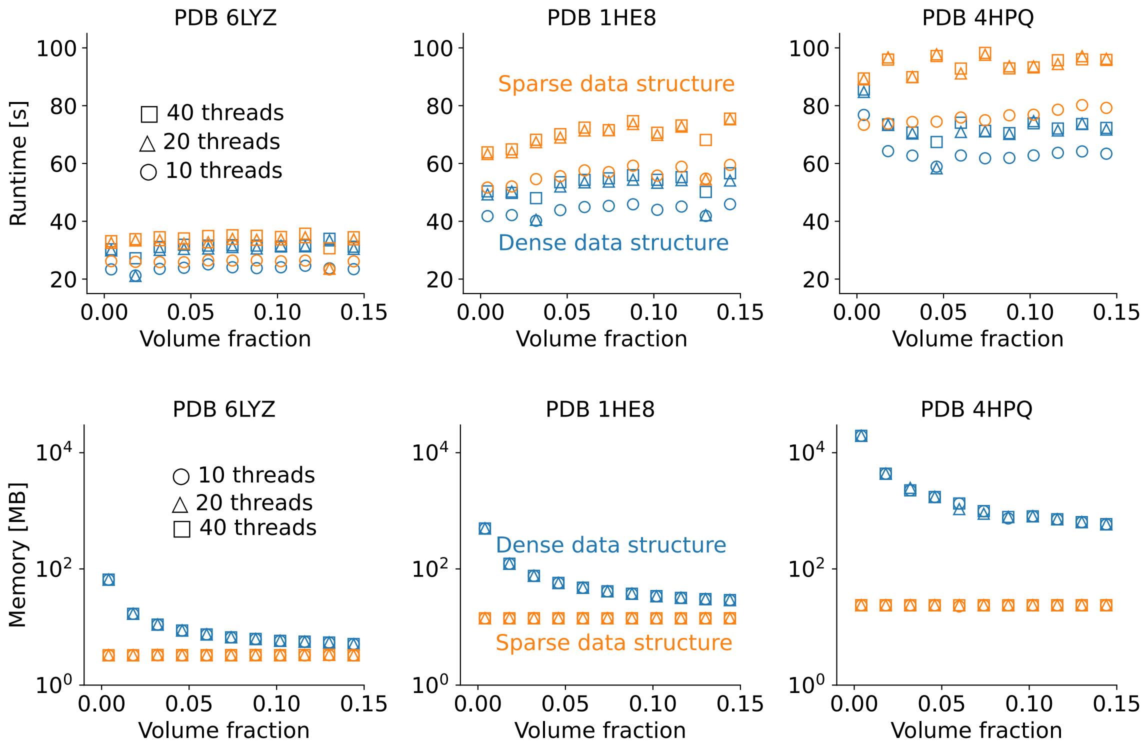

FIG. 3. Comparison of the two cell-list data structures designed to be computationally efficient (dense data structure-blue) or memory efficient (sparse data structure-orange) systems. The data structure for dense systems is computationally more efficient than the algorithm for sparse systems (top). The memory consumption of the data structure for dense systems is orders of magnitude larger than for sparse systems at low densities (bottom). The memory consumption of the latter is independent of the density and always lower than that of the dense data structure. For large enough densities, both data structures will consume the same amount of memory. Symbols indicate different numbers of threads.

of sweeps is 5000, and the temperature is set to 300 K. For the implicit Debye–Hückel solvent, we set the Debye length to 10 Å and the dielectric constant is 80, corresponding to physiological salt conditions.

Single-replica and MPI benchmarks were performed on the COBRA cluster of the Max Planck Computing and Data Facility (MPCDF). A node consists of two Intel Xeon "Skylake" processors with 20 cores @ 2.4 GHz per processor and 96–192 GB of memory. Cell-list algorithm benchmarks were performed on the DRACO cluster of the MPCDF. A node consists of two Intel Xeon E5-2698 processors with 16 cores @ 2.3 GHz per processor and 128 GB of memory.

IV. RESULTS

We first confirm that our cell-list algorithm using bead intervals scales linearly with increasing number of beads using a single thread (Fig. 2). To this end, we performed simulations of identical proteins, which were modeled as rigid domains. The number of protein copies varied between 2 and 512. The simulation of the largest protein with PDB code 4HPQ contained 656 384 beads. For each of the three proteins, the runtime increases nearly linearly with the number of protein copies in the simulation. Moreover, the runtime is independent of the domain size and depends only on the total number of beads.

