Speeding Up Molecular Dynamics Trajectory Analysis with MPI Parallelization

Edis Jakupovic[1], Oliver Beckstein[1]
[1]Department of Physics, Arizona State University

Center for Biological Physics
Arizona State University
Beckstein Lab
Computational Biophysics at Arizona State University

Introduction

Molecular dynamics (MD) computer simulation files are now commonly terabytes in size, which means analyzing the data from these files becomes a painstakingly expensive task. Parallel analysis is becoming a necessity for the efficient use of time and high-performance computing (HPC) resources. One promising way around this file I/O hurdle is to use a parallel message passing interface (MPI) implementation with the HDF5 (Hierarchical Data Format 5) file format to access a single file simultaneously with numerous processes on a parallel file system.

In this study, we benchmarked the parallel I/O performance of the H5MD[1] reader we implemented into MDAnalysis[2]. The benchmark task consisted of parallelizing the root mean square distance (RMSD) calculation after optimal structural superposition. We tested the benchmark with several algorithmic optimizations, including manually adjusting the chunk shape of the file as opposed to letting h5py (the Python library that wraps the HDF5 library) automatically chunk the file.

Methods

Benchmark Code

The MPI parallel I/O benchmark is given by the sample code below. The specific lines that are timed in benchmark are labeled by color. The benchmark was run on 3 HPC systems: ASU Agave, PSC Bridges, SDSC Comet, on up to 4 full nodes.

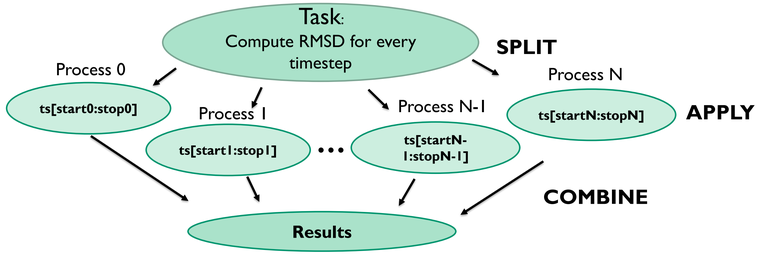
Color Legend

t^{init_top}
t^{init_traj}
$t^{I/O}$
$t^{compute}$
t^{wait}
t^{comm_gather}

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

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()

def benchmark(topology, trajectory):
    u = mda.Universe(topology)
    u.load_new(trajectory, driver="mpio", comm=comm)
    CA = u.select_atoms("protein and name CA")
    x_ref = CA.positions.copy()
    slices = make_balanced_slices(n_frames, size, start=0, stop=n_frames, step=1)
    start = slices[rank].start
    stop = slices[rank].stop
    bsize = stop - start
    sendcounts = np.array([slices[i].stop - slices[i].start for i in range(size)])

