

SPIDAL Summer REU 2020: Implementing an HDF5-based File Format Coordinate Reader into MDAnalysis

Edis Jakupovic

Department of Physics, College of Liberal Arts and Sciences
Arizona State University, Tempe, Arizona 85281

August 25, 2020

Abstract

Molecular dynamics (MD) trajectory files are now commonly Terabytes in size, which comes with the negative consequence that reading the file and extracting useful data has become a bottleneck in the analysis of MD trajectories. Results in recent works have shown that a message passing interface implementation which reads an HDF5 file in parallel is a promising approach to speeding up trajectory analysis, so the goal of this project was to implement a trajectory reader class, *H5MDReader*, into MDAnalysis that is capable of reading an H5MD file in parallel through the parallel I/O features of an MPI library. This was done using the h5py Python package, which can handle HDF5 file objects with MPI drivers for parallel reading and writing. The parallel features of *H5MDReader* have not been thoroughly tested, however the serial capabilities were benchmarked by calculating the RMSD of sample trajectories versus various file formats. The results show that, in serial, *H5MDReader* takes a substantially longer time to read files than other formats, which requires analysis into where *H5MDReader* is spending time during I/O.

1 Background

Molecular dynamics (MD) simulations provide a powerful and effective way to model the molecular happenings of a biological system. From the folding and unfolding of a protein molecule, to understanding the interaction between proteins and drug molecules, and even problems in materials science, MD simulations are a vital tool to understanding the function of biomolecules. These simulations are constructed by calculating the forces on each particle that are derived from a classical potential energy function and Newton’s equations of motion, and are stored as “trajectory” files, which contain the position (but also can contain velocity, force, etc.) of every atom in the system for every frame of time in the length of the simulation³. As computing power continues to increase, and simulation data becomes more robust, the size of these trajectory files is now commonly Terabytes in size, which comes with the negative consequence that reading the file (input) and extracting useful data (output) has become a bottleneck in the analysis of MD trajectories². Previous works^{2,3} have shown that performing analysis tasks on trajectory data in parallel, rather than in series, can greatly boost the performance of certain types of analysis.

One of the issues in the field of computational biophysics is that trajectory files can be written in many different types of file formats, to the point where poor documentation and the barrier of learning how to access the data of one specific file format can be a hindrance to scientific productivity. MDAnalysis is a Python package that fixes this issue by handling the reading of over 25 different file formats internally, and providing an intuitive interface for accessing the useful data in a trajectory file¹. Currently, trajectories can be analyzed in parallel using Parallel MDAnalysis (PMDA), which uses the Dask library that follows a split-apply-combine approach for trajectory analysis³. A message passing interface (MPI) implementation reading an HDF5 file in parallel is also a promising approach to speeding up trajectory analysis². However, MDAnalysis did not have an HDF5-based format trajectory reader, so the goal of this project was to implement an HDF5-based trajectory reader into MDAnalysis that is capable of reading a file in parallel through the parallel I/O features of an MPI library.

H5MD, or “HDF5 for molecular data”, is a file format that is used to store data from MD simulations such as particle coordinates, simulation box dimensions, and thermodynamic observables⁴. It is built upon the HDF5 file format, which is a structured, binary file format that organizes data into 2 objects: groups and datasets⁴. HDF5 files follow a hierarchical, tree-like structure, where groups represent nodes, or “splitting” points in the branches of the tree, and datasets represent the end of a branch, or the “leaves” of the tree. Both groups and datasets can contain metadata in the form of attributes, which can be especially useful in MD simulations with something like a “unit” attribute that is attached to a “positions” dataset. H5MD provides specifications with how these groups and datasets are organized and named for MD simulation data which is important for consistency in future scientific works that use HDF5-based file formats for MD simulations.

A Python package, pyh5md, already exists that can read and write H5MD files using the h5py libraries to handle HDF5 file objects with Python. Pyh5md was originally going to be used to handle the HDF5 file objects in MDAnalysis because it already adheres to H5MD’s specifications, however it is no longer being maintained by its developers. For this reason, we decided to use h5py directly to implement an *H5MDReader* class that obeys the H5MD format specifications that will add to the array of MDAnalysis coordinate readers. Some simple benchmarks were done to compare the serial performance of an root-mean-square deviation

(RMSD) calculation of the alpha carbon positions of various file format readers in MDAnalysis.