We also compared the two different cell-list data structures designed for dense and sparse molecular systems (Fig. 3). We vary the protein volume fraction for a given protein copy number of 128 by changing the simulation box volume. We find that the dense data structure is generally computationally more efficient (Fig. 3, top). However, the memory consumption of the dense data structure increases linearly by orders of magnitude with increasing box volume for a given cell size. Correspondingly, memory consumption depends inversely on the density (Fig. 3, bottom). The memory consumption of the sparse data structure remains constant over the full density range and depends linearly on increasing protein size, i.e., bead number (Fig. 3 bottom; increase in plateau from left to right). The sparse data structure thus allows us to use the cell-list algorithm when computer memory is limited. Such situations occur when running multiple replicas on a single node.

We tested the dependence of the performance of Complexes++ for a single replica and a single node on the chosen number of threads (see Fig. 4). We varied the copy numbers for each of the three

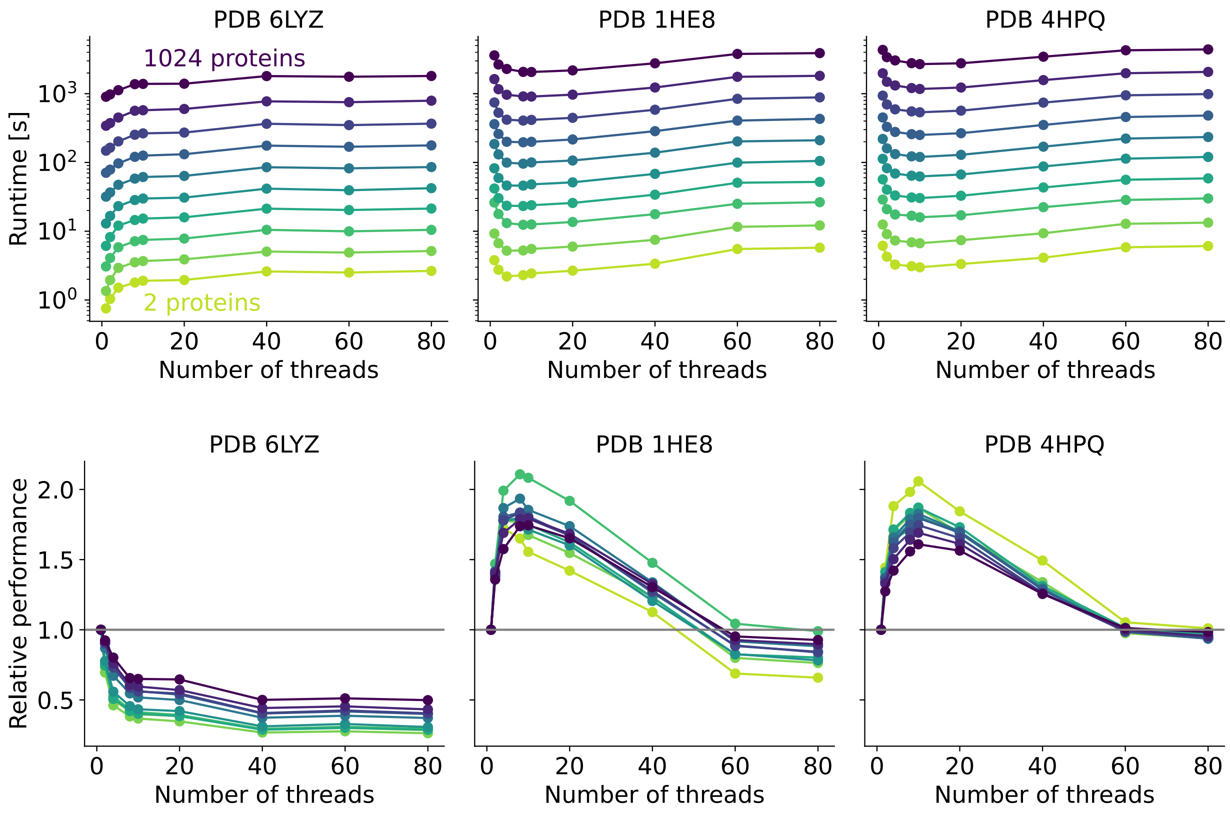

FIG. 4. Runtime of single replica simulations using an increasing number of threads for three solutions of proteins (see Table I) with protein copy numbers from 2 to 1024 in powers of 2. The shape of the curves of the runtime (top) is similar for all copy numbers. For the two largest systems, 1HE8 and 4HPQ, the runtimes show single minima for eight threads. For the smallest systems, 6LYZ, the runtime is the smallest for a single thread. To quantify this behavior, we show the relative performance with respect to a single thread (bottom). It is defined by the runtime for a single thread divided by the runtime for the given number of threads.

