CADISHI Documentation

Release 1.1.0

Klaus Reuter, Juergen Koefinger

Jan 17, 2019

Contents

CHAPTER 1

Introduction

The CADISHI package is designed to perform fast computations of histograms of the (Euclidean) distances between objects. For example, these objects may be atoms from molecular dynamics (MD) simulation datasets, or stars or galaxies from astrophysical datasets. While CADISHI is written mostly in Python, it is built upon high-performance kernels written in C++ and CUDA to exploit the CPU and GPU resources of a compute node as best as possible.

In the field of MD simulations, CADISHI is most useful in conjunction with the Capriqorn package. The name CADISHI is simply an acronym derived from (or, more precisely, the syllabic abbreviation of) *calculation of distance histograms*.

For more information, we refer to our publication:

K. Reuter, J. Koefinger; CADISHI: Fast parallel calculation of particle-pair distance histograms on CPUs and GPUs; https://arxiv.org/abs/1808.01478; 2018.

CHAPTER 2

Requirements

CADISHI requires a Python 2.7 or Python 3.X installation including the NumPy, SciPy, Cython, h5py, and PyYAML modules. We recommend to use the Anaconda Python Distribution which provides all the necessary modules out of the box. CADISHI was mostly developed using Anaconda Python 2, versions 4.0.0 and newer. Moreover, to compile the high-performance kernels, recent GCC and CUDA (optional) installations are required. GCC 4.9, 5.4, 6.4, and 7.2, and CUDA 7.5, 8.0, and 9.1 were used successfully. CADISHI was successfully tested on Linux and Mac machines.

CHAPTER 3

Features

CADISHI features a two-level parallelization. On the top level, CADISHI parallelizes over frames (i.e. snapshots of the ensemble of objects at certain points in time) using the Python multiprocessing module. A single frame is processed by a single worker process running a CPU or GPU kernel. Multiple worker processes may run simultaneously on a shared-memory multi-core machine. On the frame level, OpenMP threads are used by the CPU kernel, and CUDA threads are used by the GPU kernel. Hence, it is possible to fully exploit the resources of a shared-memory machine. E.g., on a dual-socket server with two GPUs, CADISHI would use two CPU worker processes (one per multi-core chip) and two GPU worker processes (one per GPU card).

CADISHI supports orthorhombic and triclinic periodic boxes, applying the minimum image convention to the distances internally.

Computations can be performed in single (default) or double precision. Optionally, the distances can be checked if they fit into the desired histogram width. Given the combinatorial explosion resulting from these possibilities, templated C++ code is used to generate machine code with a minimum amount of branches at runtime. Recent compilers are known to generate well-vectorized machine code from the distance calculation for the CPU. The GPU kernel benefits strongly from the fast shared-memory atomic operations introduced with the MAXWELL generation of NVIDIA GPUs.

CHAPTER 4

Input data, distance histogram computation, output data

CADISHI reads input data from an HDF5 file that is specified in the `histograms.yaml` parameter file. A typical file location is `./preprocessor_output/trajectory.h5` when CADISHI is used in concert with the Capriqorn package from the same authors. In any case the HDF5 file must have a certain internal structure as shown in the following example:

```
/1/coordinates/species_0
               species_1
               species_2
               ...
/2/coordinates/species_0
               species_1
               species_2
               ...
...
```

Frames are numbered starting with 1. The number is used as the label for the uppermost HDF5 group. For each frame the particle coordinates are stored in the sub-group 'coordinates'. Coordinate sets are double precision HDF5 datasets of size (n_i, 3) where n_i is the number of particles of species i. The coordinate datasets use the name of the species as the label which e.g. can be the name of the chemical element in the context of MD data.

To demonstrate how to feed data into CADISHI there is an **example code** available at `doc/scripts/input_example.py`. Adapt that code to easily implement a reader for arbitrary custom data.

To **convert data from a plethora of MD simulation data formats** to CADISHI's HDF5 format, we provide the script `md_to_cadishi.py` which is installed together with the main program `cadishi`. It uses the reader from the MDAnalysis package which needs to be installed to be able to use the conversion tool (but is not a requirement to run CADISHI).

For each frame read from the HDF5 file **CADISHI computes the distance histograms** between the particles for all combinations of species. The top-level parallelization of CADISHI is able to compute multiple frames simultaneously on all GPUs and CPUs available on a node. Within a frame, the computation is highly parallelized using threads.

Finally, **CADISHI writes the histograms into HDF5 files** according to the following scheme:

```
/1/histograms/species_0,species_0
              species_0,species_1
              species_0,species_2
              species_1,species_1
              species_1,species_2
              species_2,species_2
              ...
...
```

The HDF5 histogram datasets are single-column vectors of 64 bit floats. The numerical datatype was chosen to make averaging easier and more consistent.

To get an idea about all the options available please have a look at the example parameter file that comes with CADISHI. It can be generated using the command `cadishi example [--expert]`.

To **make life with HDF5 files easier** we recommend to use a graphical HDF5 viewer such as HDFView. Note, however, that HDFView does not support the LZF compression that comes with the Python HDF5 module "H5py" and is used by CADISHI by default (LZF can be disabled via the parameter file)).

