gmxapi: a Gromacs-native Python interface for molecular dynamics with ensemble and plugin support

M. Eric Irrgang[1], Caroline Davis[1], and Peter M. Kasson[1,2*]
[1]Departments of Molecular Physiology and Biomedical Engineering, University of Virginia, Charlottesville VA 22908
[2]Science for Life Laboratory, Department of Cell and Molecular Biology, Uppsala University, 75124 Uppsala, Sweden
[*] To whom correspondence should be addressed: kasson@virginia.edu, Box 800886 Charlottesville VA 22908. ORCID 0000-0002-3111-8103.

Abstract

Gmxapi provides an integrated, native Python API for both standard and advanced molecular dynamics simulations in GROMACS. The Python interface permits multiple levels of integration with the core GROMACS libraries, and legacy support is provided via an interface that mimics the command-line syntax, so that all GROMACS commands are fully available. Gmxapi has been officially supported since the GROMACS 2019 release and is installed by default in current versions of the software. Beyond simply wrapping GROMACS library operations, the API permits several advanced operations that are not feasible using the prior command-line interface. First, API allows custom user plugin code within the molecular dynamics force calculations, so users can execute custom algorithms without modifying the GROMACS source. Second, the Python interface allows tasks to be dynamically defined, so high-level algorithms for molecular dynamics simulation and analysis can be coordinated with loop and conditional operations. Gmxapi makes GROMACS more accessible to custom Python scripting while also providing support for high-level data-flow simulation algorithms that were previously feasible only in external packages.

Introduction

As molecular dynamics simulations have become more complex and mature as scientific tools, typical simulation use is shifting from manual invocation of a few simulations and analysis tools to pre-defined simulation and analysis protocols, often involving many simulation trajectories. In addition, custom applications for advanced sampling or molecular structure refinement are becoming more common, where the molecular dynamics engine is used as part of a more complex data integration protocol. In these more complex use scenarios, toolchain and file-system management as well as execution of many simulations can become limiting factors. As a result, many scientists either develop custom scripts, custom modifications of the simulation source code, or adapt general-purpose workflow engines[1-3] to molecular simulation tasks. There also exist some molecular-simulation-specific workflow engines, frequently coupled to the underlying molecular simulation code via command-line interfaces[4-9]. All of the above can be brittle, particularly without good APIs for simulation interfaces to workflow engines. Integration of parallel analysis into unified jobs can also be challenge, whether this is performed via built-in tool parallelism or user-level coding, such as Python multiprocessing packages or workflow/execution managers[2-3, 5-6, 10-12].

We previously reported on gmxapi 0.0.4[13], which allows Python driven molecular dynamics in GROMACS[14-15] to be extended at run time with custom researcher code. Here, we describe features present in gmxapi 0.2 and beyond, which offers both more advanced ensemble simulation logic and a feature-complete interface for GROMACS tools and analysis. This framework allows ensemble methods to be implemented with less code, and without patching an official GROMACS release. Data-flow-oriented programming logic and general paradigm for integrating new software tools further enhance the utility of the package. Both the libgmxapi C++ interface and the gmxapi Python package are now maintained and distributed with GROMACS; gmxapi 0.2 is integrated with GROMAC 2021.

Design and Implementation

We used pybind11[16] to implement Python bindings to a C++ API for the GROMACS[14-15] molecular simulation library. The design concept of gmxapi, as with many modern high-level interfaces, is for the user to construct a computational graph where execution is then managed by lower layers of the software stack. Similar to Keras[17], we recognize that explicit data-flow programming is not natural for all users and strive to provide an interface where data-flow can be specified explicitly or implicitly. Figure 1 illustrates the gmxapi paradigm in which control signals and data are treated in the same framework. Branching and other control events require that some work is "dynamic", such that the complete workflow graph may not be determined until runtime. However, the workflow can be fully specified in terms of workflow command and data primitives (Figure 2).

Data flow formalism

Important design considerations include allowance for software components (internal or external) to make decisions based on the requirements of the largest possible section of the work graph. Much optimization (by workflow managers or in collaboration with execution management software) is coupled to data locality, compute resource availability, and task scheduling. To allow for optimization based on data dependencies and data locality, execution is deferred as much as possible to when and where it is required.

