JCTC Journal of Chemical Theory and Computation

Article

pubs.acs.org/JCTC

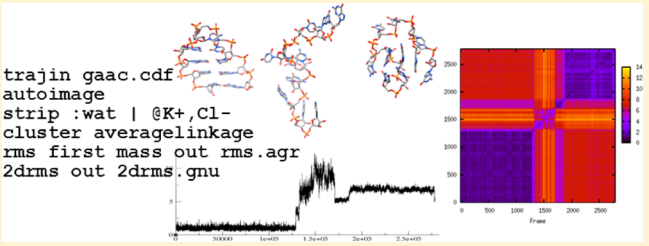
PTRAJ and CPPTRAJ: Software for Processing and Analysis of Molecular Dynamics Trajectory Data

Daniel R. Roe* and Thomas E. Cheatham, III*

Department of Medicinal Chemistry, College of Pharmacy, 2000 South 30 East Room 105, University of Utah, Salt Lake City, Utah 84112, United States

Supporting Information

ABSTRACT: We describe PTRAJ and its successor CPPTRAJ, two complementary, portable, and freely available computer programs for the analysis and processing of time series of three-dimensional atomic positions (i.e., coordinate trajectories) and the data therein derived. Common tools include the ability to manipulate the data to convert among trajectory formats, process groups of trajectories generated with ensemble methods (e.g., replica exchange molecular dynamics), image with periodic boundary conditions, create average structures, strip subsets of the system, and perform calculations such as RMS fitting, measuring distances, B-factors, radii of gyration, radial distribution functions, and time correlations, among other actions and analyses. Both the PTRAJ and CPPTRAJ programs and source code are freely available under the GNU General Public License version 3 and are currently distributed within the AmberTools 12 suite of support programs that make up part of the Amber package of computer programs (see http://ambermd.org). This overview describes the general design, features, and history of these two programs, as well as algorithmic improvements and new features available in CPPTRAJ.



INTRODUCTION

Biomolecular simulation methods have proven useful in a wide variety of applications ranging from protein folding and computer aided drug design to the characterization of materials properties.[1] Broadly defined, such methods include not only the molecular dynamics (MD) and Monte Carlo approaches, but also molecular docking, geometry optimization, free energy, and/or path sampling approaches. Any of these methods may generate a time series or an ensemble of three-dimensional (3D) atomic positions of a model or set of models, i.e. a coordinate "trajectory." In practice, a key enabler of any robust biomolecular simulation method is not only providing the ability to generate conformational ensembles that describe the processes of interest but also facilitating the means to analyze the ensemble data. Beyond simply understanding the static structure derived from experiment or the starting and end points of a simulation, much more information can be gleaned by characterizing the full ensemble or time dependent evolution of the sets of 3D atomic or model coordinates. This "trajectory analysis" refers to analyzing a (potentially large) set or time series of 3D positions and their derived properties. Compared to the gigabyte or smaller trajectories that could be generated in the mid-1990s, today we can run 100 ~300 ns replicate MD trajectories of a solvated DNA 18-mer on GPU resources such as XSEDE's Keeneland in ~4 months to generate over 22 terabytes (TB) of data representing an aggregate ~30 μs of simulation data. The data explosion is only getting worse with access to resources like Blue Waters at NCSA which has over 3000 K20X GPUs available. Given that these data sets are becoming larger and are generated more quickly, it is critical that the data generated can be analyzed not only rapidly and efficiently but in a manner that is both flexible and easy to use and ideally within a generalizable and extensible framework.

PTRAJ (short for Process TRAJectory) has served as the main analysis program of the Amber software package[2] since the early 1990s and offers a wide range of functionality, including simple geometric analysis of coordinates, conversion between different coordinate formats, advanced vector and matrix analysis, and so on. The overall goal of PTRAJ was to provide a unified interface to commonly needed analysis tools as well as provide reusable routines and an extensible framework for the easy development and incorporation of new analysis tools. Although PTRAJ has been an integral part of Amber for decades, this is the first publication providing an in-depth description of the code outside the information provided by the Amber manuals and tutorials.

Several other software packages for the analysis of coordinate trajectories exist, including VMD,[3] MMTSB,[4] MDAnalysis,[5] Pteros,[6] LOOS/PyLOOS,[7] and HiMach.[8] The MDAnalysis, Pteros, LOOS/PyLOOS, and HiMach packages provide libraries of functions (essentially analysis-oriented programming language extensions) that can be used to construct analysis programs. While this approach provides a great deal of flexibility, the low-level interface can be challenging for some users. The MMTSB package provides a series of Perl scripts,

Received: April 26, 2013
Published: June 10, 2013

each of which can perform a different type of analysis; however, combining different types of analysis into a single run is not straightforward. VMD provides a convenient graphical interface for various kinds of coordinate analysis. While VMD does provide a TCL text interface that can be used for scripting, it is not typically used for batch-processing various types of analyses (which is particularly useful when analyzing data at remote sites). In contrast to these software packages, PTRAJ provides a variety of higher-level analysis commands via a unified interface that is still amenable to batch-processing and performing multiple types on analysis in one run.

The initial incarnation of PTRAJ evolved out of the program RDPARM (ReaD PARaMeters), which was designed to read and display parameter information from a single Amber Topology file. At the time RDPARM was developed in the Kollman lab at UCSF, the formatted Amber Topology data files could be generated via multiple mechanisms including the soon-after retired Prep-Link-Edit-Parm (PLEP) programs and its replacement program still used in Amber today, LEaP.[2a] The topology files were difficult to read not only due to the formatting but also since the ordering and specification of information could be different yet still lead to equivalent results (for example, via different bond orderings or equivalent but opposite torsion atom definitions X−C−C−Y vs Y−C−C−X); because of this, one could not simply "diff" the files to expose differences between PLEP and LEaP topologies. RDPARM emerged when many of the members of the Kollman lab realized after months of simulations with new LEaP topologies that there were inconsistencies in the simulation results. A tool was needed to "read" the topology files and allow easy display of the parameters (to see what was wrong in the files), and RDPARM was created. A summary of the functionality and output is presented in the Supporting Information, section 1. At this time, many members of the Amber community had a myriad of scripts and programs to do basic MD trajectory processing and analysis, written in various languages with significant code duplication. This set of tools included both home-grown scripts and those distributed with Amber.[2a]

While experimenting with adding analysis capabilities to RDPARM, PTRAJ was designed as an extensible tool with the intention of it becoming the central place to aggregate Amber MD trajectory analysis tools to facilitate MD trajectory analysis. PTRAJ was built within RDPARM—the two source codes are merged with different run time behaviors determined by the name of the executable—promoting reuse of common functionality and providing the underlying analysis code with a single interface to the various trajectory and topology formats. The code is fairly well documented and attempts to be modular and extensible by others. In fact, the code for PTRAJ has been modified and extended by several different authors over the years, leading to new functionality (time correlation functions, matrix analyses, etc.). Moreover, PTRAJ grew to support other MD trajectory formats beyond PDB and Amber, including CHARMM[9] PSF topology and DCD MD trajectory formats (among others). Despite the attempts to abstract the functionality, extending the code is not particularly easy since the RDPARM and PTRAJ codes were written primarily in C, leading to an overly complicated code base. C was chosen as the primary language since C++ was not yet a reliable standard when the RDPARM and PTRAJ programs were initially written. The complicated code base led to code fragmentation and duplication, and memory leaks are still present. Moreover, PTRAJ was not written with speed or efficiency in mind; more important was simplicity, functionality, generality, and ease of implementation. When the software was originally written, the rate determining step of biomolecular simulation was MD trajectory generation, not analysis, and the simulations tended to be relatively short (up to thousands of frames in 1−2 ns length simulations that took months to generate). PTRAJ was also initially designed with the idea of only processing single sets of input trajectory data, with a single output trajectory file, using a single parameter/topology file. For example, it is not readily possible to calculate the Root-Mean-Square Deviation (RMSD) of a coordinate frame to a reference structure with a different topology (e.g., a mutant protein structure or a related receptor with a different ligand bound). Moreover, PTRAJ tends to output derived data as raw text files necessitating further postprocessing of the data with graphing programs to display the results. Although PTRAJ has served the computational chemistry community well for about two decades, there are several factors that drove a complete overhaul of the code.

This overhaul was the development of CPPTRAJ, a rewrite of PTRAJ in C++. Although certain portions of the PTRAJ code are used in CPPTRAJ—often initially cut-and-paste and then recoded and optimized—and elements of the PTRAJ design philosophy have been retained, the code base has been rebuilt from the ground up, with an eye toward improving calculation speed and making future additions to the code as simple as possible. As CPPTRAJ was developed, the intent was to be fully backward-compatible with PTRAJ commands and input files, although there are some differences, most notably in output formats (to allow direct visualization of results and to facilitate further postprocessing of the derived data). CPPTRAJ can read multiple topology files and reference structures, write multiple output trajectories (for which specific frames to be written can be specified), and strip topology files. In addition, the results from separate commands can be directed to the same data file (e.g., the results from two dihedral angle time series calculations like protein phi and psi angles can be written to one file), and there is native support for compressed files along with many other improvements. Overall, CPPTRAJ shows a significant speedup compared to PTRAJ, particularly when processing Amber NetCDF trajectories. In addition, several commands have been parallelized with OpenMP[10] to take advantage of multicore machines for even more speedup.

