PMDA-Parallel Molecular Dynamics Analysis

Shujie Fan[1†], Max Linke[2†], Ioannis Paraskevakos[3], Richard J. Gowers[4], Michael Gecht[2], Oliver Beckstein[1]

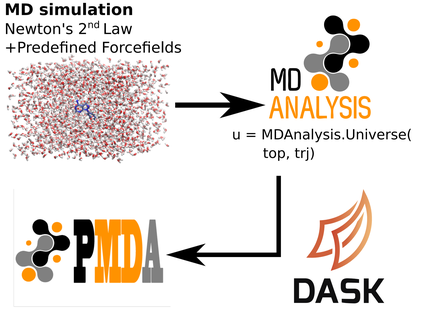
[1]Department of Physics, Arizona State University, Tempe, AZ, USA [2] Department of Theoretical Biophysics, Max Planck Institute of Biophysics, Frankfurt am Main, Germany
[3] Rutgers University, Piscataway, NJ, USA [4] Department of Chemistry, University of New Hampshire, Durham, NH, USA

E-mail: sfan19@asu.edu max.linke88@gmail.com i.paraskev@rutgers.edu richardjgowers@gmail.com michael.gecht@biophys.mpg.de obeckste@asu.edu

https://www.mdanalysis.org/pmda/

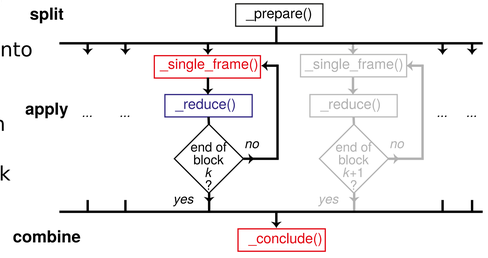
Introduction

PMDA is a Python library that builds upon MDAnalysis[1] and Dask[2] to provide parallel analysis algorithms for molecule dynamics (MD) simulations. At the core of PMDA is the idea that a common interface makes it easy to create code that can be easily parallelized.



Methods

split-apply-combine approach[3]:
The trajectory is split into blocks, analysis is performed separately and in parallel on each block ("apply"), then results from each block are gathered and combined.



Acknowledgments

We would like to thank reviewer Cyrus Harrison for the idea to plot the fractional time spent on different stages of the program. This work was supported by the National Science Foundation under grant numbers ACI-1443054 and used the Extreme Science and Engineering Discovery Environment (XSEDE) supported by National Science Foundation grant number ACI-1548562. The SDSC Comet computer at the San Diego Supercomputer Center was used under allocation TG-MCB130177. Max Linke was supported by NumFOCUS under a small development grant.

References

[1] Gowers, Richard J.; Linke, Max; Barnoud, Jonathan; Reddy, Tyler J. E.; Melo, Manuel N.; Seyler, Sean L.; Dotson, David L.; Domański, Jan; Buchoux, Sébastien; Kenney, Ian M.; and Beckstein, Oliver. 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. URL: https://www.mdanalysis.org/.
[2] Dask Development Team. Dask: Library for dynamic task scheduling, 2016, URL https://dask.org.
[3] Hadley Wickham. The split-apply-combine strategy for data analysis. Journal of Statistical Software, 40(1), 2011. doi:10.18637/jss.v040.i01.
[4] Alan H. Karp and Horace P. Flatt. Measuring parallel processor performance. Commun. ACM 33, 539-543, 1990. doi: https://doi.org/10.1145/78607.78614.

Using PMDA

PMDA is released under the GNU General Public License, version 2
Source code is available in the public GitHub repository https://github.com/MDAnalysis/pmda/.

Installation

Install with conda:
```
conda config --add channels conda-forge
conda install pmda
```

Install with pip:
```
pip install --upgrade pmda
```

Install from source:
```
git clone git@github.com:MDAnalysis/pmda.git
cd pmda
python setup.py install
```

User-defined Analysis

pmda.custom.AnalysisFromFunction():

```
import MDAnalysis as mda

u = mda.Universe(top, traj)
protein = u.select_atoms('protein')

def rgyr(ag):
    return (ag.universe.trajectory.time,
            ag.radius_of_gyration())

import pmda.custom
parallel_rgyr =
    pmda.custom.AnalysisFromFunction(
        rgyr, u, protein)
parallel_rgyr.run(n_jobs=4, n_blocks=4)
print(parallel_rgyr.results)
```