Researchers use the gmxapi Python package to assemble and execute a directed acyclic graph (DAG) of computational work. Nodes of the work graph represent discrete *operations* that consume and produce well defined inputs and outputs. Operation and data references in the work graph are merely proxy objects to the computational tasks and data until the graph is run. Execution management may be optimized by running only as much of the graph as necessary to satisfy explicit data dependencies. As a basic case, the tools for managing simulation input do not transfer data from the C++ library to the Python interpreter unless explicitly requested by the user.

For example, the researcher calls a gmxapi command to add work to the work graph and obtain a reference to the graph node representing the operation. Output attributes on the

operation may be used as inputs to further commands, and may be used as “Future” results. Calling “result()” on a Future forces dependency resolution and data localization.

Python interface

The simplest gmxapi operations manipulate signals, data topology, or typing. “join_arrays”, “logical_not”, and “make_constant” have strict definitions for inputs and outputs (Figure 2a). Operations can also be compositions of simpler operations. “concatenate_lists” is a helper function that reduces a number of inputs in terms of “join_arrays” (Figure 2b).

Where possible, the subroutines responsible for executing work in the graph bind to API calls in a compiled library. Commands like “read_tpr”, “modify_input”, and “mdrun” use a binary Python extension module (written in C++ using “pybind11”) to interact with Gromacs operations and data through libgmxapi (a C++ library installed by default with recent versions of Gromacs).

“gmxapi.commandline.cli()” is a simple pure-Python gmxapi Operation for wrapping command line tools. It is a thin wrapper of the Python “subprocess” module, so command inputs and outputs are embedded in the argument list parameter. Instances of “gmxapi.commandline.cli” are not conducive to DAG representations of data flow, so a helper function creates additional gmxapi primitives to provide a consistent feel of named inputs and outputs. “gmxapi.commandline_operation()” generates a graph of gmxapi primitives around “cli” to translate input and output arguments into operation inputs and outputs, as specified by the user (Figure 2b).

“commandline_operation” is written using a utility function that is also available to gmxapi users. “function_wrapper()” allows arbitrary user code to be transformed into a gmxapi operation by “decorating” a Python function definition with “@function_wrapper” (Figure 2c).

Conditional iteration also takes the form of conventional gmxapi command syntax, but relies on some metaprogramming under the hood. “while_loop” allows a graph to be dynamically extended as a chain of repeated operations, including fused operations constructed with the “subgraph” tool. A subgraph allows inputs and outputs to be expressed in terms of other gmxapi operations. When used with “while_loop”, internal subgraph state can be propagated from one iteration to the next.

The work graph

gmxapi allows data flow topology to be represented independently from prescriptions for execution strategies or run time resource assignment details. An array of input sources provided to a command generates a corresponding set of tasks, such as for trajectory ensemble simulations. (Note gmxapi cannot currently distinguish between coupled and uncoupled ensembles, and so currently requires co-execution through the gmxapi 0.7 mpi4py executor. Operations defined with “@function_wrapper”, including the other built-in Operations, are assumed to be uncoupled, and are launched in a sequence determined

by the DAG topology during recursive resolution of data dependencies, executed sequentially by the simple built-in gmxapi 0.1 executor.)

Commands operate on handles to work as soon as it is expressed rather than blocking for execution, meaning that chains of simulations can be expressed before the first simulation begins.

Since Python-level objects represent nodes in a directed acyclic graph of data flow (DAG), arbitrarily complex topologies can be expressed unambiguously in terms of script variables without unexpected side-effects. Trajectories can be forked or extended without re-executing simulation segments or overwriting previous results.

Ensemble trajectory data can be gathered in parallel pipelines or by iteration.

Execution and control flow

MD research protocols expressed with gmxapi participate in several types of control flow, with additional logic available to the researcher.

Code is executed only to satisfy data dependencies. Script-level logic can force data dependency resolution for (blocking) logic with e.g. “if X.result() == Y: expression”.

Logic evaluation can be deferred by using gmxapi operations, such as a "while_loop", whose condition is expressed in terms of the Future result of another gmxapi operation.

Pluggable MD extension code can arrange to access additional gmxapi resources at run time. Arbitrary function objects can be provided, such as an ensemble-wide "ReduceAll" callable that the custom code can use for adaptivity. gmxapi also registers with the Gromacs “StopSignal” facility so that plugin code can dynamically truncate the originally prescribed trajectory, such as when an ensemble value has converged.