GENERAL OVERVIEW

Although PTRAJ and CPPTRAJ have some limited interactive capability, they are mainly designed to be run as a batch job with predefined input outlining a series of commands to process and analyze the data. An example run might look like

```
ptraj GAAC.topo commands.in
```

where in this case, the file GAAC.topo defines the topology of an 18-mer DNA duplex in solution, and the input file (commands.in) is as follows:

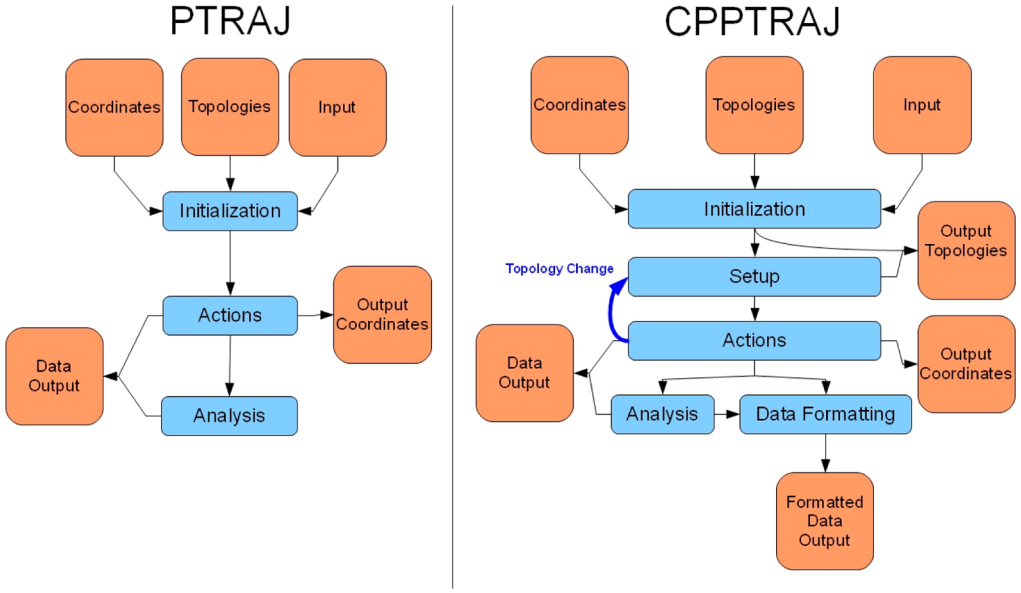


Figure 1. Overall program flow of PTRAJ (left) and CPPTRAJ (right). Orange boxes represent data; blue boxes represent phases of the program.

```
trajin GAAC.cdf.1
trajin GAAC.cdf.2
trajout GAAC-strip.crd nobox
center :1−18 mass origin
image origin center familiar byres :19−36
center :1−36 mass origin
image origin center familiar
rms first mass out rms time 50 :1−36
average avgpdb/avg.pdb pdb :1−36
strip :WAT
```

The above input file reads in two trajectory files (via **trajin**), performs centering and imaging of the coordinates (via **center** and **image** with respect to a particular atom selection, e.g., **:1−18** refers to residues 1−18; see Supporting Information, section 3 for an extended description of PTRAJ/CPPTRAJ mask syntax), performs an RMS fit to the first frame, outputs an average structure, strips (removes) residues named **WAT** (in Amber this is the standard name for water molecules), and outputs the resulting coordinates to another trajectory (via **trajout**). Note that the placement of the **trajout** command does not matter; coordinates produced via **trajout** are always output after all actions have processed a frame.

To better understand the trajectory processing and analysis, it is useful to describe the code flow which is shown in Figure 1 and is similar for both PTRAJ and CPPTRAJ. PTRAJ has three distinct phases: initialization, actions, and analysis. In the initialization phase, the topology file and an input file of commands are read in and parsed. The format of the topology file is automatically detected and is used to define the size of the system, the atom and residue names and numbers for atom selections, solvent, and other properties if present (e.g., atomic masses, charges, etc.). Next, input commands are parsed, either from a file or from standard input; this may include reading and checking reference coordinates or input trajectories. As with topology files, the format of coordinate/trajectory files is automatically detected. One current limitation of PTRAJ is that for many actions it is necessary to first predetermine how many frames will be processed for memory allocation, meaning that certain actions will not function properly if the total number of frames cannot be determined, as is the case for corrupted or certain compressed trajectories.

As the input file is read in, the various commands to be processed are put into a stack of "actions" which will be performed sequentially on each coordinate frame. At this stage, necessary memory allocation and setup is performed. After the initialization phase comes the actions phase where coordinate frames are read one by one from the input trajectories and processed by each action. Actions can generate data at this point, as well as change the "state" of the system (for example, if a **strip** command is specified to remove coordinates, all subsequent actions after the strip will use the new and truncated set of coordinates). In order to retain generality, there is little checking on the "actions" to see if they make sense; this can lead to issues if care is not taken to understand the implications. For example, trying to periodically image coordinates after an RMS fit rotates the periodic cell will alter the coordinates incorrectly by imaging with respect to the unrotated unit cell. On the other hand, by not imposing strict rules there is more flexibility. For example, atomic positional fluctuations (or B-factors) can be calculated either with free coordinates (including molecule rotation) or by prefitting to a common reference frame; in other words, the **atomicfluct** command does not require a particular fit to a reference frame. If an output trajectory was specified, output coordinates are written during this phase once all actions have processed the current trajectory frame. Once coordinate processing is complete, any accumulated data can be used to perform various types of analysis in the analysis phase. Last, any data or information not written during the actions phase is printed.

CPPTRAJ extends this flow into five distinct phases: initialization, setup, actions, analysis, and data formatting. During initialization, everything (including specification of topologies) can be prepared based on user input, again either from an input file or via standard input. All topology information and reference structures are read. Coordinate trajectories are prepared for reading, and actions/analyses to be performed are instantiated. The next two phases, setup and actions, comprise coordinate reading and data accumulation.

During the setup phase, all topology-dependent aspects of each action (e.g., atom mask parsing) are processed. When a trajectory with a different topology needs to be loaded, all actions are set up again for the new topology before continuing. During the actions phase, input trajectories are read in one frame at a time and processed by each action. As with PTRAJ, actions are performed in sequence on each coordinate frame that is read in, and data may be generated at this point. It should be noted that although some actions do output data in the actions phase, the majority do not for reasons of efficiency (see e.g. the "**secstruct**" command in the Discussion). In contrast to PTRAJ, output coordinates can be written out during action processing (as opposed to only after all actions are processed); this allows output of coordinates before modification from, e.g., an RMS fit. CPPTRAJ also has the ability to write out Amber topology files, either during the initialization or setup phases, which is particularly useful when actions modify topologies. As with PTRAJ, once coordinate processing is complete, any accumulated data can be used to perform analysis during the analysis phase. During the last phase, data formatting, any data slated for output is formatted and then written to disk.

Overall, the procedure followed by CPPTRAJ is similar to PTRAJ, with three notable exceptions: (1) Trajectories can have different topologies (thus actions require separate initialization and setup phases). (2) The number of frames does not need to be known for memory allocation in actions, as data set memory can be allocated dynamically. (3) There is now a distinct output phase after trajectory processing and analysis in which data can be formatted. This latter phase is intended to give the user more output format options, as well as to eliminate the need for some common postprocessing steps. Most data generated from actions can be output in one of three formats (essentially any output specified with the "**out**" keyword): standard data (data in whitespace-delimited columns), XMGRACE (a freely available graphing program for X-windows, http://plasma-gate.weizmann.ac.il/Grace/), and Gnuplot contour map (another freely available graphing program, http://www.gnuplot.info/). The output file type is detected from the filename extension. Data from separate actions can be output to single or multiple files in any combination desired by the user.

For example, consider the following CPPTRAJ input:

```
parm DPDP.parm7
trajin DPDP.nc
distance end_to_end :1:22 out standard.dat
radgyr RoG nomax out standard.dat
```

This input reads in a single trajectory and performs two actions, a distance calculation (named **end_to_end**) and a radius of gyration calculation (named **RoG**), with both actions performing output to the file "**standard.dat**". In PTRAJ, this would result in the radius of gyration action overwriting the results of the distance action. However, in CPPTRAJ the results of both actions are written to the same file in separate columns:

#Frame	end_to_end	RoG
1	15.8933	7.7659
2	13.6477	7.9568
3	14.8465	7.7254

This output format is considered "standard data" output in CPPTRAJ, which is the default. Note that the column headers are taken from the name specified with the action; default names are assigned if not specified.