Pre-defined Analysis

```
import MDAnalysis as mda
from pmda import rms

u = mda.Universe(top, trj)
ca = u.select_atoms('name CA')
u.trajectory[0]
ref = u.select_atoms('name CA')
rmsd = rms.RMSD(ca, ref)
rmsd.run(n_jobs=4, n_blocks=4)
print(rmsd.rmsd)
```

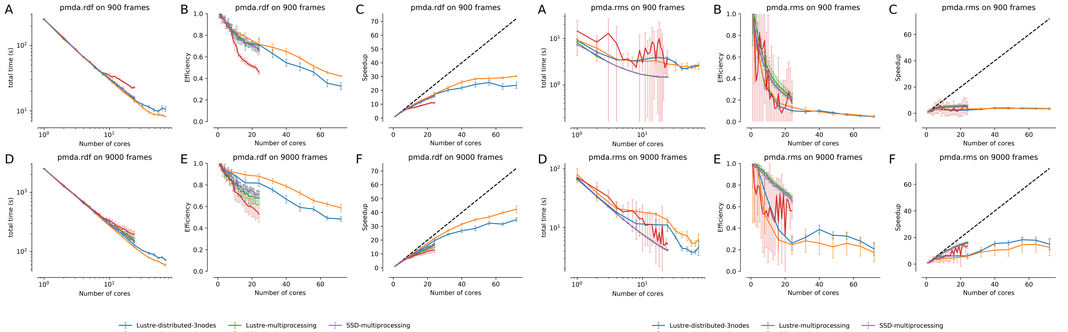
pmda.parallel.ParallelAnalysisBase:

```
import numpy as np
from pmda.parallel import ParallelAnalysisBase
class RGYR(ParallelAnalysisBase):
    def __init__(self, protein):
        universe = protein.universe
        super(RGYR, self).__init__(
            universe, (protein,))
    def _prepare(self):
        self.rgyr = None
    def _conclude(self):
        self.rgyr = np.vstack(self._results)
    def _single_frame(self, ts, atomgroups):
        protein = atomgroups[0]
        return (
            ts.time, protein.radius_of_gyration())
parallel_rgyr = RGYR(protein)
parallel_rgyr.run(n_jobs=4, n_blocks=4)
print(parallel_rgyr.results)
```

Performance Evaluation

We tested two tasks(RDF and RMSD) with different combinations of Dask schedulers with different means to read the trajectory data as shown in the Table.

configuration label	file storage	scheduler	max nodes	max processes
Lustre-distributed-3nodes	Lustre	*distributed*	3	72
Lustre-distributed-6nodes	Lustre	*distributed*	6	72
Lustre-multiprocessing	Lustre	*multiprocessing*	1	24
SSD-distributed	SSD	*distributed*	1	24
SSD-multiprocessing	SSD	*multiprocessing*	1	24



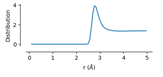
Speed-up: $S(M) = \frac{t^{total}(1)}{t^{total}(M)}$

Efficiency: $E(M) = \frac{S(M)}{M}$

Serial fraction[4]: $f(M) = \frac{1/S(M) - 1/M}{1 - 1/M}$

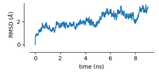
RDF

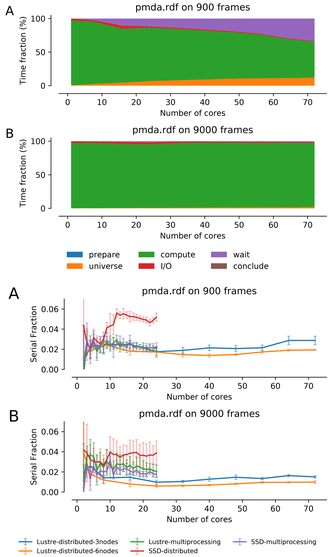
Water oxygen-oxygen radial distribution function for all 24,239 oxygen atoms in the water molecules.



RMSD

Time series of root mean square distance after optimum superposition (RMSD) of all 564 Cα atoms of a protein.





Conclusion

The PMDA Python package provides a framework to parallelize analysis of MD trajectories with a simple split-apply-combine approach by combining Dask with MDAnalysis.
We showed that performance depends on the type of analysis that is being performed. Compute-intensive tasks such as the RDF calculation can show good strong scaling with up to about a hundred cores on a typical supercomputer and a sizable speed-up that approached 40. Such performance should make this an attractive solution for many users. For tasks such as the RMSD calculation, whose speed-up is limited by a considerable serial fraction, a single multi-core workstation seems sufficient to achieve speed-ups on the order of 10 and HPC resources would not help improve the performance.