    rmsd_array = np.empty(bsize, dtype=float)
    for i, frame in enumerate(range(start, stop)):
        ts = u.trajectory[frame]
        rmsd_array[i] = rmsd(CA.positions, x_ref, superposition=True)
    comm.Barrier()
    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)
```

Parallelization Scheme

We used a split-apply-combine scheme in which the trajectory is split into *N* blocks for *N* processes. Each process receives a unique start and stop index for their time step (ts) iteration.



Results

Baseline Results

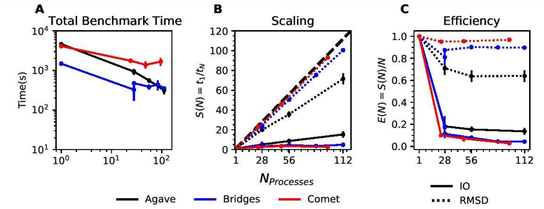
Baseline results for the default, auto-chunked file with no optimizations applied. Maximum I/O scaling reached 15x at 112 cores on Agave (**B**).

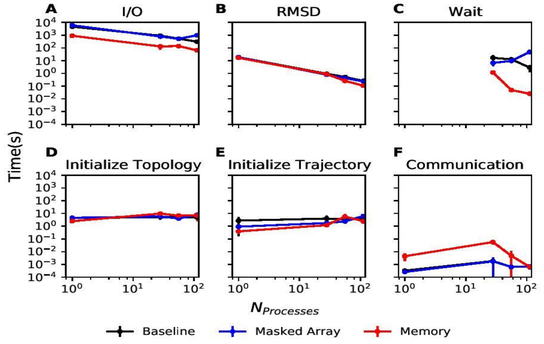
Optimization Results

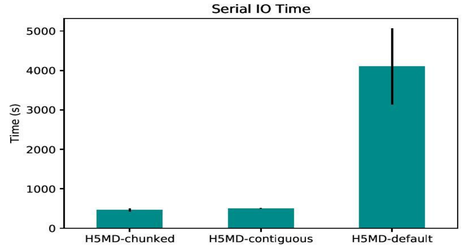
Loading the entire trajectory into memory prior to computation showed the largest improvement in I/O time (**A**). This suggests the auto-chunked file layout on disk was not suited for the access pattern of the benchmark.

Impact of HDF5 Chunking

A ~10x serial speedup was observed by using custom chunk shapes. H5MD-contiguous was saved with no chunking applied.

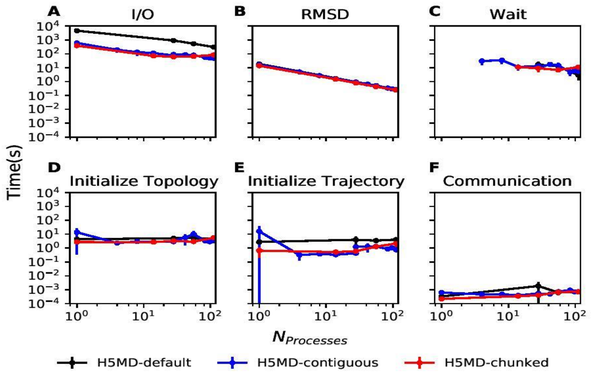




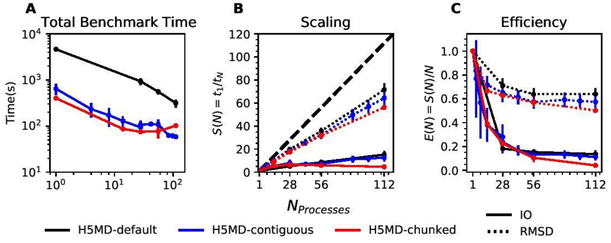


HDF5 Chunking Parallel Results

The H5MD-default serial run was 4623s while the 112-core H5MD-contiguous run was 47s (**A**) (98x speedup with respect to baseline).



Similar speedups were observed for all 3 files up to 2 nodes, with the default and contiguous files reaching ~15x at 4 nodes (**B**).



Conclusions

Our implementation of an HDF5-based file format trajectory reader in MDAnalysis can perform parallel MPI I/O, and our benchmarks on various national HPC environments show that speed-ups on the order of 20x for 48 cores are attainable. Scaling up to achieve higher parallel data ingestion rates remains challenging, so we developed several algorithmic optimizations in our analysis workflows that lead to improvements in I/O times. The results from these optimization attempts led us to find that the original data file that was auto-chunked by h5py's chunking algorithm had an incredibly inefficient chunk layout of the original file.

Our properly chunked file led to I/O time speedups of 98x (with respect to the serial I/O time of the auto-chunked file) at 112 cores on Agave, which means carefully thinking not only about how your file is accessed, but also how the file is stored on disk can result in a reduction of analysis time from 4623 to 47 seconds. The addition of the HDF5 reader provides a foundation for the development of parallel trajectory analysis with MPI and the MDAnalysis package.

Acknowledgements

The authors thank Dr. Pierre de Buyl for advice on the implementation of the h5md format reading code and acknowledge Gil Speyer and Jason Yalim from the Research Computing Core Facilities at Arizona State University for support with the Agave cluster and BeeGFS. This work was supported by the National Science Foundation through a REU supplement to award ACI1443054 and used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1548562. The SDSC Comet computer at the San Diego Supercomputer Center and PSC Bridges computer at the Pittsburgh Supercomputing Center were used under allocation TG-MCB130177. The authors acknowledge Research Computing at Arizona State University for providing HPC and storage resources that contributed to the research results reported within this paper.

References

[1] 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.
[2] Richard J. Gowers, Max Linke, Jonathan Barnoud, Tyler J. E. Reddy, Manuel N. Melo, Sean L. Seyler, David L Dotson, Jan Domanski, 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. doi:10.25080/Majora-629e541a-00e.