Moreover, to demonstrate how to access the HDF5 data written by CADISHI from Python we provide the example program `doc/scripts/plot_hdf5_data.py`. It opens the HDF5 container, reads the last frame and plots a histogram using matplotlib. As an input file, the CADISHI output from the test case can be taken.

Finally, CADISHI comes with a HDF5 unpack tool (`cadishi unpack`) and a HDF5 merge tool (`cadishi merge`). The latter tool can also be used to decompress single HDF5 files quickly before viewing them with HDFView.

CHAPTER 5

Parameters

CADISHI is highly configurable via its YAML parameter file (create a sample file using `cadishi example [--expert]`). Below we pick the most important parameters and provide some background explanation.

- `histogram:dr`: Defines the histogram bin width. The total number of bins is determined in conjunction with the following parameter.
- `histogram:r_max`: Defines the histogram cutoff radius. Make sure that your input data is compatible with the cutoff radius, i.e. the maximum distance between all points must be smaller. In this context, the following settings are important.
- `cpu:check_input` and `gpu:check_input`: Checks at runtime if any distance computed is equal or smaller than the cutoff radius. If this is not the case, an exception is raised that causes CADISHI to exit. If these parameters are set to false, no checks are performed and outliers may write into non-allocated memory. In order to achieve the highest possible performance, make absolutely sure that your data fits into r_max and disable the checks.
- `input:periodic_box`: The default value is null, causing CADISHI to automatically look for the presence of box information in the input data. A box can as well be specified explicitly, or disabled by setting the value to an empty list [].

To achieve optimum **performance**, the number of workers and threads must be tuned to your system. For large frames (>O(100000) particles) it is reasonable to use few workers and a large number of threads per worker, e.g. if you have a 16 core CPU you could run a single cpu worker process with 14 threads, while keeping the 2 more cores busy with CADISHI's reader and writer processes (you always have these two processes). For small frames, use more workers and fewer (down to one) threads. Running some experiments is useful to gain experience with specific datasets. Note that CADISHI supports NUMA awareness, i.e. processes can be pinned to CPUs. Calculations on large frames *greatly* benefit from the GPU kernels. Enable one worker per GPU. We have seen a binning rate of up to 495 billion particle-pairs per second on a single Volta GPU.

CHAPTER 6

Installation

The package comes with a standard Python setup.py file. It is installed e.g. as follows into the user's homedirectory:

```
python setup.py install --user
```

In this case, setup.py copies the CADISHI files into the directory `~/.local`. Make sure that your `PATH` environment variable includes the directory `~/.local/bin`. To enable a CUDA build the `nvcc` compiler wrapper must be found via the `PATH` environment variable. Influential boolean environment variables are `CAD_DEBUG`, `CAD_OPENMP`, `CAD_CUDA`, and `CAD_SAFE_CUDA_FLAGS`; if set they override the keys listed in the options section of the file `setup.cfg`.

CHAPTER 7

Usage

The CADISHI package provides a single main executable `cadishi` which supports commands and options (similar to the well-known concept used by `git`).

To get a quick overview on the options of CADISHI issue the following command:

```
cadishi --help
```

To get started a parameter file controlling the distance histogram calculation is required. A basic example is included with the package and can be created as follows:

```
cadishi example [--expert]
```

By default, the parameter file's name is `histograms.yaml`. Edit the parameter file to your needs. In particular, the compute kernels *pydh* (CPU) and *cudh* (GPU) can be configured. In the example parameter file, the input configuration points to the default test case included with the CADISHI package. Adapt the input configuration to your actual data. An example code on how to import data into CADISHI is available at `doc/scripts/input_example.py`. As a second step the distance histogram calculation is run as follows:

```
cadishi histo
```

CHAPTER 8

Source documentation

The documentation linked below is generated automatically from the docstrings present in the CADISHI source code.

8.1 CADISHI modules

The following documentation was automatically generated from the docstrings present in the source code files.

8.1.1 cadishi

8.1.2 cadishi.base

8.1.3 cadishi.io.ascii

8.1.4 cadishi.io.dummy

8.1.5 cadishi.io.hdf5

8.1.6 cadishi.io.pickle

8.1.7 cadishi.dict_util

8.1.8 cadishi.dictfs

8.1.9 cadishi.h5pickle

8.1.10 cadishi.util

8.1.11 cadishi.version

8.2 CADISHI compute kernels

The following documentation was automatically generated from the docstrings present in the source code files.

8.2.1 cadishi.kernel

8.2.2 cadishi.kernel.common

8.2.3 cadishi.kernel.dist

8.2.4 cadishi.kernel.pydh

8.2.5 cadishi.kernel.cudh

8.3 CADISHI executables

The executables are accessible via the `cadishi` command.

The following documentation was automatically generated from the docstrings present in the source code files.

8.3.1 cadishi.exe

8.3.2 cadishi.exe.cli

8.3.3 cadishi.exe.histograms

8.3.4 cadishi.exe.unpack

CHAPTER 9

License and Citation

In case you're using CADISHI for your own academic or non-academic research, **we kindly request that you cite CADISHI in your publications and presentations**. We suggest the following citation as appropriate:

K. Reuter, J. Koefinger; CADISHI: Fast parallel calculation of particle-pair distance histograms on CPUs and GPUs; https://arxiv.org/abs/1808.01478; 2018.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search