Vol. 29 no. 18 2013, pages 2363–2364
doi:10.1093/bioinformatics/btt402

Advance Access publication July 10, 2013

pyRMSD: a Python package for efficient pairwise RMSD matrix calculation and handling

Víctor A. Gil[1] and Víctor Guallar[1,2,*]

[1]Joint BSC-IRB Research Program in Computational Biology, Barcelona Supercomputing Center, 08034 Barcelona, Spain and [2]Institució Catalana de Recerca i Estudis Avançats (ICREA), Passeig Lluís Companys 23, E-08010 Barcelona, Spain

Associate Editor: Anna Tramontano

ABSTRACT

Summary: We introduce pyRMSD, an open source standalone Python package that aims at offering an integrative and efficient way of performing Root Mean Square Deviation (RMSD)-related calculations of large sets of structures. It is specially tuned to do fast collective RMSD calculations, as pairwise RMSD matrices, implementing up to three well-known superposition algorithms. pyRMSD provides its own symmetric distance matrix class that, besides the fact that it can be used as a regular matrix, helps to save memory and increases memory access speed. This last feature can dramatically improve the overall performance of any Python algorithm using it. In addition, its extensibility, testing suites and documentation make it a good choice to those in need of a workbench for developing or testing new algorithms.

Availability: The source code (under MIT license), installer, test suites and benchmarks can be found at https://pele.bsc.es/ under the tools section.

Contact: victor.guallar@bsc.es

Supplementary information: Supplementary data are available at *Bioinformatics* online.

Received on April 24, 2013; revised on June 11, 2013; accepted on July 5, 2013

1 INTRODUCTION

As molecular modeling keeps expanding, obtaining the Root Mean Square Deviation (RMSD) with optimum superposition for a large set of structures in an efficient and fast manner is a necessity. Clustering methods, for example, which are becoming increasingly popular as trajectory analysis and compression tools (Karpen *et al*., 1993; Phillips *et al*., 2011), can benefit from the use of a pre-calculated pairwise distance matrix or even totally depend on it, e.g. Spectral Clustering (Luxburg, 2007). However, as hardware and algorithms improve, the output size of simulations grows bigger, and the calculation of the distance matrix becomes the bottleneck in any process depending on it. There are several implementations of the different superposition algorithms, which are written in wide spectra of programming languages. Almost all Molecular Dynamics packages and biomolecule handling software include their own RMSD calculation tools. Every time programmers need to use an external RMSD solution in a project, they have two options. The first one is to use an external source or library, which requires previous knowledge of the language in which it was written and its dependencies. A second option is to use a precompiled tool with a bigger scope, which means creating an interface with their own application by writing wrappers and output converters (with the consequent performance loss). In general, the main problems to face are fragmentation, excess of or missing features, bad documentation, lack of sources and the intrinsic difficulty of the languages used. pyRMSD is a Python package that overcomes all the above problems in the following way:

- It is totally focused on the calculation of RMSD. It provides solutions for all the usual RMSD problems and is specially tuned for RMSD collective calculations, like pairwise RMSD matrices, a feature that is usually missing in most utilities.
- Python (www.python.org) is an easy to learn and use programming language, which has an extensive library pool that includes wrappers for almost all libraries used in science. This makes it one of the better languages for scientific software prototyping and development.
- As pure Python implementations have a poor performance (even when using fine tuned packages as numpy), pyRMSD uses Python C extensions with OpenMP and CUDA code, allowing the full use of multicore machines and Graphics Processing Units (GPU).
- It implements the most important superposition algorithms in the same place.
- It is documented, well tested and open source; therefore, it can be the perfect workbench for any experienced user who wants to develop and test their own superposition algorithms.

2 IMPLEMENTATION

2.1 Features

pyRMSD is built around two main classes: the *RMSDCalculator* and the *CondensedMatrix*. The *RMSDCalculator* class provides a straightforward interface to three superposition algorithm implementations, as well as some convenience methods to set up their options: Kabsch's superposition algorithm (Kabsch, 1978), QTRFIT (Heisterberg, 1990) and the Quaternion Characteristic Polinomial method (QCP) (Theobald, 2005). All have been written as Python C extensions, with serial and parallel (OpenMP) versions for the first two and an additional CUDA version for the last.

*To whom correspondence should be addressed.

Downloaded from http://bioinformatics.oxfordjournals.org/ by guest on February 7, 2016

© The Author 2013. Published by Oxford University Press. All rights reserved. For Permissions, please e-mail: journals.permissions@oup.com

RMSDCalculator's methods cover all the usual scenarios for superposition and RMSD calculation:

(1) Pairwise RMSD calculation.
(2) Reference versus the rest of the set.
(3) Reference versus following conformations.
(4) Calculation of a pairwise RMSD matrix of the whole set.
(5) Iterative superposition of a set of conformations.