The format of the output can be changed simply by changing the filename extension. For example, to output in XMGRACE format, the user can simply change the name of the output file to "standard.agr", or to obtain Gnuplot map output, the name can be changed to "standard.gnu. The results of this are shown in Figure 2. The figures shown are exactly what is rendered by

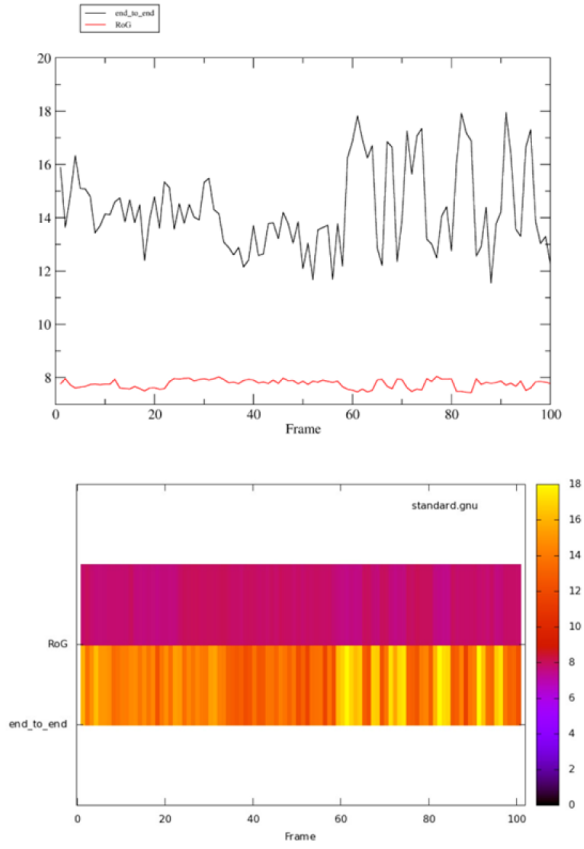


Figure 2. XMGRACE (top) and Gnuplot map (bottom) output of CPPTRAJ showing how data from separate actions (in this case, a distance and radius of gyration calculation) may be combined. The plots were created using XMGRACE 5.1.23 and GNUPLOT 4.4 on the data output from CPPTRAJ and were not modified in any way.

XMGRACE and Gnuplot directly from the CPPTRAJ output; i.e., no alterations were made, and no massaging of the files was required to get axis labels and legends.

OVERVIEW OF FILE PROCESSING

Both PTRAJ and CPPTRAJ handle a wide variety of topology and trajectory file formats, shown in Table 1. Detection of the file type is automatic and proceeds via "magic numbers" (i.e., hex signature) for binary files and via other characteristics for ASCII files (e.g., PDB keywords, etc.). In PTRAJ, trajectory files can be read in compressed (zip/gzip/bzip2) format via external program calls. In CPPTRAJ, both topology and trajectory files can be read and written in compressed (gzip/bzip2) format via internal calls (to the freely available zlib/libbz2 libraries respectively). In addition, output data files can be written compressed.

Topology information can be read from Amber Topology, PDB, Charmm PSF, and (CPPTRAJ only) Mol2 files. Additionally, in CPPTRAJ atom bonding information can be

Table 1. File Formats Supported by PTRAJ/CPPTRAJ

file format	type[a]	supported by
Amber Topology	Parm	both
PDB	Coord/Parm	both
Charmm PSF	Parm	both
Mol2	Coord/Parm	CPPTRAJ
Amber Trajectory	Coord	both
Amber NetCDF	Coord	both
Amber Restart	Coord	both
Amber NetCDF Restart	Coord	CPPTRAJ
Charmm DCD	Coord	both
Scripps Binpos	Coord	PTRAJ

[a]Parm = topology file; Coord = coordinates/trajectory.

generated (via the "**bondsearch**" keyword) for topology files that do not already have such information for use in actions which require it (e.g., the linear combination of pairwise overlaps surface area algorithm[11]). CPPTRAJ also has the ability to write Amber topology files, which can be particularly useful when the topology has been modified such as after a "**strip**" command; see the Commands Overview section for more details.

PTRAJ and CPPTRAJ can read and write in several trajectory formats: SCRIPPS BINPOS (PTRAJ only), Amber Coordinates (ASCII), Amber Restart, Amber NetCDF trajectory, Charmm DCD, PDB, and (CPPTRAJ only) Amber NetCDF restart and Mol2 files. Trajectory files can be read in either for processing or for use as reference frames for certain actions (e.g., RMSD). Trajectory files can be read in using user-specified start, stop, and offset frame numbers. In addition, users can choose to process frames at a specific temperature from an ensemble of trajectories generated from Amber replica exchange molecular dynamics (REMD) via the "**remdtraj remdtrajtemp**" keywords. Replica trajectories are automatically searched for one given replica filename based on the assumption of a numeric file suffix. Output trajectories are written after coordinate processing. Multiple input trajectories can be written to a single output trajectory. Existing trajectories can also be appended.

CPPTRAJ also has some additional trajectory processing functionality worth mentioning. For processing replica trajectories, users can explicitly specify trajectory file names in addition to the automatic file name search. A given replica ensemble can also be converted to a temperature ensemble in one pass using the "**remdout**" keyword. When reading in trajectories, the "**last**" keyword can be used if the stop frame is not known in advance, and the "**lastframe**" keyword can be used to explicitly choose the final frame of a trajectory. When writing out trajectories, users can specify which input frames to be output. Also, multiple output trajectories can be specified, and trajectories can be written out during action processing (as opposed to only after in PTRAJ) with the "**outtraj**" command. Examples are shown in the Supporting Information, section 5.

PTRAJ/CPPTRAJ COMMANDS OVERVIEW

Both PTRAJ and CPPTRAJ can calculate various geometric quantities, including distance, angle, dihedral, radius of gyration, and pucker; a complete list of commands is provided in the Supporting Information, section 2. The distance calculations can perform imaging for both orthorhombic and non-orthorhombic cells; by default the shortest imaged distance is used, although this behavior can be changed. Radius of gyration is defined by

$$\mathrm{Rg} = \sqrt{\frac{1}{N}\sum_{N}^{i=0}(r_i - r_\mathrm{m})^2}$$

where N is the number of atoms, r_i denotes atomic position, and r_m denotes the mean position of all atoms. Five-membered ring pucker can be calculated using the method of Altona and Sundaralingam[12] or the method of Cremer and Pople.[13] With the analysis utilities, averages and standard deviations can be reported, with proper cyclic averages being calculated for periodic values (e.g., torsions).

Both PTRAJ and CPPTRAJ employ a version of Kabsch's algorithm[14] for calculating the best-fit RMSD of a structure to a reference structure, although there are several small differences in the implementation between the two programs. Specifically, CPPTRAJ allows the specification of separate masks for the target and reference, allowing some more flexibility. CPPTRAJ also contains an additional mode that automatically calculates the no-fit RMSD of specified residues after an overall RMS-fit has been performed (enabled with the "**perres**" keyword), which gives an idea of the motion of individual residues within the overall reference frame.

PTRAJ and CPPTRAJ have the ability to calculate protein secondary structure using the method of Kabsch and Sander[15] (also known as the DSSP method). Both programs can output both secondary structure for each residue per frame as well as the average secondary structure of each residue over all frames. In addition, CPPTRAJ can output these results in native XMGRACE and/or Gnuplot format, shown in Figure 3.

PTRAJ and CPPTRAJ also have several commands for modifying coordinates and/or topology. The "**strip**" command removes user-specified atoms from coordinates and topology, which is useful when, for example, removing solvent molecules from a trajectory. In addition to this, CPPTRAJ can output a corresponding stripped Amber topology file that matches the stripped coordinates. Similar to "**strip**," the "**closest**" or "**closestwaters**" command can be used to keep only a user-specified number of waters near a specific area of solute (e.g., near a bound ligand), and solvent can be redefined using the "**solvent**" command for the generality of keeping nearby molecules. As with "**strip**," CPPTRAJ can also be used to output a corresponding Amber topology. CPPTRAJ also has an additional command, "**unstrip**," which can be used to restore a topology to its original state, which can be used for example to separate a complex into separate receptor and ligand trajectories in one pass; indeed, CPPTRAJ is used by the MM-PBSA method in Amber (MMPBSA.py python script) for this very purpose.[16]

For trajectories with periodic boundary conditions, PTRAJ and CPPTRAJ can perform imaging in order to bring molecules outside the primary unit cell back into the primary unit cell. This is usually done by centering molecules of interest prior to imaging. Imaging can be performed for both orthorhombic and nonorthorhombic cells. Nonorthorhombic cell types include all of the general triclinic unit cells, and imaging can be performed to the triclinic unit cell shape (which often looks like a slanted rectangle, see Figure 4) with the "**triclinic**" keyword or to the more familiar or spherical shape that displays the molecules such that the single periodic image closest to the center of the unit cell is displayed (with the "**familiar**" keyword). PTRAJ defaults to the triclinic, for historical reasons, whereas