2 Methods

2.1 Implementation

We added a module, *H5MD.py*, to MDAnalysis.coordinates that includes *class Timestep* and *class H5MDReader*. The Timestep class derives from *base.Timestep*, and overrides the method *_init_unitcell()* because H5MD stores unitcell information as a matrix, with the edge vectors of the triclinic box being the rows of the matrix. The *dimensions()* property and setter are then overridden, where the *core.triclinic_box()* method is used to convert the box vectors given by the H5MD file into [lx, ly, lz, alpha, beta, gamma] format for MDAnalysis to display.

H5MDReader derives from *base.ReaderBase*, which derives from *base.ProtoReader*. Upon initialization, *H5MDReader* checks for the arguments *driver* and *comm* that are passed to *h5py.File* to open the HDF5 with parallel MPI I/O capabilities, however this implementation is not thoroughly tested yet. The method *open_trajectory()* calls the method *h5py.File* to open the HDF5 file object, and provide a file handle as *self._file*. If units are provided in the H5MD file, *_translate_h5md_units()* will convert the unit string from H5MD notation to MDAnalysis standard unit notation.

Two crucial methods were overridden that *H5MDReader* uses to construct the Timestep object when a specific frame is called with MDAnalysis: *_read_frame()* and *_read_next_timestep()*, where *_read_next_timestep()* simply calls *_read_frame()* shifted by +1 frame. The method *_read_frame()* extracts the data from the H5MD file with the helper functions, *_copy_data()*, *_get_frame_dataset()*, and *_convert_units()*. *_copy_data()* scans the “observables” group in the H5MD files and stores any data into the *ts.data* dictionary of an MDAnalysis universe. *_get_frame_dataset()* checks to see if the groups position, velocity, or force exist in the H5MD file and store the array for that specific frame into *ts.positions*, *ts.velocities*, and *ts.forces*, respectively. *_convert_units()* converts the values in the position, velocity, and force array to MDAnalysis standard units.

2.2 Benchmarking

In order to test the serial capabilities of *H5MDReader*, we ran some simple benchmarks that used the *time.time()* method to calculate the time it took for MDAnalysis to perform an RMSD calculation on the alpha carbon positions of the atoms of a sample trajectory in the MDAnalysis testsuite datafiles, and compared these results with *pyh5md* computing the same thing on the same array of positions.

To set up the files for benchmarking, we used the *cobrotoxin.tpr* topology file from MDAnalysisTests.datafiles, and used 4 identical trajectory files: *cobrotoxin.dcd*, *cobrotoxin.xtc*, *cobrotoxin.trr*, and *cobrotoxin.h5md*. It should be noted that *cobrotoxin.xtc* and *cobrotoxin.dcd* do not contain velocity and force data. The trajectories were extended by 1000x their original length by writing new files with the MDAnalysis *ChainReader* class to have a size of 3000 frames of 19385 atoms. A Python script was written where the file was opened, and iterated over each time frame to calculate the RMSD. Three time values were stored: the total I/O time for MDAnalysis to open the timestep, and the total time for the RMSD calculation to complete, and the

total time for the loop to complete which included both the I/O time and RMSD calculation time. These total times were divided by the number of frames in the trajectory to give the time/frame for loop, I/O, and RMSD calculation. The benchmark was run 5 times, and the mean and standard deviation of the 5 runs was used to plot bar graphs illustrating how long each section of the benchmark took. The same benchmark was done using pyh5md to open *cobrotoxin.h5md* instead of MDAnalysis and added to the bar graphs.

3 Results and Discussion

The goal of this project was to implement an HDF5-based format trajectory reader into MDAnalysis that is capable of reading a file in parallel through the parallel I/O features of an MPI library. While the parallel features of the implemented *H5MDReader* have not been thoroughly tested, the serial capabilities of the reader have been. A suite of tests were written in MDAnalysisTests.coordinates.test_h5md.py that verified the validity of the data read into the reader, as well as making sure all of the appropriate errors are raised when needed, such as the H5MD file not containing units, yet *convert_units* is set to *True* upon initialization. *H5MDReader* unpacks an H5MD file’s “particles” group and stores the box dimensions, positions, velocities, and forces into appropriate MDAnalysis Timestep attributes, along with any units associated with each dataset. It does not currently read charge or mass data from the “particles” group, however this is typically handled in the topology file in MDAnalysis. It also has the capability to store any datasets found in the “observables” group into the Timestep.data dictionary. As far as the remaining “connectivity” and “parameters” groups defined in the H5MD specifications, the implemented *H5MDReader* does not currently read any data from these groups, however it can easily be implemented given the accessibility allowed by h5py.