proteins from 2 to 1024 in steps of the power of 2. For all copy numbers of a protein, the dependence of the runtime on the number of threads is similar in shape (Fig. 4, top). For the two larger systems, 1HE8 and 4HPQ, the optimal runtime is reached with eight threads. There, the runtime as a function of the number of threads shows a clear minimum. For the smallest system, 6LYZ, the optimal number of threads is one. The runtime increases with increasing numbers of threads and converges to a plateau.

We next illustrate the behavior of the runtime for different protein copy numbers by defining the relative performance as a function of the number of threads. The relative performance is given by the runtime for a single thread divided by the runtime for the given number of threads (Fig. 4, bottom). For the two larger systems, the relative performance has a clear peak at about eight threads. There, the relative performance is nearly twice as good as for a single thread, or equally that the time to solution is only half. The highest throughput is reached for a single thread. For the smallest system here, performance peaks for a single thread. For this setting, time to solution is the shortest and throughput is the largest.

For replica-exchange simulations, the MPI parallelization can make use of multiple nodes for different replicas and multiple threads per replica (Fig. 5). For different numbers of replicas of protein solutions of 1HE8 with 1024 copies, we used several MPI/thread configurations and different numbers of nodes. Each replica is managed by a single MPI process, such that if we have more replicas than MPI processes, some MPI processes will have to process more than one replica. Then, each process uses one or several threads, as is done with the pure OpenMP implementation. We chose the number of MPI processes per node and the number of threads per process such that the CPU cores are fully used. That is the number of MPI processes per node times the number of threads per process is equal to the number of physical cores per node. The latter is 40 on the compute cluster used for benchmarking. We used one, two, four, and eight threads per MPI process.

We find that the runtime decreases with increasing node numbers in all cases. For the same node number, single threads per replica are generally more efficient. However, when using more threads per replica/process, we use more nodes and thus decrease the runtime further. The lowest overall runtime is reached with eight threads per process and the largest numbers of used nodes. With these settings, we reach a similar runtime for the three different numbers of replicas.

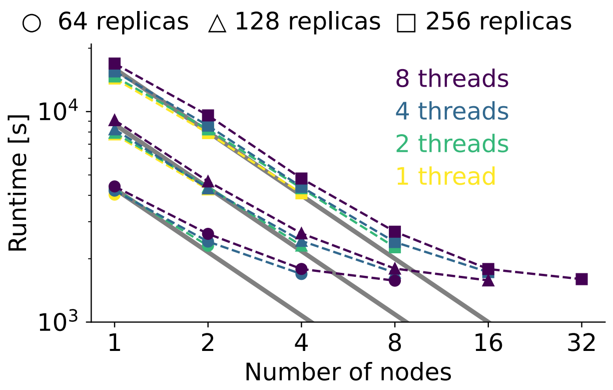

FIG. 5. Total runtime of MPI replica exchange simulations of 1HE8 for a varying number of nodes with 1024 protein copies per replica. For a given number of nodes, we varied the numbers of threads per MPI process (one—magenta, two—blue, four—green, eight—yellow) and the numbers of replicas (64—circle, 128—triangle, 256—square). The thick gray lines show ideal scaling, proportional to the inverse of the number of nodes, as a guide to the eye. The total number of MPI processes does not exceed the total number of replicas. Additionally, we choose the number of MPI processes to use all available cores. Thus, different combinations of numbers of nodes and threads are realized for different numbers of replicas.