Interoperation of Python with binary object libraries

The Python interpreter serves as a convenient tool for dynamically loading object libraries at run time. Gmxapi establishes a protocol[18] for providing MD extension code objects to the GROMACS simulator during launch. This allows researcher code to be executed during the MD integration loop with minimal overhead. Python is used only to facilitate binding. Once the simulation launches, GROMACS and the extension code communicate directly with C++ interfaces. This is possible with a standard GROMACS installation. The researcher does not need to patch or modify GROMACS itself.

Gmxapi does not prescribe a C/C++ bindings strategy. We have used pybind11 for Gromacs bindings, but the protocols for Python interaction with compiled code rely only on the Python C API (PyCapsule) and the public libgmxapi C++ interface.

Results

Simulation input generally requires a chain of tools, including command line programs whose input and output can only be expressed as file paths. gmxapi syntax allows the output of any tool to be used as the input to another with consistent Python syntax. Figure 3 illustrates a chain of Gromacs tools preparing simulation input, feeding into "gmxapi.mdrun" to run a batch of simulations.

To illustrate the application of gmxapi, we demonstrate refinement of an HIV gp41 conformational ensemble based on DEER spectroscopy data. We illustrate this using a starting crystal structure for the BG505 SOSIP of HIV gp41[19] and previously reported DEER spectroscopy data of 5 different spin labels positions on this SOSIP[20]. The gp41 trimer was asymmetrically restrained (each of the three monomer-monomer distances was sampled separately), and the previously reported BRER algorithm for refining heterogeneous conformational ensembles[18] was applied. This involves a custom force plugin for GROMACS, and gmxapi was used to run an array of 250 refinement simulations, each randomly sampling target distance values from the experimental distribution for each measured spin-label pair. Further details are given in the Supplement. Full convergence of the refinement would require additional simulation sampling of the 15-dimensional experimental distribution, but even after 24 wall-clock hours of simulation a reasonable sampling of the experimental distributions was obtained (Figure 3). This application illustrates the ease of simulation ensemble management and application of custom biasing algorithms within Gromacs facilitated by gmxapi.

Availability and Future Directions

The libgmxapi C++ interface appeared in the GROMACS 2019 release and has been part of a standard GROMACS installation since 2020. Gmxapi 0.2 (described here) shipped with GROMACS 2021.

The gmxapi Python package is also maintained as part of the GROMACS repository at https://gitlab.com/gromacs/gromacs. It can be installed from the GROMACS sources (https://gitlab.com/gromacs/gromacs/-/tree/master/python_packaging/src) or from https://pypi.org/project/gmxapi/ with “pip”, but needs to be told which GROMACS installation to use. During installation, the gmxapi Python package builds a C++ extension module against the GROMACS installation. See the documentation at https://manual.gromacs.org/current/gmxapi for details, or just activate your GROMACS installation in the usual manner (such as by sourcing GMXRC or loading an environment module on an HPC system).

Custom Molecular Dynamics extension code is illustrated in a “sample_restraint” package: https://gitlab.com/gromacs/gromacs/-/tree/master/python_packaging/sample_restraint. Bias-resampling ensemble refinement (BRER) MD restraint potentials[18] are forked from this sample code and can be found at https://github.com/kassonlab/brer_plugin. Scripted BRER workflows are available at https://github.com/kassonlab/run_brer. The “run_brer” scripts were developed prior to gmxapi 0.1 and use internal logic to manage some data flow and to maintain workflow

state. Both “run_brer” and “gmxapi” are being updated to illustrate that data flow and workflow state can be managed by the framework to allow simpler, more robust application code for applying new methods like BRER.

Python design patterns using Futures are becoming more widespread and standardized [7, 21]. Minimal refinement of the gmxapi Future protocol is needed to be fully compatible with native and third-party frameworks for concurrent or asynchronous program flow.

Gmxapi issues are tracked with the label “gmxapi” at https://gitlab.com/gromacs/gromacs/-/issues. Code contributions follow the GROMACS contribution procedure (https://manual.gromacs.org/current/dev-manual/contribute.html). However, gmxapi is intended to allow for maximal extensibility without requiring modification to the sources.