The benchmarks were performed on a remote workstation on a solid state drive, and the files were prepared for benchmarking as given in the Methods section. Interestingly, the H5MD file with 3 frames was much larger than the TRR file which contained identical data. Table 1 gives the size of each file in its initial trajectory size, and with 1000x the frames. When the trajectories are expanded to a larger number of frames, however, this difference disappears, which seems to indicate the HDF5 data structures take up more space than the TRR file, but as the size of the arrays increases they begin to dominate the file size. Figure 1

Size	DCD	TRR	XTC	H5MD
1x	683 KB	2 MB	194 KB	22 MB
1000x	666 MB	2 GB	189 MB	2 GB

Table 1: File sizes for different trajectory formats for 1x frames and 1000x frames

below gives the bar graphs for the benchmark loop total time and time per frame for each file format using MDAnalysis, as well as the same benchmark using the pyh5md package. From the figure, it seems that *H5MDReader* takes a substantially longer time to perform the benchmark than the other file formats. DCD and XTC do not contain velocity and force data, so it makes sense that their benchmarks would be faster, however the TRR file contains the same trajectory data, yet takes less than half the time has the H5MD file.

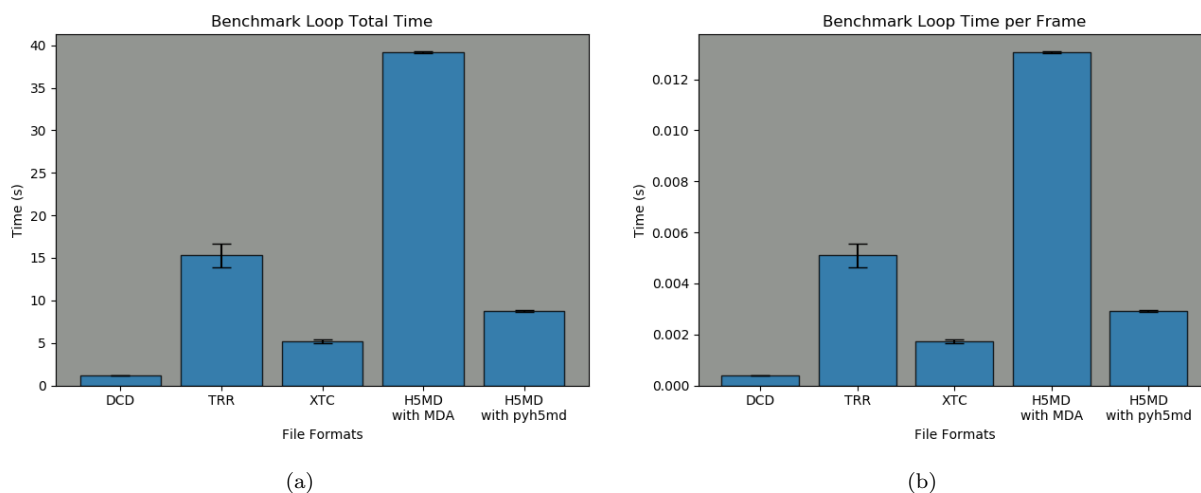


Figure 1: Benchmark loop times with standard deviation error bars. (a) gives the total time spent during the loop and (b) gives the time spent per frame.

To see at which point during the benchmark this discrepancy comes from, we can look at the I/O time and the RMSD calculation time of the benchmark, given in figure 2 below. From sub-figures (c) and (d) in Figure 2, we see that all of the file formats are relatively equal in their RMSD computation time, however H5MD with MDAnalysis is still slightly slower than the rest. Sub-figures (a) and (b), however, show the same large difference between the TRR and H5MD benchmark times. These results indicate that the I/O portion of MDAnalysis opening and reading the timestep is responsible for large discrepancy between the H5MD benchmark time and the other file formats. This is a peculiar result and requires a deeper look into what *H5MDReader* is spending the most time on during the benchmark.