V. DISCUSSION AND CONCLUSIONS

Complexes++ was developed for simulations of protein systems with the KH model and KH-like models. The KH model and its variations have a wide range of different applications, which warrants the development of highly efficient and versatile simulation software. Applications include, for example, the study of the binding kinetics of the HIV-1 capsid proteins,[53,54] the kinetic behavior of proteins in crowded environments,[16,55–57] inferential structure determination,[15,30,58–60] protein–protein interactions,[26,47,61,62] protein design,[63] docking,[64] initial structure generation for atomistic simulations of dense protein solutions,[8] epitope map generation for the SARS-CoV-2 spike protein,[65] and multi-enzyme complexes.[18,31,66–68] Models like the KH model are also excellent tools for method development due to their coarse but realistic description of proteins, their comparably simple energy functions, and their computational efficiency.[47] Note that recent applications of the KH model have already used our Complexes++ and *pycomplexes* software.[8,18,47,65]

To further support such applications, we designed our software to be not only efficient but also easily extensible. Compared to previous implementations of the KH model, new interaction potentials can be easily implemented in Complexes++. The addition of these potentials allows us to fine-tune pairwise interactions for a large variety of different systems. For example, the WCA potentials[21] could be used to fit coarse-grained models into electron densities. The electron density would be represented by beads that interact purely attractively via the WCA attractive potential with proteins. At low enough temperatures in a temperature replica-exchange simulation, we expect the proteins to assemble according to the electron density. To further facilitate customization, different domain types can be defined at runtime with explicit definitions of interaction potentials between different domain types. The new interaction potentials also allow us to implement alternative coarse-grained models.

One can improve the predictive power of coarse-grained force fields by encoding knowledge about the interactions and their deficiencies by modifying the force fields accordingly. *pycomplexes* has such functionality to set up diverse systems and modify them and their force fields built in. These capabilities are especially useful when we integrate additional knowledge in the form of experimental measurements.[11,69]

The extensibility of our software is an important foundation for further improvements. For example, molecular dynamics can efficiently propagate the coordinates of disordered linkers[16] and could be implemented in a hybrid Monte Carlo scheme.[70] Another feature might be to account for the flexibility of proteins.[71] Due to its flexible design, our software is set to implement libraries of different conformers and switch protein structures using a Monte Carlo scheme. Such structure libraries could be generated by combining elastic network models for proteins with low-frequency normal modes.[72]

We developed Complexes++ and its *pycomplexes* toolbox specifically for the KH and related models to be able to take full advantage of their properties. The KH linker model has been used in LAMMPS[73] and HOOMD-blue[43] to simulate liquid–liquid phase separation.[16] The KH model has also been implemented on GPUs.[74] Due to its clear, efficient, and extensible structure, our open-source code is well suited to combine protein models based on the KH model with, for example, elastic models of lipid membranes.[75] Such a combination of models of different scales is crucial for the computational investigation of complex sub-cellular structures due to their shear sizes and the large time scales of relevant processes.

In summary, Complexes++ and *pycomplexes* are efficient and flexible computational tools to study large scale protein complexes and their solutions. Our software emphasizes ease-of-use in setting up, modifying, post-processing, and visualizing simulations of the KH model and related models. The simulations are highly efficient, also due to our cell-list method specifically tailored to the wide range of applications of KH-like models. Our software thus enables scientists to efficiently explore structural ensembles of protein complexes and possibly refine these ensembles with experimental data.[11,30,69]

Our open-source software can be downloaded free of charge at https://github.com/bio-phys/complexespp. We encourage users to take advantage of this community platform and raise any issues they encounter with the software. The code can be re-used in agreement with the GNU Lesser General Public License version 3.

ACKNOWLEDGMENTS

We thank Young C. Kim and Bartosz Różycki for useful discussions; Benoit-Lofti Jacquelin for support in developing the *pycomplexes* toolbox; Klaus Reuter for support with the installation procedure; and Katrin Reichel, Alfredo Jost-Lopez, and Johannes Schöneberg for testing our software and providing feedback. We used the high-performance computing resources of the MPCDF. This work was supported by the Max Planck Society.

AUTHOR DECLARATIONS

Conflict of Interest

The authors have no conflicts to disclose.

Author Contributions

Max Linke: Conceptualization (equal); Formal analysis (lead); Methodology (equal); Software (lead); Validation (lead); Writing – original draft (supporting); Writing – review & editing (supporting). **Patrick K. Quoika**: Conceptualization (supporting); Methodology (supporting); Software (supporting); Validation (supporting); Writing – review & editing (supporting). **Berenger Bramas**: Software (supporting). **Jürgen Köfinger**: Conceptualization (supporting); Methodology (supporting); Supervision (equal); Validation (supporting); Writing – original draft (lead); Writing – review & editing (lead). **Gerhard Hummer**: Conceptualization (lead); Funding acquisition (lead); Methodology (lead); Project administration (lead); Resources (lead); Supervision (lead); Validation (supporting); Writing – review & editing (supporting).