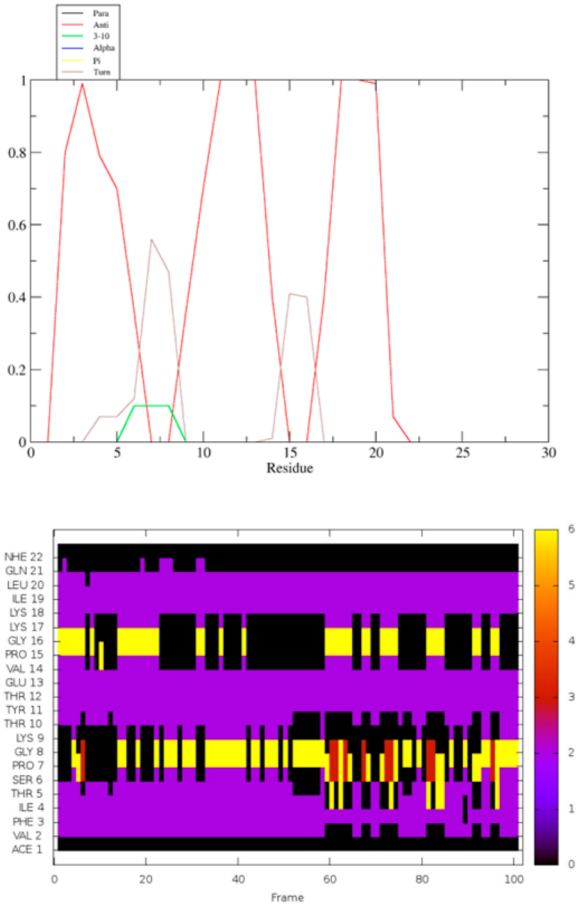


Figure 3. CPPTRAJ native XMGRACE (top) and Gnuplot (bottom) output for the "secstruct" (DSSP) command for a short trajectory of a model β sheet peptide. The plots represent exactly what is output from CPPTRAJ and were not modified in any way. In the XMGRACE output, the average secondary structure content (y axis) for each residue (x axis) is shown. In the Gnuplot output, each residue (y axis) is assigned a secondary structure type (0 = no structure, 1 = parallel beta, 2 = antiparallel beta, 3 = 310 helix, 4 = α helix, 5 = PI helix, 6 = turn) for each frame (x axis).

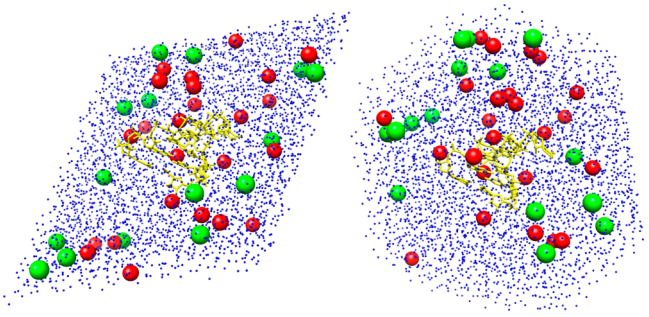


Figure 4. An RNA tetraloop (yellow) with surrounding water solvent (blue, only showing the oxygen atoms) and ions (Na^+ in red and Cl^- in green) showing equivalent periodic unit cells imaged either to the triclinic unit cell (left) or more spherical or familiar imaging (right). Image created with VMD 1.9.1.[3]

CPPTRAJ defaults to familiar, which is the common imaging mode in Amber for truncated octahedral unit cells.

In addition to imaging by molecule, PTRAJ can image by atom, by residue, or according to a user-specified mask expression and can build nearby unit cells through user-specified offsets to create larger unit cells or to better understand packing. For example to output coordinates one unit cell over in each direction, the command would be "**image xoffset 1 yoffset 1 zoffset 1**". Both PTRAJ and CPPTRAJ have the ability to "**unwrap**" coordinates, i.e. perform the reverse of imaging, which is necessary if one is interested in diffusive properties. CPPTRAJ also has an action called "**autoimage**," which performs centering/imaging with no additional input from the user and will not separate molecular complexes (such as DNA or other multimers), which was occasionally an issue with PTRAJ imaging.

VECTOR/MATRIX ANALYSIS

PTRAJ and CPPTRAJ have several commands which allow users to track and analyze vectors and matrices generated from input coordinates. In addition to storing simple atom coordinate vectors (e.g., between a nitrogen atom bonded to a hydrogen atom), both principal axis vectors and dipole vectors can be stored as well. Vectors can also be stored for use with the Isotropic Reorganizational Eigenmode Dynamics (IRED) approach of Prompers and Brüschweiler.[17] Vectors are stored as 2 sets of Cartesian coordinates (magnitude and origin). Auto or cross time correlation functions can be calculated for vectors using spherical harmonics addition theorem and a fast-Fourier transform (FFT) approach via cross-correlation/Wiener-Khinchin theorem.

Several types of matrices can be calculated including distance, covariance, mass-weighted covariance, correlation, distance-covariance, IRED, and Isotropically Distributed Ensemble Analysis.[18] Distance matrices record the average distance between atom pairs. (Mass-weighted) covariance matrices record the coordinate covariance. Correlation records the correlation between atom vectors, and distance-covariance records the covariance between atom-pair distances. IRED matrices use previously defined IRED vectors to generate an IRED matrix.

Using the "**analyze matrix**" command, symmetric matrices can be diagonalized, and the eigenvectors and eigenvalues calculated and output. If all eigenvalues/eigenvectors are desired, the **dspev**() routine from the LAPACK math library[19] is used for the calculation. If desired, only the eigenvalues may be computed. If a subset of the eigenvectors/eigenvalues is desired, the **dsaupd**()/**dseupd**() routines from the ARPACK math library[20] are used. If the matrix being analyzed is an IRED matrix, IRED order parameters can be calculated and output. If the matrix being analyzed is mass-weighted covariance, quasi-harmonic analysis (entropy, heat capacity, and internal energy) can be performed using the standard statistical mechanical formula for an ideal gas. Eigenvectors from covariance matrices may also be reduced according to the procedure of Abseher and Nilges,[21] which can be useful for comparing eigenvectors from Cartesian-space to those in distance space. Eigenmodes generated from "**analyze matrix**" can be further analyzed to calculate RMS fluctuations, displacements of Cartesian coordinates along mode directions, or dipole−dipole correlation functions. The modes can also be read back in and coordinate frames projected onto them (using the "**project**" command) to calculate how much that frame contributes to each mode. Principal component (or quasi-harmonic) analysis is a common use of the matrix analysis facilities. A typical workflow would involve calculating the covariance matrix (mass-weighted for quasi-harmonic analysis), diagonalizing it to get the eigenmodes, then projecting frames onto those

eigenmodes to obtain the values of the principal components at each frame. Examples are shown in the Supporting Information, section 4.

PTRAJ-SPECIFIC COMMANDS

Advanced Cluster Analysis. While CPPTRAJ has the ability to perform cluster analysis on input coordinates via a hierarchical agglomerative approach with single, average, or complete linkage, PTRAJ is able to perform clustering using a much greater variety of clustering algorithms, including top-down splitting, Bayesian, and COBWEB among others.[22] In addition, in PTRAJ clustering one has the ability to "**sieve**" input trajectories in order to reduce total computation time. During a cluster sieve, clustering is initially performed on a smaller subset of the input coordinates, after which the remaining input coordinates are added to resulting clusters based on their closeness to the cluster centroid.

Due to the complexity of the clustering code in PTRAJ, it has not yet been fully ported to CPPTRAJ, although this is planned for future versions of CPPTRAJ.

Hydrogen Bond Facility. Although both PTRAJ and CPPTRAJ both have commands to track hydrogen bonds, the PTRAJ hydrogen bonding facility differs significantly from CPPTRAJ. In PTRAJ, hydrogen bond donors and acceptors are defined prior to an "**hbond**" command with the "**donor**" and "**acceptor**" keywords. Note that in PTRAJ the definition of hydrogen bond donor and acceptor are reversed with respect to standard conventions; the electron pair "acceptor" is bonded to hydrogen and the "donor" is the atom to which the hydrogen bond is formed (i.e., in PTRAJ a "donor" can be thought of as donating electrons to the hydrogen atom). This has not been changed in order to preserve backward compatibility. Solute donors can either be specified using a mask or by giving a residue and atom name, or by specifying a mask. Solute acceptors can be specified by giving residue, heavy atom, and hydrogen atom names, or by giving heavy atom and hydrogen atom masks. This information is then used by a subsequent "**hbond**" command. Solute−solute, solute−solvent, and solvent−solvent hydrogen bonds can all be tracked. A hydrogen bond is considered formed when distance and angle cutoff criteria are met. If desired, the user can disable the angle criterion. PTRAJ can also be used to track particle−particle interactions by specifying the same selection for the "**acceptor**" specification twice, for example to track ion interactions with a typical donor. In addition, PTRAJ allows definition of **solventdonor** and **solventacceptor** hydrogen bonds, for example to track generic solvent interactions (e.g., if any solvent interacts in contrast to each specific solvent molecule and the definition of solvent is general). An example script is shown in the Supporting Information, section 6. At the end of coordinate processing, a summary of the average occupancy, distance, and angle of each hydrogen bond found is output. Note that in PTRAJ, hydrogen bond angles are reported as deviations from linear (i.e. as 180° - ANGLE). The time-series of each hydrogen bond can optionally be saved, from which details on hydrogen bond lifetimes can be calculated. PTRAJ will also report when solute residues are bridged by a single water molecule.