A discussion forum is available at https://gromacs.bioexcel.eu/tag/gmxapi

Acknowledgements. The authors thank Mark Abraham for many helpful discussions. This work was supported by R01 GM115790 and OAC-1835780 to P.M.K. Computational resources on Frontera were provided under NSF LRAC MCB20006.

Figures

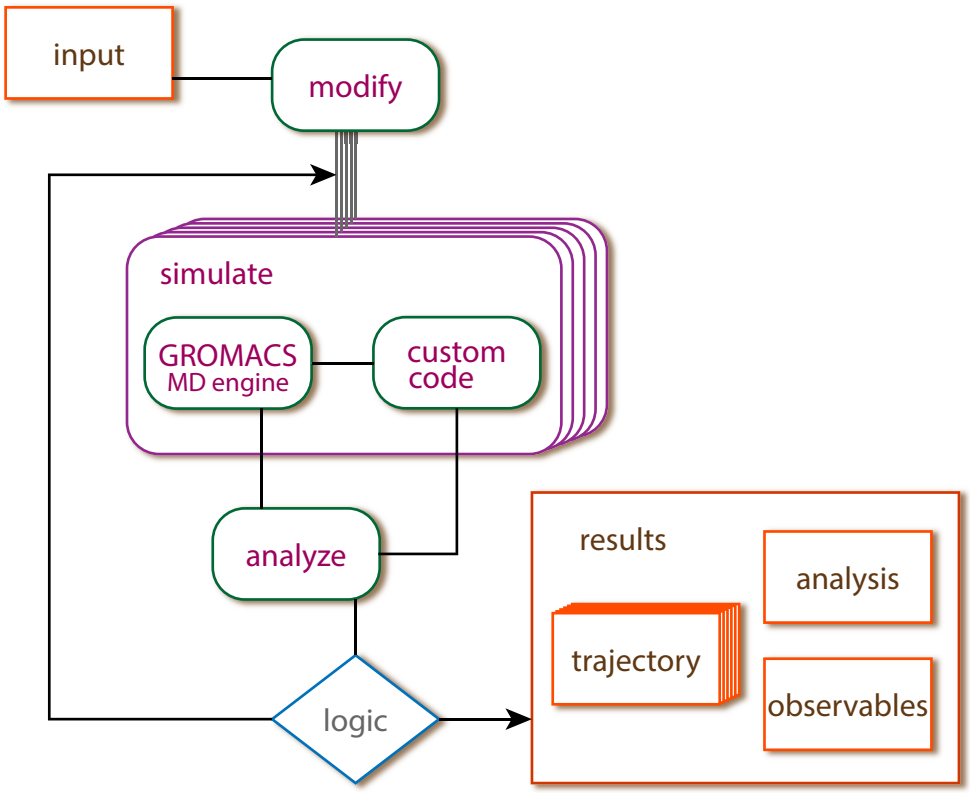

Figure 1: Schematic of combined data flow and control flow for a segment of a complex simulation workflow.