DATA AVAILABILITY

The data that support the findings of this study are available from the corresponding author upon reasonable request.

APPENDIX A: ADDING EXPLICIT LINKERS

1. Adding Gaussian polymer chains

Pycomplexes uses an iterative algorithm to generate positions for the linker beads, given fixed positions for the first and last beads of the linker. Given the positions of beads N and 1, a single bead $N-1$ can be generated with the following algorithm:

1. Randomly choose a distance $R_{N,N-1} = d_{\text{start}}$ between beads N and $N-1$ distributed according to Ref. 76,

$$P(R_{ij}) = 4\pi R_{ij}^2 \left(\frac{3}{2\pi\langle R_{ij}^2\rangle}\right)^{3/2} \exp\left(-\frac{3R_{ij}^2}{2\langle R_{ij}^2\rangle}\right) \tag{A1}$$

for $R_{ij} > 0$, where $\langle R_{ij}^2\rangle = b^2(j-i)$ is the mean squared distances between beads i and j. This distance defines a sphere around bead $i = N$, on which bead $j = N-1$ will be placed.

2. Randomly choose a distance d_{end} between beads $i = 1$ and $j = N-1$, according to Eq. (A1), so that the sphere around bead 1 intersects with the sphere calculated in step 1. If the two spheres do not intersect, discard both randomly chosen radii and repeat steps 1 and 2.
3. Choose a random point on the intersection of two spheres centered at beads 1 and N to place the bead $N-1$. In three dimensions, the intersection is a circle, so a random angle θ has to be drawn.

To grow the next bead, with index $N-2$, simply repeat the algorithm with bead $N-1$ as the new starting point to choose the random distance $R_{N-1,N-2}$. Repeat this algorithm until all beads are generated.

This algorithm gives the three distances between the beads 1, N, and N-1 and the angle θ that uniquely determine where a new bead should be placed. To calculate the actual coordinates in the coordinate system of the simulation, we use the following algorithm:

1. Determine the axis $\vec{z'} = (z'_0, z'_1, z'_2)^\top$ along the vector $\vec{r}_N - \vec{r}_1$.
2. Determine and normalize perpendicular axes $\vec{x'} = \left(-\frac{z'_1+z'_2}{z'_0}, 1, 1\right)^\top$. Permutate elements of $\vec{x'}$ if $z'_0 = 0$.
3. Determine $\vec{y'} = \vec{x'} \times \vec{z'}$, where $\times$ is the cross product.
4. Determine angle ϕ using $\cos(\phi) = \frac{d_{\text{start}}^2 + d^2 - d_{\text{end}}^2}{2 d_{\text{start}} d}$, with $d = |\vec{r}_N - \vec{r}_1|$.

5. Calculate $\vec{r}_{N-1}$ in the coordinate system spanned by $(\vec{x}', \vec{y}', \vec{z}')$ from the spherical coordinates given by $(d_{\text{start}}, \theta, \phi)$.
6. Convert $\vec{r}_{N-1}$ into reference coordinate system $(\vec{x}, \vec{y}, \vec{z})$.

To avoid overlaps between beads and to generate more extended configurations, we add overlap checks in steps 1 and 3 of the first algorithm and redraw distances if two beads are too close to each other.

2. Relaxing Gaussian polymer chains

To generate more physical bead coordinates, we have to relax the structures generated by the previous algorithms. For the relaxation, we use the linker energy of the KH force field[13] in combination with a Monte Carlo equilibration. In a trial move, we displace an individual bead of the linker. The start and end beads are held fixed. For a linker of length N the probability to pick a bead is uniform between all $N - 2$ beads. A sweep consists of $N - 2$ trial moves. Note that we only do displacements of the beads and no angular or dihedral trial moves. Therefore, we adjust the size of the maximum displacement after each sweep to achieve a target acceptance ratio of 30%. If after a sweep the acceptance ratio is greater than 30%, we increase this step-width by 10%, if it is below 30%, we decrease the step-width by 10%. Note that we do not reset the counters for the numbers of attempted and accepted moves.