CPPTRAJ-SPECIFIC COMMANDS

CPPTRAJ has several actions not available in PTRAJ. The solvent accessible surface area of a molecule can be calculated using either the Connolly method[23] or the LCPO method[11] (with the "**molsurf**" and "**surf**" commands, respectively). Although both PTRAJ and CPPTRAJ can use distance-based masks that use reference coordinates, CPPTRAJ can also make use of a distance-based mask that updates each frame via the "**mask**" command, which can be used for example to write separate PDB files of all water molecules within a certain distance of a ligand (as opposed to the "**closest**" command, which only outputs a fixed number of molecules). CPPTRAJ can also calculate J-coupling values from dihedral angles according to the procedure of Chou et al.[24] or Perez et al.[25]

CPPTRAJ comes with many test cases that also serve as examples of how to run various commands. These test cases are part of the larger AmberTools distribution; when AmberTools is installed, they are located in the directory $AMBERHOME/AmberTools/test/cpptraj.

Automatic Imaging. Imaging a trajectory could prove to be challenging with PTRAJ for systems with more than one nonsolvent molecule (e.g., DNA duplexes, receptor−ligand complexes, protein tetramers, etc.). The "**autoimage**" command in CPPTRAJ was designed to perform centering and imaging in one step with minimal input from the user. The command functions by dividing all molecules into three regions: anchor, mobile, and fixed. The command will pick defaults for each region, although any of the regions can be chosen by the user via mask expressions. The anchor region is made up of a single molecule that will be centered either to the box center or the coordinate origin; all other molecules are imaged with respect to the anchor molecule. By default the first molecule is chosen as the anchor. The mobile region is made up of molecules that can be imaged freely; by default all solvents and ions are chosen to be mobile. The fixed region is made up of all remaining molecules; molecules in this region are imaged only if the imaged position is closer to the anchor molecule. An example of this command is shown in the Supporting Information, section 5.7.

New Hydrogen Bonding Facility. Like PTRAJ, CPPTRAJ has the ability to keep track of hydrogen bonds formed over the course of a trajectory. However, there are several important differences from the PTRAJ implementation. Unlike PTRAJ, where hydrogen bond donors and acceptors were specified with separate commands, in CPPTRAJ, hydrogen bond donors and acceptors are specified with the "**hbond**" command as masks. Note that unlike PTRAJ, the definitions of hydrogen bond donor and acceptor follow standard conventions (hydrogen bond donors consist of a heavy atom and hydrogen; hydrogen bond acceptors consist of a single atom). Hydrogen bond donors and acceptors can be searched for automatically following the simplistic criterion that donors are considered to be N, O, and F atoms bonded to hydrogen, while acceptors are considered to be N, O, and F atoms with no bonded hydrogen atoms. Donors and/or acceptors can also be explicitly specified with "**donormask**" and "**acceptormask**" keywords, respectively. As with PTRAJ, a distance and angle criterion is used to determine when a hydrogen bond is present, and the average occupancy, distance, and angle of each hydrogen bond found is output. Unlike PTRAJ however, only solute−solute hydrogen bonds are searched for, and there is no option to record the time series (and hence get lifetimes) of hydrogen bonds, although these features are planned for future releases of CPPTRAJ. An example of this command is shown in the Supporting Information, section 5.8.

Nucleic Acid Structure Analysis. CPPTRAJ has the ability to calculate basic nucleic acid structure parameters. Base pair parameters (shear, stretch, stagger, buckle, propeller twist, and opening), base pair step parameters (shift, slide, rise, tilt, roll, and twist), and helical parameters (X-displacement, Y-displacement, rise, inclination, tip, and twist) are calculated using the same procedure employed by 3DNA[26] using reference frame coordinates from Olson et al.[27] Base pairing is determined on a per frame basis. First, each base is identified and assigned a standard reference frame. Next, for each base, potential base-pairing partners are determined based on how close their reference frames are and whether they are hydrogen-bonded; currently only base pairs with standard Watson−Crick hydrogen bonding patterns are recognized. CPPTRAJ has some ability to recognize nonstandard/modified nucleic acid bases through a user-specified argument which attempts to map the nonstandard base to one of the five standard bases. Currently, this works as long as the modified base contains at least the same heavy atoms as the standard base to which it is being mapped. An example of this command is shown in the Supporting Information, section 5.9.

Rotational Diffusion Tensor. The rotational diffusion properties of a molecule can be calculated with CPPTRAJ using the "**rotdif**" command, which determines the diffusion tensor with both small and full anisotropy. The procedure followed is briefly described here; for further details, refer to the original implementation by Wong and Case.[28] First, a user-specified number of random vectors are generated and rotated using rotation matrices obtained from RMS fitting the trajectory of the target molecule with a suitable reference (typically the average structure). The correlation function C_l for each vector is then determined using a Legendre polynomial P_l of order l = 1 or l = 2, specified by the user (default is 2):

$$C_l(\tau) = \sum_{\tau}^{i=0} \sum_{N-i}^{j=0} P_l[n_j \cdot n_{j+i}]$$

where τ denotes the maximum lag time, n represents the rotated vector, i and j denote different times, and N is the total number of frames. Since trajectories are of finite size, τ should be somewhat less than N in order to limit statistical errors. The average (or "effective") correlation time for a vector $T_l(n)$ can be determined from the integral over the calculated correlation function; in CPPTRAJ, this is accomplished by first creating a cubic spline mesh, then performing simple integration via the trapezoid rule. This can then be used to calculate the local effective diffusion constant $d_{loc}(n,l)$ for that vector using an iterative procedure:

$$d_{loc}(n, l) \equiv \frac{1}{l(l+1)T_l(n)}$$

Once the local effective diffusion constants for each vector have been determined, they can be used to determine a tensor Q by solving

$$d_{eff} = A^T * Q$$

where d_{eff} is a column vector composed of the local diffusion constants and A^T is a 6 by N matrix, the rows of which are composed of n vector components. Q is related to the diffusion tensor D by

$$Q = \frac{3D_{av}I - D}{2}$$

The d_{eff} equation can be solved for Q (and therefore D) using singular value decomposition; in CPPTRAJ this is done via an external call to a LAPACK routine (**dgesvd**). Diagonalization of the diffusion tensor to yield principal components and axes is also accomplished with a LAPACK routine (**dsyev**).[19] The diffusion tensor in the full anisotropic limit is then found using a downhill simplex minimization scheme that uses the diffusion tensor in the small anisotropic limit as initial input. An example of this command is shown in the Supporting Information, section 5.10.

RMSD Autocorrelation. Time correlation functions are useful tools that allow discernment of patterns that may be hidden in a given set of data. A property that one may be interested in calculating the time correlation function for is RMSD, in order to determine the similarity of structures in a trajectory over different time scales and to assess convergence to the average structure. However, one cannot directly calculate the time correlation function of a series of RMSD values and obtain this, as the difference between two RMSD values may not be indicative of how similar the structures are. For example, suppose one has a trajectory of a small peptide which starts in a helical conformation, then adopts both hairpin and extended conformations. With respect to the starting structure, the hairpin and extended conformations may have similar RMSD values, although they are obviously different structures.

To calculate the similarity of structures over different time scales, we have developed the RMSD average correlation method, or "**rmsavgcorr**". This method calculates the autocorrelation of RMSD as the average RMSD of coordinate frames which have been averaged over sliding time windows of a certain size:

$$\mathrm{RAC}(\tau) = \frac{\sum_{t=0}^{N} \mathrm{RMSD}(\mathrm{AvgCrd}(t, t+\tau))}{N - \tau + 1}$$

where τ is the window size (ranging from 1 to N), N is the total number of frames, AvgCrd(t, t + τ) is the average coordinates from frames t to t + τ, and RMSD(.) represents the calculation of best-fit RMSD of the given frame to reference AvgCrd(0, τ). This means that RAC(1) is just the average RMSD to the first frame, and RAC(N) is always 0.0.

Atom Mapping. In some cases, two structures of the same molecule may exist with different atom ordering. This can occur for example when generating a ligand with two different programs; hydrogen atoms may be placed at the end of one file, whereas they may be placed close to their bonded heavy atom in another. The two structures then become impossible to compare using, e.g., the RMSD algorithm in PTRAJ/CPPTRAJ, since it is expected that the atom ordering between the target and reference structure matches. To address this issue, the "**atommap**" command was created for CPPTRAJ. The "**atommap**" command employs a very simple algorithm which attempts to map a target structure onto a reference structure (i.e., solve in an approximate way the maximum common subgraph isomorphism problem).[29]

The central idea of the algorithm is to assign each atom in the target and reference a unique ID based on its chemical environment and from these IDs attempt to create a map between the target and reference. To create the unique ID, first each atom is assigned a single character based on its atomic element, so carbon becomes C, hydrogen becomes H, etc. The next step is to assign each atom what is called an AtomID, which is made up of the single character plus the characters of

the atoms it is bonded to. So for example, the α carbon in an alanine side chain would receive an AtomID of CCCHN since it is bonded to a carbonyl carbon, a beta carbon, an alpha hydrogen, and an amide N. The AtomID is then combined with the AtomID of all bonded atoms and sorted in alphabetical order to form the unique ID. In the case of alanine, the AtomIDs of the previously mentioned atoms bonded to the α carbon are CCOO, CCHHH, HC, and NCHH, so the total unique ID for the α carbon is CCCCCCCCCHHHHHH-HNNOO. The unique ID of an atom therefore reflects the local chemical environment of the atom. Because the chemical environment of an atom may not truly be unique (which is the case with symmetric atoms for example) each unique ID is checked to see how many times it is repeated in the molecule. If it is not repeated, it is marked as truly unique.

