MPI-parallel Molecular Dynamics Trajectory Analysis with the H5MD Format in the MDAnalysis Python Package

Edis Jakupovic[‡], Oliver Beckstein[‡*]

Abstract—Fill here

Index Terms—Molecular Dynamics Simulations, High Performance Computing, Python, MDAnalysis, HDF5, H5MD, MPI I/O

Introduction

As HPC resources continue to increase, the size of molecular dynamics (MD) simulation files are now commonly terabytes in size, making serial analysis of these trajectory files impractical. Parallel analysis is a necessity for the efficient use of both HPC resources and a scientist's time. MDAnalysis is a widely used Python library that can read and write over 20 popular MD file formats while providing the same user-friendly interface [GLB+16]. Previous work that focused on developing a task-based approach to parallel analysis found that an IO bound task only scaled to 12 cores due to a file IO bottleneck [SFMLIP+19]. Our previous feasibility study suggested that parallel reading via MPI-IO and HDF5 can lead to good scaling although it only used a reduced size custom HDF5 trajectory and did not provide a usable implementation of a true MD trajectory reader [KPF+20].

H5MD, or "HDF5 for molecular data", is an HDF5-based file format that is used to store MD simulation data, such as particle coordinates, box dimensions, and thermodynamic observables [dBCH14]. HDF5 is a structured, binary file format that organizes data into 2 objects: groups and datasets, which follows a hierarchical, tree-like structure, where groups represent nodes of the tree, and datasets represent the leaves [Col14]. The HDF5 library can be built on top of a message passing interface (MPI) implementation so that a file can be accessed in parallel on a parallel filesystem such as Lustre or BeeGFS. We implemented a parallel MPI-IO capable HDF5-based file format trajectory reader into MDAnalysis, H5MDReader, that adheres to H5MD specifications. H5MDReader interfaces with h5py, a high level Python package that provides a Pythonic interface to the HDF5 format such that accessing a file in parallel is as easy as passing a keyword argument into h5py.File, and all of parallel disk access occurs under the hood.

We benchmarked H5MDReader's parallel reading capabilities with MDAnalysis on three HPC clusters: ASU Agave, SDSC Comet, and PSC Bridges. The benchmark consisted of a simple split-apply-combine scheme of an IO-bound task that split a 90k frame (113GB) trajectory into n chunks for n processes, where each process a task on their chunk of data, and then gathered the results back to the root process. For the computational task, we computed the time series root mean squared distance (RMSD) of the positions of the alpha carbons in the protein to their initial coordinates at the first frame of the trajectory. The RMSD calculation is not only a very common task performed to analyze the dynamics of the structure of a protein, but more importantly is a very fast computation that is heavily bounded by how quickly data can be read from the file. Therefore it provided an excellent analysis candidate to test the I/O capabilities of H5MDReader.

Across the three HPC clusters tested, the benchmarks were done on both a BeeGFS and Lustre parallel filesystem which is highly suited for multi-node MPI parallelization. We tested various algorithmic optimizations for our benchmark, including altering the stripe count, loading only necessary coordinate information with numpy.Masked_arrays, and front loading all IO by loading the entire trajectory into memory prior to the RMSD calculation.

BRIEFLY DISCUSS RESULTS AND CHUNKING

Methods

In order to obtain detailed timing information we instrumented code as follows:

```
class timeit(object):
    def __enter__(self):
        self._start_time = time.time()
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        end_time = time.time()
        self.elapsed = end_time - self._start_time
        # always propagate exceptions forward
        return False
```

```
import MDAnalysis as mda
from MDAnalysis.analysis.rms import rmsd
from mpi4py import MPI
import numpy as np

def benchmark(topology, trajectory):
```

‡ Arizona State University
* Corresponding author: obeckste@asu.edu

```python
    with timeit() as init_top:
        u = mda.Universe(topology)
    with timeit() as init_traj:
        u.load_new(trajectory, driver="mpio", comm=MPI.COMM_WORLD)
    t_init_top = init_top.elapsed
    t_init_traj = init_traj.elapsed
    CA = u.select_atoms("protein and name CA")
    x_ref = CA.positions.copy()

    total_io = 0
    total_rmsd = 0
    rmsd_array = np.empty(bsize, dtype=float)
    for i, frame in enumerate(range(start, stop)):
        with timeit() as io:
            ts = u.trajectory[frame]
        total_io += io.elapsed
        with timeit() as rms:
            rmsd_array[i] = rmsd(CA.positions, x_ref, superposition=True)
        total_rmsd += rms.elapsed

    with timeit() as wait_time:
        comm.Barrier()
    t_wait = wait_time.elapsed

    with timeit() as comm_gather:
        rmsd_buffer = None
        if rank == 0:
            rmsd_buffer = np.empty(n_frames, dtype=float)
        comm.Gatherv(sendbuf=rmsd_array, recvbuf=(rmsd_buffer, sendcounts), root=0)
    t_comm_gather = comm_gather.elapsed
```

Results and Discussion

TODO

Conclusions

TODO

Acknowledgments

Funding was provided by the National Science Foundation for a REU supplement to award ACI1443054. The SDSC Comet computer at the San Diego Supercomputer Center was used under allocation TG-MCB130177. The authors acknowledge Research Computing at Arizona State University for providing HPC resources that have contributed to the research results reported within this paper. We would like to acknowledge Gil Speyer and Jason Yalim from the Research Computing Core Facilities at Arizona State University for advice and consultation.

References

[Col14] Andrew Collette. Python and hdf5. In Meghan Blanchette and Rachel Roumeliotis, editors, *Python and HDF5*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472., 2014.

[dBCH14] Pierre de Buyl, Peter H. Colberg, and Felix Höfling. H5MD: A structured, efficient, and portable file format for molecular data. *Computer Physics Communications*, 185(6):1546 – 1553, 2014. doi:10.1016/j.cpc.2014.01.018.

[GLB+16] Richard J. Gowers, Max Linke, Jonathan Barnoud, Tyler J. E. Reddy, Manuel N. Melo, Sean L. Seyler, David L Dotson, Jan Domański, Sébastien Buchoux, Ian M. Kenney, and Oliver Beckstein. MDAnalysis: A Python package for the rapid analysis of molecular dynamics simulations. In Sebastian Benthall and Scott Rostrup, editors, *Proceedings of the 15th Python in Science Conference*, pages 98–105, Austin, TX, 2016. SciPy. URL: https://www.mdanalysis.org, doi:10.25080/Majora-629e541a-00e.

[KPF+20] Mahzad Khoshlessan, Ioannis Paraskevakos, Geoffrey C. Fox, Shantenu Jha, and Oliver Beckstein. Parallel performance of molecular dynamics trajectory analysis. *Concurrency and Computation: Practice and Experience*, 32:e5789, 2020. doi:10.1002/cpe.5789.

[SFMLIP+19] Shujie Fan, Max Linke, Ioannis Paraskevakos, Richard J. Gowers, Michael Gecht, and Oliver Beckstein. PMDA - Parallel Molecular Dynamics Analysis. In Chris Calloway, David Lippa, Dillon Niederhut, and David Shupe, editors, *Proceedings of the 18th Python in Science Conference*, pages 134 – 142, Austin, TX, 2019. SciPy. doi:10.25080/Majora-7ddc1dd1-013.