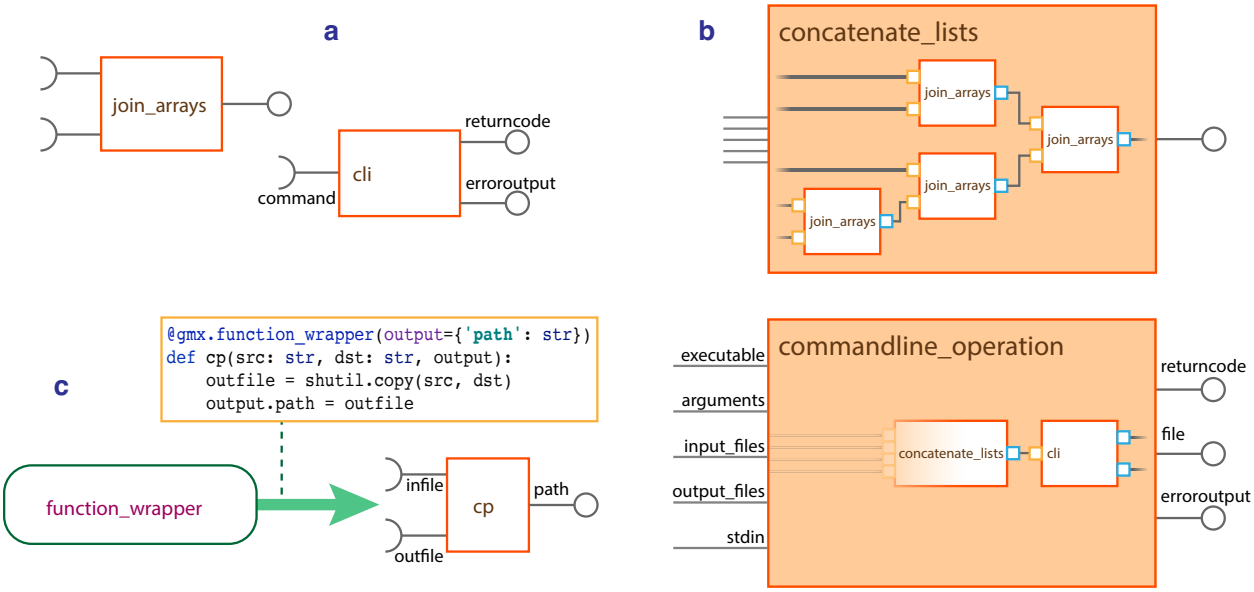

Figure 2: (a) gmxapi operations have well defined inputs and outputs. "gmxapi.commandline.cli()" is a thin pure-Python wrapper for command line tools. (b) "concatenate_lists" is a helper function that reduces a number of inputs in terms of "join_arrays". The helper function "commandline_operation" generates a graph of

gmxapi primitives around "cli" to provide a more consistent feel of named inputs and outputs. (c) @function_wrapper allows arbitrary user code to be transformed into a gmxapi operation.