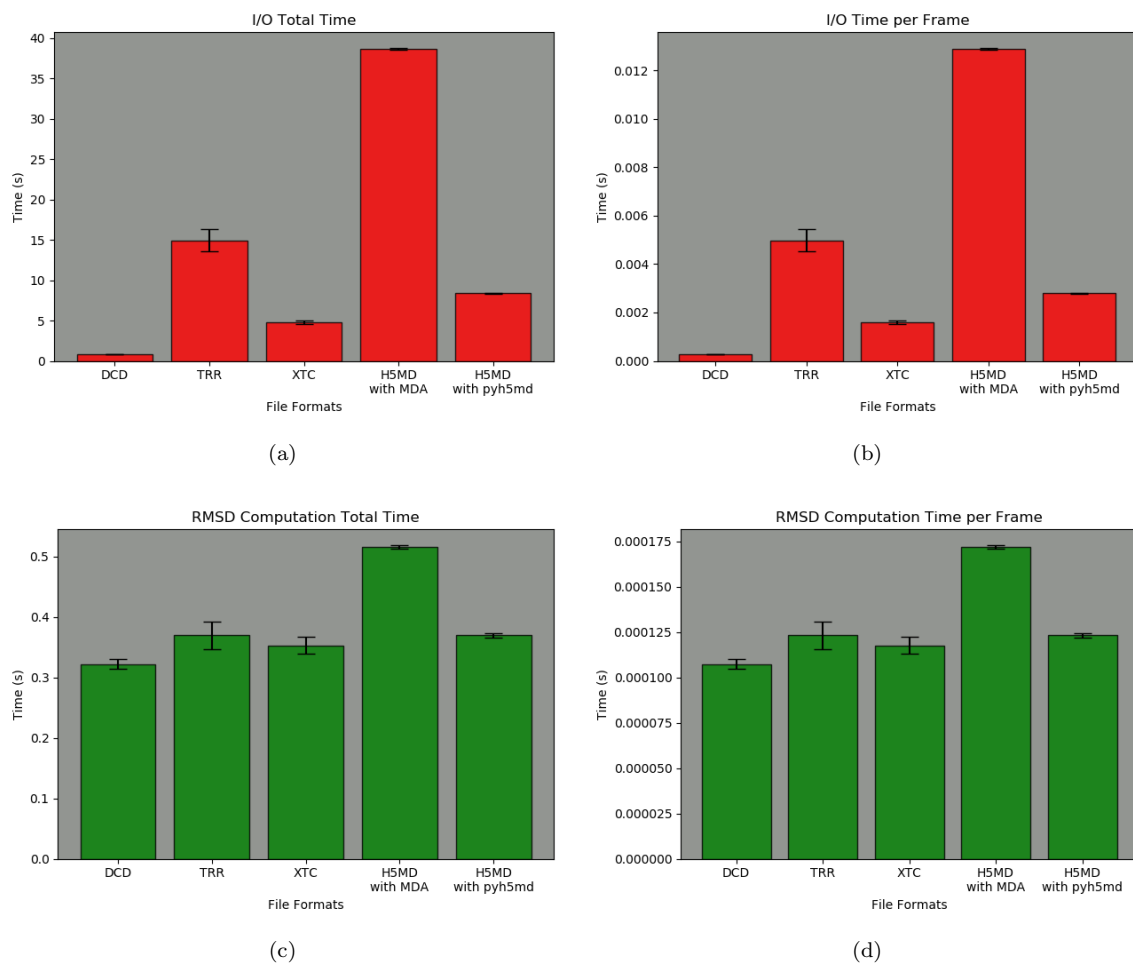


Figure 2: I/O and RMSD calculation times with standard deviation error bars. (a) and (c) give the total time spent during the benchmark. (b) and (d) give the time spent per frame.

4 Conclusion

The aim of this project was to implement an HDF5-based file format coordinate reader into the MDAnalysis package. The implemented *H5MDReader* class successfully reads trajectory data from an H5MD file, however our benchmarks show I/O speed of handling an H5MD file with MDAnalysis is much slower than a TRR file with the same data. More work is required to see where *H5MDReader* is spending its time while opening and reading a file. Further testing is also required on the parallel MPI I/O capabilities of *H5MDReader*.

5 Acknowledgements

Funding was provided by the National Science Foundation for a REU supplement to award ACI1443054.

References

- [1] R. J. Gowers, M. Linke, J. Barnoud, T. J. E. Reddy, M. N. Melo, S. L. Seyler, D. L. Dotson, J. Domanski, S. Buchoux, I. M. Kenney, and O. Beckstein. MDAnalysis: A Python package for the rapid analysis of molecular dynamics simulations. In S. Benthall and S. Rostrup, editors, Proceedings of the 15th Python in Science Conference, pages 102-109, Austin, TX, 2016. SciPy
- [2] Khoshlessan, Mahzad & Paraskevagos, Ioannis & Fox, Geoffrey & Jha, Shantenu & Beckstein, Oliver. (2019). Parallel Performance of Molecular Dynamics Trajectory Analysis.
- [3] Shujie Fan, Max Linke, Ioannis Paraskevagos, Richard Gowers, Michael Gecht, and Oliver Beckstein. “PMDA - Parallel Molecular Dynamics Analysis”. In: Python in Science Conference. Austin, Texas, 2019, pp. 134–142.
- [4] Pierre Buyl, Peter Colberg, Felix Höfling (2013). H5MD: A structured, efficient, and portable file format for molecular data. Computer Physics Communications. 185. 10.1016/j.cpc.2014.01.018.