APPENDIX B: TYPICAL WORKFLOW

Below, we show a minimal code example for a typical workflow with Complexes++. Conveniently, we use *pycomplexes* for the preparation and analysis of the simulations. In this example, we use *pycomplexes* from the command line; however, it may also be used through Python, e.g., via interactive Python notebooks.

First, we prepare a short equilibration of our system of interest, Code Example 1. To this end, we use *pycomplexes* to generate the input file for this simulation, `equilibrate.cplx`. This file is processed by Complexes++.

The submodule "`pycomplexes convert`" processes the input file, `binding.top`, which we prepared beforehand. This is a human-readable file, which contains topological information such as the geometry of the simulation box and a PDB structure from which the coarse-grained topology shall be prepared. After completion of the simulation, we may analyze the output file `equilibrate.stat` to check, e.g., acceptance rate and potential energy. Besides that, we also obtain a trajectory and a reference structure in the form of a PDB file.

Next, we may start our production run, Code Example 2. We use *pycomplexes* to continue the production run from the equilibrated structure.

After completion of the simulation, we use *pycomplexes* to analyze our results, Code Example 3. We may, e.g., prepare a coarse-grained representation of the simulation trajectory in VMD. Furthermore, we may grow the linker domains in post-processing or we may analyze a replica exchange trajectory.

For a complete list of submodules of *pycomplexes*, a manual, and extensive tutorials, please see https://github.com/bio-phys/complexespp.

Code Example 1. System equilibration.

```
$pycomplexes convert binding.top \
                     equilibrate.cplx
$complexes++ --config=equilibrate.yaml
```

Code Example 2. Production run.

```
$pycomplexes equilibration \
          equilibrate.cplx \
          equilibrate.xtc \
          equilibrate_reference.pdb \
          production.cplx
$complexes++ --config=production.yaml
```

Code Example 3. Simulation analysis.

```
$pycomplexes visualize \
          -cc production.yaml
$pycomplexes addlinker \
          production.conf \
          production.xtc
$pycomplexes demux \
          production.log
          demux.xvg
```

REFERENCES

[1] M. Levitt, Angew. Chem., Int. Ed. **53**, 10006 (2014).
[2] M. Vendruscolo and C. M. Dobson, Curr. Biol. **21**, R68 (2011).
[3] C. Clementi, Curr. Opin. Struc. Biol. **18**, 10 (2008).
[4] S. Takada, Curr. Opin. Struct. Biol. **22**, 130 (2012).
[5] S. Kmiecik, D. Gront, M. Kolinski, L. Wieteska, A. E. Dawid, and A. Kolinski, Chem. Rev. **116**, 7898 (2016).
[6] M. Zacharias, Proteins **60**, 252 (2005).
[7] J. Wang, N. Charron, B. Husic, S. Olsson, F. Noé, and C. Clementi, J. Chem. Phys. **154**, 164113 (2021).
[8] S. von Bülow, M. Siggel, M. Linke, and G. Hummer, Proc. Natl. Acad. Sci. U.S.A. **116**, 9843 (2019).
[9] G. Nawrocki, W. Im, Y. Sugita, and M. Feig, Proc. Natl. Acad. Sci. U.S.A. **116**, 24562 (2019).
[10] W. Rieping, M. Habeck, and M. Nilges, Science **309**, 303 (2005).
[11] G. Hummer and J. Köfinger, J. Chem. Phys. **143**, 243150 (2015).
[12] S. Bottaro, T. Bengtsen, and K. Lindorff-Larsen, in *Methods in Molecular Biology*, edited by Z. Gáspári (Springer, 2020), Vol. 2112, pp. 219–240.
[13] Y. C. Kim and G. Hummer, J. Mol. Biol. **375**, 1416 (2008).
[14] S. Miyazawa and R. L. Jernigan, J. Mol. Biol. **256**, 623 (1996).
[15] J. Köfinger, M. J. Ragusa, I.-H. Lee, G. Hummer, and J. H. Hurley, Structure **23**, 809 (2015).
[16] G. L. Dignon, W. Zheng, Y. C. Kim, R. B. Best, and J. Mittal, PLoS Comput. Biol. **14**, e1005941 (2018).
[17] E. Boura, B. Różycki, D. Z. Herrick, H. S. Chung, J. Vecer, W. A. Eaton, D. S. Cafiso, G. Hummer, and J. H. Hurley, Proc. Natl. Acad. Sci. U.S.A. **108**, 9437 (2011).