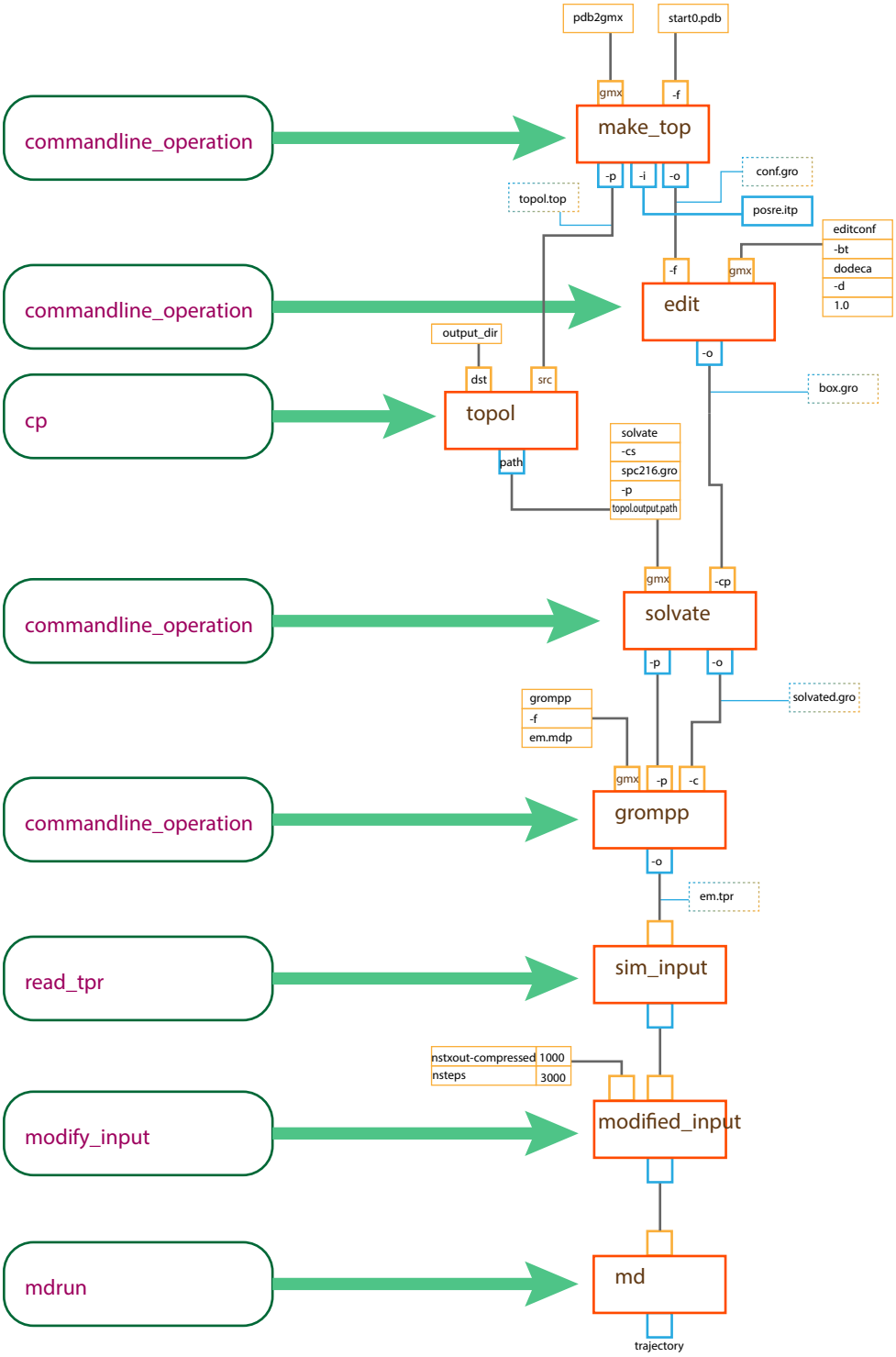

Figure 3: Simulation input generally requires a chain of tools, including command line programs whose input and output can only be expressed as file paths. gmxapi syntax

allows the output of any tool to be used as the input to another with consistent Python syntax. Here we show a chain of Gromacs tools preparing simulation input, feeding into "gmxapi.mdrun" to run a batch of simulations.