The atom mapping procedure is illustrated in Figure 5. Once the truly unique atoms have been found in the target and reference molecules, these atoms are mapped to each other (Figure 5, blue atoms). This constitutes an initial guess which the rest of the algorithm uses to map the structures. Once the initial unique atoms have been mapped, the algorithm proceeds and attempts to map the remaining unmapped atoms. The first step is to identify atoms that are the only ones of their kind bonded to a mapped atom (e.g., a lone hydrogen bonded to a mapped carbon atom, Figure 5, red atom). To ensure that chirality in molecules is preserved, the second step is to identify atoms that are bonded to partially mapped chiral centers (i.e., the chiral atom and two atoms it is bonded to are mapped). These atoms are then mapped by matching dihedral angles formed between them and the three mapped atoms (Figure 5, orange atoms). If any atoms are mapped in this way, the algorithm returns to the first step. The final step is to map atoms based on their element and bonding to previously mapped atoms (Figure 5, yellow atoms). The atoms in this stage are usually symmetric in some way. If atoms are mapped in this way, the algorithm then returns to the first step (Figure 5, green atoms); otherwise mapping is complete.

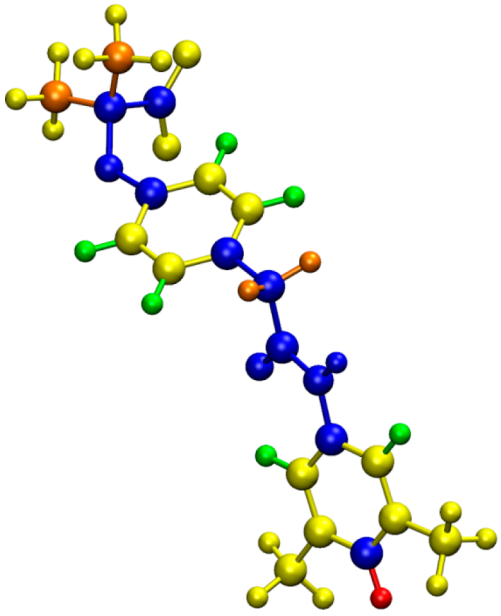


Figure 5. Illustration of "atommap" stages. (1) Blue atoms, initial unique assignment. (2) Red atom, only atom bonded to a mapped atom. (3) Orange atoms, mapped based on chirality. (4) Yellow atoms, assignment based on element and bonding to mapped atoms. (5) A return to step 2, atoms can now be mapped as they are the only atoms bonded to previously mapped atoms. Image created with VMD 1.9.1.[3]

When the molecule is highly symmetric, it is possible that no unique atoms can be found to make an initial map. When this occurs, the algorithm attempts to guess an initial mapping based on atoms whose unique IDs are duplicated only once, preferring chiral atoms. The algorithm performs mapping using each potential pair as an initial guess and ranks each guess based on RMSD of mapped atoms. The mapping which produces the lowest RMSD is then used.

After mapping is complete, several options are available. The map can be used to reorder an input trajectory based on the target so that it matches the reference, or only the map itself or the RMSD of mapped atoms can be printed if desired. In cases where not all atoms can be mapped (either due to different numbers of atoms in the target and reference or incomplete mapping), structures can be modified to only include mapped atoms. Despite the relative simplicity of the algorithm, it can be quite effective, at least for small molecules. In a recent docking study in which the atom ordering of crystal poses did not match that of the starting poses, the CPPTRAJ "**atommap**" algorithm was able to map all 85 ligands, including structures where the protonation state changed and highly symmetric structures.[30] An example of this command is shown in the Supporting Information, section 5.11.

■ ADDING NEW FUNCTIONALITY

Both PTRAJ and CPPTRAJ were developed with the idea that the addition of new functionality (actions, analyses, trajectory formats) should be easy. In PTRAJ, all actions were located in the "actions.c" file and all analyses in the "analysis.c" file. The addition of new actions/analyses was done via the addition of a function to the appropriate file, adding a pointer to that function to the appropriate array in the "dispatch.c" file, and then adding an entry to the ptrajSetup() routine in the "ptraj.c" file. The addition of new trajectory file formats was slightly more complicated, requiring the modification of several routines in the "ptraj.c" file, as well as potentially adding new functions to the "trajectory.c" file, although comments in the code guide potential developers on how to do this.

A goal of CPPTRAJ was to make the addition of new functionality more straightforward. To this end, the addition of new actions, analyses, and trajectory formats requires implementing a new class that inherits from the appropriate base class type (Action, Analysis, and TrajectoryIO, respectively), then adding an entry for the new type in the appropriate container (ActionList, AnalysisList, and TrajectoryFile, respectively). The advantage of this approach is that the code for different actions/analyses/trajectory formats is compartmentalized, enhancing code readability and making future code maintenance easier. The CPPTRAJ codebase is itself extensively documented in Doxygen format (http://www.doxygen.org), and the code is distributed with a developers guide that has step-by-step instructions for adding new actions among other information.

■ DISCUSSION OF PTRAJ/CPPTRAJ PERFORMANCE

There are several differences between PTRAJ and CPPTRAJ, both in the implementation of underlying algorithms and general code layout, which are worth noting as they contribute to the generally better performance of CPPTRAJ vs PTRAJ.

One major difference is that by design CPPTRAJ attempts to exploit the locality of reference as much as possible, i.e. to make data access as efficient as possible.[31] This tends to be a feature when programming with an object-oriented language such as C++ using modern compilers, as classes tend to keep both code and the target data together. In CPPTRAJ, this strategy was also employed for the storage of coordinates. Whereas in PTRAJ, X, Y, and Z coordinates are stored in three separate arrays, in CPPTRAJ, X, Y, and Z coordinates are stored sequentially in a single array. This results in significantly improved performance when processing Amber NetCDF trajectories in particular, since such trajectories have their coordinates stored in a similar fashion.

Another difference is in the implementation of atom masks. In PTRAJ, atom masks are stored as an integer array of size N_{Atom} (where N_{Atom} is the total number of atoms in the system), set to 1 if the atom is selected and 0 if not. This means that for example a routine calculating the center of mass of atoms in a mask would have a loop that always executes N_{Atom} times, with a conditional inside the loop determining whether an atom is selected or not (although it should be noted there are some actions in PTRAJ which do have special cases where only one atom is selected). In contrast, CPPTRAJ employs two mask types. The first is similar to PTRAJ masks, where there is a character array of size N_{Atom}, set to "T" if the atom is selected and "F" if the atom is not selected. This is useful when one needs to know both selected and unselected atoms. However, it is far more common to only be interested in selected atoms, and so the second mask type in CPPTRAJ is an integer array of size $N_{Selected}$ (where $N_{Selected}$ is the total number of selected atoms), containing only the atom numbers of selected atoms. Typically $N_{Selected}$ is much smaller than N_{Atom}. So to use the previous example of a center of mass calculation, instead of having to execute N_{Atom} times, the loop only needs to be executed $N_{Selected}$ times with no conditional inside the loop. This also saves memory as only selected atoms are stored versus all atoms for a PTRAJ mask.

Although both PTRAJ and CPPTRAJ have parallelized code, the strategies employed in each case are radically different. For PTRAJ, the strategy was to parallelize trajectory reads, i.e. divide each trajectory among the number of threads being used, within an MPI[32] framework. So given a 1000 frame trajectory and two threads, thread 0 would read frames 0−499, while thread 1 would read frames 500−999. While this did at times result in a significant speedup and scaled reasonably well, there were several problems with this approach. The first was that as implemented the number of input frames was required to be a multiple of the number of threads; so for example it was not possible to read 753 frames with two threads. Another is that the performance has proved to be extremely dependent on the underlying file system; for some standard nonparallel file systems it was possible to see no speedup (or even a slowdown) when using multiple threads. Finally, this approach also requires that the trajectory be "seekable", i.e. the file can be opened at a specific frame; as such its use is currently restricted to Amber formatted and NetCDF trajectories.

To avoid these issues, it was decided that for CPPTRAJ only certain time-consuming actions would be parallelized using OpenMP.[10] As it is now quite commonplace for even desktop machines to have two or more cores, the simplicity of parallelizing time-consuming actions within a shared-memory framework makes OpenMP an attractive parallelization solution. In many cases, speedup can be achieved by simply using an OpenMP pragma in front of a key loop. Since the input trajectory is never split up, there is no postprocessing to recombine data from actions on different threads in the correct order.