Moreover, it offers two additional options that further extend the previous methods. The first one is allowing the modification of input coordinates to obtain the superposed conformations. The second one is to the use of different coordinate sets for superposition and RMSD calculation.

The *CondensedMatrix* class models a symmetric squared matrix. It allows the same row/column access of a regular matrix, storing only the upper triangle and thus saving half of the memory. The class has been completely written in C, allowing access times which are up to 6× faster compared with its Python counterpart. As a consequence, any algorithm that requires intensive matrix read access improves its performance. For instance, our cardinality function benchmark, available in the benchmarks folder, shows a 100× free speedup just by using it.

pyRMSD also provides two small helper classes that make the process of generating a pairwise RMSD matrix easier. The *Reader* class obtains the coordinate sets by means of a simple and fast C written PDB reader. Finally, the *MatrixHandler* class is capable of creating a distance matrix from a set of coordinates and managing its persistence, with functions to load and save matrices from disk.

2.2 Usage

The following code snippet illustrates the creation and access of a pairwise RMSD matrix of a 35 k frames trajectory, available in the 'benchmark/data' folder, using the QCP superposition algorithm, in its CUDA version:

```
from pyRMSD.matrixHandler import MatrixHandler
from pyRMSD.utils.proteinReading import Reader
cords = Reader().readThisFile('amber_35k.pdb')\
                    .gettingOnlyCAs().read()
matrix = MatrixHandler()\
                    .createMatrix(coords, 'QCP_OMP_CALCULATOR')
```

Here, we can find a minimum subset of all the features of pyRMSD and of the *MatrixHandler* class itself. However, it is a good example of how this class nicely encapsulates all the steps of creating a matrix and of the succinct interface presented to the user.

2.3 Performance

Using pyRMSD, we have coded a set of benchmark scripts to understand the performance differences between the three implemented algorithms. We have observed that, in all studied scenarios, QCP is the faster method.

We have also compared the performance of our four QCP implementations. Compared with the serial code, our OpenMP version is 5× faster; our CUDA-based implementation shows a 11× speedup (see Fig. 1). This leads us to conclude that GPU implementations can really make the difference in this kind of problems.

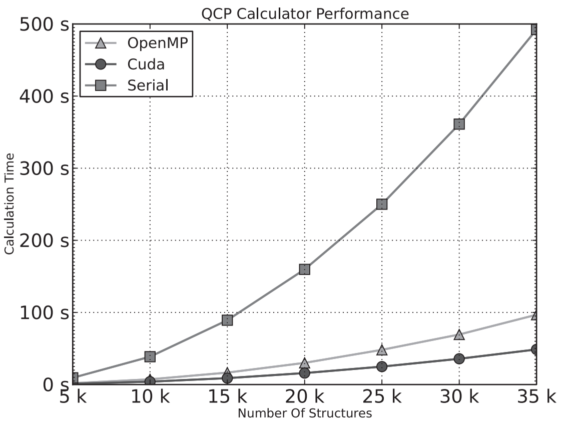

Fig. 1. QCP calculator performance over a Ubiquitin trajectory (only CAs) using a 6 cores Intel Xeon E5649 CPU with an NVIDIA M2090 GPU. OpenMP version reaches a 5× speedup. CUDA version gets a maximum 11× speedup (almost 12 million RMSD calculations per second)

These and other benchmarks, as well as a comparison with other packages, are discussed in depth in the Supplementary Data.

3 CONCLUSIONS

We have created pyRMSD, a user-friendly RMSD focused Python package, which allows, besides other functionalities, the efficient creation of RMSD pairwise matrices. Its design provides a natural way of accessing its functionalities making it a good candidate to be used in bigger packages to replace slower RMSD functions. This is specially true for those who need to calculate and access large pairwise RMSD matrices, as clustering-related packages.

Funding: European project PELE (ERC-2009-Adg 25027).

Conflict of Interest: none declared.

REFERENCES

Heisterberg,D.J. (1990) QTRFIT algorithm for superimposing two similar rigid molecules. The Ohio Supercomputer Center, Ohio State University, Columbus, OH.
Kabsch,W. (1978) A discussion of the solution for the best rotation to relate two sets of vectors. *Acta. Crystallogr. A*, **34**, 827–828.
Karpen,M.E. *et al.* (1993) Statistical clustering techniques for the analysis of long molecular dynamics trajectories: analysis of 2.2-ns trajectories of YPGDV. *Biochemistry*, **32**, 412–420.
Luxburg,U. (2007) A tutorial on spectral clustering. *Stat. Comp.*, **17**, 395–416.
Phillips,J.L. *et al.* (2011) Validating clustering of molecular dynamics simulations using polymer models. *BMC Bioinformatics*, **12**, 445.
Theobald,D.L. (2005) Rapid calculation of RMSDs using a quaternion-based characteristic polynomial. *Acta. Crystallogr. A*, **61**, 478–480.

Downloaded from http://bioinformatics.oxfordjournals.org/ by guest on February 7, 2016