[18] M. Klaus, E. Rossini, A. Linden, K. S. Paithankar, M. Zeug, Z. Ignatova, H. Urlaub, C. Khosla, J. Köfinger, G. Hummer, and M. Grininger, JACS Au **1**, 2162 (2021).

[19] ISO, ISO\IEC 14882:2017 Information technology — Programming languages — C++, 5th ed. (pub-ISO, pub-ISO:adr, 2017).

[20] H. M. Berman, Nucleic Acids Res. **28**, 235 (2000).

[21] J. D. Weeks, D. Chandler, and H. C. Andersen, J. Chem. Phys. **54**, 5237 (1971).

[22] I. Antes, Proteins **78**, 1084 (2010).

[23] *Modern Electronic Structure Theory, Advanced Series in Physical Chemistry No. V. 2*, edited by D. Yarkony (World Scientific, Singapore; River Edge, NJ, 1995).

[24] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, J. Chem. Phys. **21**, 1087 (1953).

[25] U. H. E. Hansmann, Chem. Phys. Lett. **281**, 140 (1997).

[26] K.-i. Okazaki, T. Sato, and M. Takano, J. Am. Chem. Soc. **134**, 8918 (2012).

[27] G. Bussi, Mol. Phys. **112**, 379 (2014).

[28] D. Frenkel and B. Smit, *Understanding Molecular Simulations*, 2nd ed. (Academic Press, London, 2002).

[29] V. Hornak, R. Abel, A. Okur, B. Strockbine, A. Roitberg, and C. Simmerling, Proteins **65**, 712 (2006).

[30] B. Różycki, Y. C. Kim, and G. Hummer, Structure **19**, 109 (2011).

[31] B. Różycki, P.-A. Cazade, S. O'Mahony, D. Thompson, and M. Cieplak, Phys. Chem. Chem. Phys. **19**, 21414 (2017).

[32] C. Hyeon, G. Morrison, and D. Thirumalai, Proc. Natl. Acad. Sci. U.S.A. **105**, 9604 (2008).

[33] E. P. O'Brien, G. Morrison, B. R. Brooks, and D. Thirumalai, J. Chem. Phys. **130**, 124903 (2009).

[34] R. B. Best, Y.-G. Chen, and G. Hummer, Structure **13**, 1755 (2005).

[35] E. Gamma, R. Helm, R. Johnson, R. E. Johnson, J. Vlissides *et al.*, *Design Patterns: Elements of Reusable Object-Oriented Software* (Pearson Deutschland GmbH, 1995).

[36] B. Bramas, Scientific Programming **2017**, 5482468.

[37] E. Agullo, B. Bramas, O. Coulaud, E. Darve, M. Messner, and T. Takahashi, Concurr. Comput. Pract. Exp. **28**, 2608 (2016).

[38] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, Nature **585**, 357 (2020).

[39] R. Gowers, M. Linke, J. Barnoud, T. Reddy, M. Melo, S. Seyler, J. Domański, D. Dotson, S. Buchoux, I. Kenney, and O. Beckstein, in *Proceedings of the 15th Python in Science Conference, SciPy, 11-17 July 2016, Austin, TX*, edited by S. Benthall and S. Rostrup (SciPy, 2016), pp. 98–105.

[40] K. Thomas, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, C. Willing, and J. D. Team, in *Positioning and Power in Academic Publishing: Players, Agents and Agendas, 20th International Conference on Electronic Publishing* (ELPUB, Göttingen, Germany, 2016), pp. 87–90.

[41] D. A. Case, T. E. Cheatham, T. Darden, H. Gohlke, R. Luo, K. M. Merz, A. Onufriev, C. Simmerling, B. Wang, and R. J. Woods, J. Comput. Chem. **26**, 1668 (2005).

[42] M. J. Abraham, T. Murtola, R. Schulz, S. Páll, J. C. Smith, B. Hess, and E. Lindahl, SoftwareX **1-2**, 19 (2015).

[43] J. A. Anderson, J. Glaser, and S. C. Glotzer, Comput. Mater. Sci. **173**, 109363 (2020).

[44] O. Ben-Kiki, C. Evans, and I. döt Net, http://yaml.org/spec/1.2/spec.html (2018).

[45] B. Różycki and E. Boura, Biophys. Chem. **288**, 106843 (2022).