Benchmarks. Benchmarks of several commonly used actions are shown for PTRAJ and CPPTRAJ in Table 2.

Table 2. Time Necessary to Complete Common Actions for 1000 Frames (Unless Otherwise Noted) in AmberTools 12 CPPTRAJ and PTRAJ, and Speedup of CPPTRAJ vs PTRAJ

command	CPPTRAJ (s)	PTRAJ-OPT (s)	speedup
angle	2.77	4.18	1.51
center	3.32	5.71	1.72
dihedral	2.78	4.20	1.51
DSSP	9.47	138.94	14.67
image (general triclinic)	4.46	7.97	1.79
image (truncated oct.)	8.44	16.12	1.91
pucker	2.80	4.21	1.50
radius of gyration	2.80	4.48	1.60
RMSD (best-fit)	3.60	19.23	5.34
running avg	6.48	27.46	4.24
strip	2.92	3.85	1.32
closest[a]	16.62	33.33	2.01
radial dist. fn[a]	65.96	125.06	1.90

[a]Only one frame processed.

Both programs were compiled with GNU compilers, version 4.5.1. In AmberTools, PTRAJ is currently built without explicit compiler optimizations turned on for historical reasons, so timings are reported for PTRAJ with compiler optimizations turned on (denoted "PTRAJ-OPT"). Turning on compiler optimizations for PTRAJ results in a speedup (versus unoptimized PTRAJ) of 1.39× on average. The trajectory processed was an Amber NetCDF trajectory (49 115 atoms). All actions processed 1000 frames, except for the "closest" and "radial" actions, which only processed one frame due to their time-consuming nature. The CPU used was an AMD Athlon 64 X2 4600+ (2.4 GHz).

For all commands, CPPTRAJ is faster than PTRAJ, with an overall average speedup of 3.15×. For relatively simple calculations such as **angle**, **dihedral**, **pucker**, **radgyr** (radius of gyration), **strip**, and **center**, the average speed-up for CPPTRAJ is 1.53×. Much of this is a result of the better handling of NetCDF files and improved locality of reference as mentioned in the previous section. Both image actions show slightly better speedups of 1.79× and 1.91×; this is due to faster handling of atom masks in CPPTRAJ, as well as vectorization of the nonorthogonal imaging code originally used in PTRAJ.

The **rms** action shows a much larger speedup of 5.34×. This is largely the result of two changes to the RMSD calculation from PTRAJ: (1) copies of the target and reference coordinate frames are made prior to the calculation based on atom masks, so that the RMSD calculation itself is only ever on the atoms of interest, eliminating the need for conditionals to check for selected atoms and/or mass information inside the RMSD calculation loop, and (2) the reference frame is set up and precentered once instead of each time the RMSD calculation is performed.

The **secstruct** action (DSSP analysis) has a huge speedup of 14.67×. Much of this is due to the fact that in PTRAJ **secstruct** data are output during trajectory processing, whereas in CPPTRAJ data are output after trajectory processing. This

illustrates how optimizing disk access can dramatically improve performance, although it does so at the expense of using more memory.

OpenMP Benchmarks. Benchmarks for several generally time-consuming actions which have been parallelized using OpenMP in CPPTRAJ are shown in Table 3. CPPTRAJ was compiled with GNU version 4.4.3 compilers. Trajectory is NetCDF 49 115 atoms and 15 022 water molecules; the number of frames used for each command was chosen so that the single thread benchmark would be around 60 s in length and is listed next to the command. This is run on a system with four AMD Opteron 6174 CPUs (48 cores total), 2.2 GHz.

Table 3. CPPTRAJ OpenMP Benchmarks for Certain Parallelized Actions

command	frames	# threads	time (s)	speedup	efficiency
closest 10 :1−268 first	3	1	49.68	1.00	1.00
		2	25.15	1.98	0.99
		4	12.94	3.84	0.96
		8	6.93	7.17	0.90
mask ″(:190−210 <:3.0) &:WAT″	10	1	44.76	1.00	1.00
		2	22.73	1.97	0.98
		4	11.77	3.80	0.95
		8	6.09	7.35	0.92
radial Radial.agr 0.5 10.0 :WAT@O	1	1	69.02	1.00	1.00
		2	35.00	1.97	0.99
		4	17.63	3.91	0.98
		8	9.48	7.28	0.91
secstruct out dssp.gnu	1000	1	11.90	1.00	1.00
		2	7.11	1.67	0.84
		4	5.75	2.07	0.52
		8	5.64	2.11	0.26
surf :1−268 out surf.dat	400	1	62.21	1.00	1.00
		2	32.52	1.91	0.96
		4	17.26	3.60	0.90
		8	9.36	6.65	0.83

In general, actions scale reasonably well up to eight threads. Routines which need to calculate many distances between single atoms (**closest**, **mask**, and **radial**) scale the best and remain around 90% efficient to eight threads. The **surf** action scales reasonably well out to eight threads, remaining 83% efficient. The reason **surf** does not scale as well is likely due to the fact that the underlying algorithm contains more nested loops than **closest**, **mask**, and **radial**. In contrast, the **secstruct** action does not scale well beyond two threads. This is probably because the parallelization of **secstruct** was done on the level of residues (rather than the level of distances), resulting in an often uneven distribution of calculations to each thread. It should be noted that with the exception of the **radial** command, the only coding required to parallelize these actions was a single OpenMP pragma in front of the outermost loop, and there remains significant room for improvement.

CONCLUSIONS

PTRAJ has provided the computational chemistry community the means to perform a wide variety of analyses on data generated from computational simulations for over almost two decades. However, dramatic increases in the size of trajectories being processed combined with the aging PTRAJ code-base necessitated a code rewrite. CPPTRAJ has been created to initially complement but eventually replace PTRAJ as the main analysis engine of the Amber software package. CPPTRAJ has significantly improved performance compared to PTRAJ, and the reorganization of the code should make future additions to the code easier. Although CPPTRAJ supports most of the functionality and syntax from PTRAJ, the code is not yet fully compatible.

One of the main goals for further development of CPPTRAJ is to enhance and increase the flexibility of data set handling. For example, although CPPTRAJ allows the creation of several types of matrices, users may want to create a custom matrix composed of data from various different sources (e.g., distances, angles, dihedrals, etc.) and perform various operations on that matrix (such as principal component analysis). Additional future aims for CPPTRAJ are to become fully backward-compatible with PTRAJ, add novel analysis capabilities, add more parallelization and improve upon existing parallelization, and increase the number of recognized topology and coordinate file formats.

ASSOCIATED CONTENT

Supporting Information

Summary of RDPARM functionality and output. Summary of PTRAJ/CPPTRAJ commands. Description of PTRAJ/CPPTRAJ atom mask selection syntax. PTRAJ/CPPTRAJ matrix/vector analysis scripts. Example CPPTRAJ scripts/commands. Example PTRAJ scripts. This material is available free of charge via the Internet at http://pubs.acs.org.

AUTHOR INFORMATION

Corresponding Author

*E-mail: daniel.r.roe@gmail.com (D.R.R.), tec3@utah.edu (T.E.C.III)

Author Contributions

The manuscript was written through contributions of all authors. Cheatham is the primary author of PTRAJ and Roe is the primary author of CPPTRAJ. All authors have given approval to the final version of the manuscript.

Funding Sources

NIH R01-GM081411 and NSF OCI-1036208.

Notes

The authors declare no competing financial interest.

ACKNOWLEDGMENTS

We would like to acknowledge locally Niel Henriksen, Rodrigo Galindo and Christina Bergonzo for extensive testing of the codes, and also the larger community of Amber developers, members of the Amber mailing list, and others for feedback on improvements to the codes. We would also like to acknowledge the Center for High Performance Computing at the University of Utah, NIH R01 GM-081411, NRAC XSEDE MCA01S027, and NSF/NCSA/U Illinois Blue Waters (PRAC OCI-1036208 and OCI 07-25070) for access to exceptional computational resources and support.

REFERENCES

(1) (a) Klepeis, J. L.; Lindorff-Larsen, K.; Dror, R. O.; Shaw, D. E. Long-timescale molecular dynamics simulations of protein structure and function. *Curr. Opin. Struct. Biol.* **2009**, *19* (2), 120−127. (b) Durrant, J. D.; McCammon, J. A. Molecular dynamics simulations and drug discovery. *BMC Biol.* **2011**, *9*, 71. (c) Schlick, T.; Collepardo-

Guevara, R.; Halvorsen, L. A.; Jung, S.; Xiao, X. Biomolecularmodeling and simulation: a field coming of age. *Q. Rev. Biophys.* **2011**, *44* (2), 191−228. (d) Wereszczynski, J.; McCammon, J. A. Statistical mechanics and molecular dynamics in evaluating thermodynamic properties of biomolecular recognition. *Q. Rev. Biophys.* **2012**, *45* (1), 1−25. (e) Perez, A.; Luque, F. J.; Orozco, M. Frontiers in molecular dynamics simulations of DNA. *Acc. Chem. Res.* **2012**, *45* (2), 196−205.