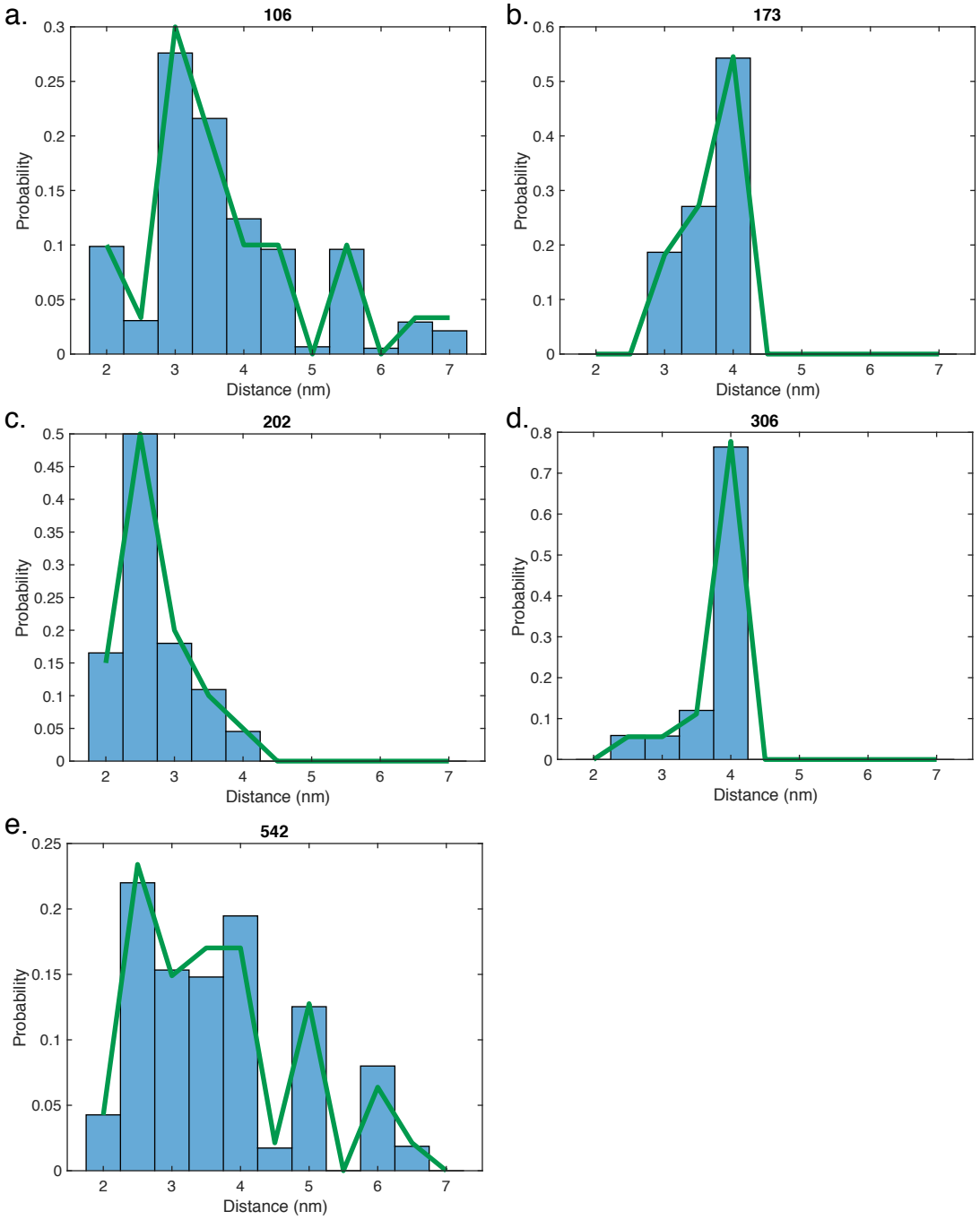

Figure 4: Residue-residue distance distributions in a simulated ensemble of HIV gp41. DEER spectroscopy was used to measure distance distributions between two monomers of the gp41 trimer, so each residue designates a monomer-monomer

residue pair. Panels a-e show plots for the 5 restrained residues in the HIV trimer: 106, 173, 202, 306, and 542 respectively. Discretized DEER distance distributions are plotted in green, and simulation results are plotted in blue bars. The simulation ensemble shows good convergence to the measured values..