[46] W. Humphrey, A. Dalke, and K. Schulten, J. Mol. Graph. **14**, 33 (1996).

[47] A. Jost Lopez, P. K. Quoika, M. Linke, G. Hummer, and J. Köfinger, J. Phys. Chem. B **124**, 4673 (2020).

[48] A. C. Stark, C. T. Andrews, and A. H. Elcock, J. Chem. Theory Comput. **9**, 4176 (2013).

[49] Z. Benayad, S. von Bülow, L. S. Stelzl, and G. Hummer, J. Chem. Theory Comput. **17**, 525 (2021).

[50] R. Diamond, J. Mol. Biol. **82**, 371 (1974).

[51] M. E. Pacold, S. Suire, O. Perisic, S. Lara-Gonzalez, C. T. Davis, E. H. Walker, P. T. Hawkins, L. Stephens, J. F. Eccleston, and R. L. Williams, Cell **103**, 931 (2000).

[52] M. J. Ragusa, R. E. Stanley, and J. H. Hurley, Cell **151**, 1501 (2012).

[53] F. Zhu and B. Chen, J. Chem. Inf. Model. **55**, 1361 (2015).

[54] H. Sha and F. Zhu, J. Chem. Inf. Model. **57**, 1134 (2017).

[55] Y. C. Kim and J. Mittal, Phys. Rev. Lett. **110**, 208102 (2013).

[56] Y. C. Kim, R. B. Best, and J. Mittal, J. Chem. Phys. **133**, 205101 (2010).

[57] J. Rosen, Y. C. Kim, and J. Mittal, J. Phys. Chem. B **115**, 2683 (2011).

[58] A. Baumlova, D. Chalupska, B. Różycki, M. Jovic, E. Wisniewski, M. Klima, A. Dubankova, D. P. Kloer, R. Nencka, T. Balla, and E. Boura, EMBO Rep. **15**, 1085 (2014).

[59] B. Różycki, M. Cieplak, and M. Czjzek, J. Struct. Biol. **191**, 68 (2015).

[60] D. Chalupska, A. Eisenreichova, B. Różycki, L. Rezabkova, J. Humpolickova, M. Klima, and E. Boura, J. Struct. Biol. **200**, 36 (2017).

[61] D. Malinverni, A. Jost Lopez, P. De Los Rios, G. Hummer, and A. Barducci, eLife **6**, e23471 (2017).

[62] K. M. Ravikumar, W. Huang, and S. Yang, Biophys. J. **103**, 837 (2012).

[63] S. Yadahalli, V. V. Hemanth Giri Rao, and S. Gosavi, Isr. J. Chem. **54**, 1230 (2014).

[64] N. Fortoul, P. Singh, C.-Y. Hui, M. Bykhovskaia, and A. Jagota, Biophys. J. **108**, 2258 (2015).

[65] M. Sikora, S. von Bülow, F. E. C. Blanc, M. Gecht, R. Covino, and G. Hummer, PLoS Comput. Biol. **17**, e1008790 (2021).

[66] N. Fortoul, M. Bykhovskaia, and A. Jagota, J. Phys. Chem. B **122**, 10834 (2018).

[67] B. G. Horan, G. H. Zerze, Y. C. Kim, D. Vavylonis, and J. Mittal, FEBS Lett. **592**, 1804 (2018).

[68] B. Różycki and M. Cieplak, Mol. BioSyst. **12**, 3589 (2016).

[69] S. Bottaro and K. Lindorff-Larsen, Science **361**, 355 (2018).

[70] B. Mehlig, D. W. Heermann, and B. M. Forrest, Phys. Rev. B **45**, 679 (1992).

[71] M. Zacharias, Protein Sci. **12**, 1271 (2003).

[72] A. May and M. Zacharias, Proteins **70**, 794 (2008).

[73] S. Plimpton, J. Comput. Phys. **117**, 1 (1995).

[74] I. Tunbridge, R. B. Best, J. Gain, and M. M. Kuttel, J. Chem. Theory Comput. **6**, 3588 (2010).

[75] M. Siggel, S. Kehl, K. Reuter, J. Köfinger, and G. Hummer, "TriMem: A parallelized hybrid Monte Carlo software for efficient simulations of lipid membranes," J. Chem. Phys. **157**, 174801 (2022).

[76] H. Yamakawa, *Modern Theory of Polymer Solutions* (Harper and Row, NewYork, 1971).