(2) (a) Pearlman, D. A.; Case, D. A.; Caldwell, J. W.; Ross, W. S.; Cheatham, T. E.; Debolt, S.; Ferguson, D.; Seibel, G.; Kollman, P. AMBER, a package of computer programs for applying molecular mechanics, normal mode analysis, molecular dynamics and free energy calculations to simulate the structure and energetic properties of molecules. *Comput. Phys. Commun.* **1995**, *91* (1−3), 1−41. (b) Case, D. A.; Cheatham, T. E., III; Darden, T.; Gohlke, H.; Luo, R.; Merz, K. M., Jr.; Onufriev, A.; Simmerling, C.; Wang, B.; Woods, R. J. The Amber biomolecular simulation programs. *J. Comput. Chem.* **2005**, *26* (16), 1668−1688.

(3) Humphrey, W.; Dalke, A.; Schulten, K. VMD - Visual Molecular Dynamics. *J. Mol. Graphics Modell.* **1996**, *14*, 33−38.

(4) Feig, M.; Karanicolas, J.; Brooks, C. L., III. MMTSB Tool Set: enhanced sampling and multiscale modeling methods for applications in structural biology. *J. Mol. Graphics Modell.* **2001**, *22* (5), 377−395.

(5) Michaud-Agrawal, N.; Denning, E. J.; Woolf, T. B.; Beckstein, O. MDAnalysis: A Toolkit for the Analysis of Molecular Dynamics Simulations. *J. Comput. Chem.* **2011**, *32*, 2319−2327.

(6) Yesylevskyy, S. O. Pteros: Fast and Easy to Use Open-Source C++ Library for Molecular Analysis. *J. Comput. Chem.* **2012**, *33*, 1632−1636.

(7) Romo, T. D.; Grossfield, A. LOOS: An extensible platform for the structural analysis of simulations. In *31st Annual International Conference of the IEEE EMBS*; IEEE: New York, 2009; pp 2332−2335.

(8) Tu, T.; Rendleman, C. A.; Borhani, D. W.; Dror, R. O.; Gullingsrud, J.; Jensen, M. Ø.; Klepeis, J. L.; Maragakis, P.; Miller, P.; Stafford, K. A.; Shaw, D. E. A Scalable Parallel Framework for Analyzing Terascale Molecular Dynamics Simulation Trajectories. In *Proceedings of the ACM/IEEE Conference on Supercomputing (SC08)*, Austin, TX, November 15−21, 2008; IEEE: New York: 2008; pp 15−21.

(9) Brooks, B. R.; Brooks, C. L., III; Mackerell, A. D., Jr.; Nilsson, L.; Petrella, R. J.; Roux, B.; Won, Y.; Archontis, G.; Bartels, C.; Boresch, S.; Caflisch, A.; Caves, L.; Cui, Q.; Dinner, A. R.; Feig, M.; Fischer, S.; Gao, J.; Hodoscek, M.; Im, W.; Kuczera, K.; Lazaridis, T.; Ma, J.; Ovchinnikov, V.; Paci, E.; Pastor, R. W.; Post, C. B.; Pu, J. Z.; Schaefer, M.; Tidor, B.; Venable, R. M.; Woodcock, H. L.; Wu, X.; Yang, W.; York, D. M.; Karplus, M. CHARMM: the biomolecular simulation program. *J. Comput. Chem.* **2009**, *30* (10), 1545−1614.

(10) Dagum, L.; Menon, R. OpenMP: An Industry-Standard API for Shared-Memory Programming. *IEEE Comput. Sci. Eng.* **1998**, *5* (1), 46−55.

(11) Weiser, J.; Shenkin, P. S.; Still, W. C. Approximate atomic surfaces from linear combinations of pairwise overlaps (LCPO). *J. Comput. Chem.* **1999**, *20*, 217−230.

(12) (a) Altona, C.; Sundaralingam, M. Conformational analysis of the sugar ring in nucleosides and nucleotides. A new description using the concept of pseudorotation. *J. Am. Chem. Soc.* **1972**, *94* (23), 8205−8212. (b) Harvey, S. C.; Prabhakaran, M. Ribose puckering - structure, dynamics, energetics, and the pseudorotation cycle. *J. Am. Chem. Soc.* **1986**, *108*, 6128−6136.

(13) Cremer, D.; Pople, J. A. A general definition of ring puckering coordinates. *J. Am. Chem. Soc.* **1975**, *97*, 1354−1358.

(14) Kabsch, W. A discussion of the solution for the best rotation to relate two sets of vectors. *Acta Crystallogr., Sect. A* **1978**, *34*, 827−828.

(15) Kabsch, W.; Sander, C. Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers* **1983**, *22* (12), 2577−2637.

(16) Miller, B. R., III; McGee, T. D., Jr.; Swails, J. M.; Homeyer, N.; Gohlke, H.; Roitberg, A. E. MMPBSA.py: An efficient program for end-state free energy calculations. *J. Chem. Theory Comput.* **2012**, *8*, 3314−3321.

(17) Prompers, J. J.; Bruschweiler, R. General framework for studying the dynamics of folded and nonfolded proteins by NMR relaxation spectroscopy and MD simulation. *J. Am. Chem. Soc.* **2002**, *124* (16), 4522−4534.

(18) Prompers, J. J.; Bruschweiler, R. Dynamic and structural analysis of isotropically distributed molecular ensembles. *Proteins* **2002**, *46* (2), 177−189.

(19) Anderson, E.; Bai, Z.; Bischof, C.; Blackford, J.; Demmel, J.; Dongara, J.; Du Croz, J.; Greenbaum, A.; Hammarling, S.; McKenney, A.; Sorensen, D. C. *LAPACK Users' Guide*, 3rd ed.; SIAM: Philadelphia, PA, 1999.

(20) Lehoucq, R. B.; Sorensen, D. C.; Yang, C. *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*; SIAM: Philadelphia, PA, 1998.

(21) Abseher, R.; Nilges, M. Are there non-trivial dynamic cross-correlations in proteins? *J. Mol. Biol.* **1998**, *279* (4), 911−920.

(22) Shao, J.; Tanner, S. W.; Thompson, N.; Cheatham, T. E., III. Clustering molecular dynamics trajectories. 1. Characterizing the performance of different clustering algorithms. *J. Chem. Theory Comput.* **2007**, *3*, 2312−2334.

(23) (a) Connolly, M. L. Solvent-accessible surfaces of proteins and nucleic acids. *Science* **1983**, *221* (4612), 709−713. (b) Connolly, M. Analytical molecular surface calculation. *J. Appl. Crystallogr.* **1983**, *16*, 548−558.

(24) Chou, J. J.; Case, D. A.; Bax, A. Insights into the mobility of methyl-bearing side chains in proteins. *J. Am. Chem. Soc.* **2003**, *125*, 8959−8966.

(25) Perez, C.; Lohr, F.; Ruterjans, H.; Schmidt, J. M. Self-consistent Karplus parametrization of 3J couplings depending on the polypeptide side-chain torsion chi1. *J. Am. Chem. Soc.* **2001**, *123* (29), 7081−7093.

(26) (a) Lu, X. J.; Olson, W. K. 3DNA: a software package for the analysis, rebuilding and visualization of three-dimensional nucleic acid structures. *Nucleic Acids Res.* **2003**, *31* (17), 5108−5121. (b) Babcock, M. S.; Pednault, E. P.; Olson, W. K. Nucleic acid structure analysis. Mathematics for local Cartesian and helical structure parameters that are truly comparable between structures. *J. Mol. Biol.* **1994**, *237* (1), 125−156.

(27) Olson, W. K.; Bansal, M.; Burley, S. K.; Dickerson, R. E.; Gerstein, M.; Harvey, S. C.; Heinemann, U.; Lu, X. J.; Neidle, S.; Shakked, Z.; Sklenar, H.; Suzuki, M.; Tung, C. S.; Westhof, E.; Wolberger, C.; Berman, H. M. A standard reference frame for the description of nucleic acid base-pair geometry. *J. Mol. Biol.* **2001**, *313* (1), 229−237.

(28) Wong, V.; Case, D. A. Evaluating rotational diffusion from protein MD simulations. *J. Phys. Chem. B* **2008**, *112* (19), 6013−6024.

(29) Raymond, J. W.; Willett, P. Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *J. Comput.-Aided Mol. Des.* **2002**, *16* (7), 521−533.

(30) Brozell, S. R.; Mukherjee, S.; Balius, T. E.; Roe, D. R.; Case, D. A.; Rizzo, R. C. Evaluation of DOCK 6 as a pose generation and database enrichment tool. *J. Comput.-Aided Mol. Des.* **2012**, *26* (6), 749−773.

(31) Bacon, D. F.; Graham, S. L.; Sharp, O. J. Compiler transformations for high-performance computing. *ACM Comput. Surv.* **1994**, *26*, 345−420.

(32) Geist, A.; Gropp, W.; Huss-Lederman, S.; Lumsdaine, A.; Lusk, E.; Saphir, W.; Skjellum, T.; Snir, M. MPI-2: Extending the message-passing interface. In *Euro-Par'96 Parallel Processing Lecture Notes in Computer Science*; Springer: New York, 1996; pp 128−135.