Literature Cited

1. Goecks, J.; Nekrutenko, A.; Taylor, J.; Team, G., Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology* **2010,** *11* (8).
2. Deelman, E.; Singh, G.; Su, M.-H.; Blythe, J.; Gil, Y.; Kesselman, C.; Mehta, G.; Vahi, K.; Berriman, G. B.; Good, J., Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming* **2005,** *13* (3), 219-237.
3. Wolstencroft, K.; Haines, R.; Fellows, D.; Williams, A.; Withers, D.; Owen, S.; Soiland-Reyes, S.; Dunlop, I.; Nenadic, A.; Fisher, P.; Bhagat, J.; Belhajjame, K.; Bacall, F.; Hardisty, A.; de la Hidalga, A. N.; Vargas, M. P. B.; Sufi, S.; Goble, C., The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. *Nucleic Acids Res* **2013,** *41* (W1), W557-W561.
4. Pronk, S.; Larsson, P.; Pouya, I.; Bowman, G. R.; Haque, I. S.; Beauchamp, K.; Hess, B.; Pande, V. S.; Kasson, P. M.; Lindahl, E., Copernicus: A new paradigm for parallel adaptive molecular dynamics. *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis* **2011**, 60.
5. Pronk, S.; Pouya, I.; Lundborg, M.; Rotskoff, G.; Wesén, B.; Kasson, P. M.; Lindahl, E., Molecular Simulation Workflows as Parallel Algorithms: The Execution Engine of Copernicus, a Distributed High-Performance Computing Platform. *J Chem Theory Comput* **2015**.
6. Jain, A.; Ong, S. P.; Chen, W.; Medasani, B.; Qu, X. H.; Kocher, M.; Brafman, M.; Petretto, G.; Rignanese, G. M.; Hautier, G.; Gunter, D.; Persson, K. A., FireWorks: a dynamic workflow system designed for high-throughput applications. *Concurr Comp-Pract E* **2015,** *27* (17), 5037-5059.
7. Babuji, Y.; Woodard, A.; Li, Z.; Katz, D. S.; Clifford, B.; Kumar, R.; Lacinski, L.; Chard, R.; Wozniak, J. M.; Foster, I. In *Parsl: Pervasive parallel programming in python*, 2019; pp 25-36.
8. Wilde, M.; Hategan, M.; Wozniak, J. M.; Clifford, B.; Katz, D. S.; Foster, I., Swift: A language for distributed parallel scripting. *Parallel Computing* **2011,** *37* (9), 633-652.
9. Balasubramanian, V.; Bethune, I.; Shkurti, A.; Breitmoser, E.; Hruska, E.; Clementi, C.; Laughton, C.; Jha, S. In *Extasy: Scalable and flexible coupling of md simulations and advanced sampling techniques*, e-Science (e-Science), 2016 IEEE 12th International Conference on, IEEE: 2016; pp 361-370.
10. van der Aalst, W. M. P.; Jablonski, S., Dealing with workflow change: identification of issues and solutions. *Comput Syst Sci Eng* **2000,** *15* (5), 267-276.
11. Ludascher, B.; Altintas, I.; Berkley, C.; Higgins, D.; Jaeger, E.; Jones, M.; Lee, E. A.; Tao, J.; Zhao, Y., Scientific workflow management and the Kepler system. *Concurr Comp-Pract E* **2006,** *18* (10), 1039-1065.
12. Mattoso, M.; Dias, J.; Ocana, K. A. C. S.; Ogasawara, E.; Costa, F.; Horta, F.; Silva, V.; de Oliveira, D., Dynamic steering of HPC scientific workflows: A survey. *Future Gener Comp Sy* **2015,** *46*, 100-113.
13. Irrgang, M. E.; Hays, J. M.; Kasson, P. M., gmxapi: a high-level interface for advanced control and extension of molecular dynamics simulations. *Bioinformatics* **2018,** *34* (22), 3945-3947.
14. Pronk, S.; Pall, S.; Schulz, R.; Larsson, P.; Bjelkmar, P.; Apostolov, R.; Shirts, M. R.; Smith, J. C.; Kasson, P. M.; van der Spoel, D.; Hess, B.; Lindahl, E., GROMACS 4.5: a high-throughput and highly parallel open source molecular simulation toolkit. *Bioinformatics* **2013,** *29* (7), 845-54.
15. Abraham, M. J.; Murtola, T.; Schulz, R.; Páll, S.; Smith, J. C.; Hess, B.; Lindahl, E., GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX* **2015,** *1*, 19-25.
16. Wenzel, J.; Rhinelander, J.; Moldovan, D. pybind11–Seamless operability between C++ 11 and Python. https://github.com/pybind/pybind11.
17. Chollet, F., Keras. *Keras. io* **2015**.
18. Hays, J. M.; Cafiso, D. S.; Kasson, P. M., Hybrid Refinement of Heterogeneous Conformational Ensembles Using Spectroscopic Data. *J Phys Chem Lett* **2019**, 3410-3414.
19. Do Kwon, Y.; Pancera, M.; Acharya, P.; Georgiev, I. S.; Crooks, E. T.; Gorman, J.; Joyce, M. G.; Guttman, M.; Ma, X.; Narpala, S.; Soto, C.; Terry, D. S.; Yang, Y.; Zhou, T.; Ahlsen, G.; Bailer, R. T.; Chambers, M.; Chuang, G.-Y.; Doria-Rose, N. A.; Druz, A.; Hallen, M. A.; Harned, A.; Kirys, T.; Louder, M. K.; O'Dell, S.; Ofek, G.; Osawa, K.; Prabhakaran, M.; Sastry, M.; Stewart-Jones, G. B. E.; Stuckey, J.; Thomas, P. V.; Tittley, T.; Williams, C.; Zhang, B.; Zhao, H.; Zhou, Z.; Donald, B. R.; Lee, L. K.; Zolla-Pazner, S.; Baxa, U.; Schön, A.; Freire, E.; Shapiro, L.; Lee, K. K.; Arthos, J.; Munro, J. B.; Blanchard, S. C.; Mothes, W.; Binley, J. M.; McDermott, A. B.; Mascola, J. R.; Kwong, P. D., Crystal structure, conformational fixation and entry-related interactions of mature ligand-free HIV-1 Env. *Nature Structural & Molecular Biology* **2015,** *22* (7), 522-531.

20. Stadtmueller, B. M.; Bridges, M. D.; Dam, K.-M.; Lerch, M. T.; Huey-Tubman, K. E.; Hubbell, W. L.; Bjorkman, P. J., DEER Spectroscopy Measurements Reveal Multiple Conformations of HIV-1 SOSIP Envelopes that Show Similarities with Envelopes on Native Virions. *Immunity* **2018,** *49* (2), 235-246.e4.
21. Hadjidoukas, P. E.; Bartezzaghi, A.; Scheidegger, F.; Istrate, R.; Bekas, C.; Malossi, A. C. I., torcpy: Supporting task parallelism in Python. *SoftwareX* **2020,** *12*, 100517.