Computer Physics Communications 185 (2014) 1546–1553

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

H5MD: A structured, efficient, and portable file format for molecular data

Pierre de Buyl [a,b,*], Peter H. Colberg [b], Felix Höfling [c,d]

[a] Center for Nonlinear Phenomena and Complex Systems, Université libre de Bruxelles, CP231, Campus Plaine, 1050 Brussels, Belgium
[b] Chemical Physics Theory Group, Department of Chemistry, University of Toronto, Toronto, Ontario M5S 3H6, Canada
[c] Max-Planck-Institut für Intelligente Systeme, Heisenbergstraße 3, 70569 Stuttgart, Germany
[d] IV. Institut für Theoretische Physik, Universität Stuttgart, Pfaffenwaldring 57, 70569 Stuttgart, Germany

ARTICLE INFO

Article history:
Received 1 September 2013
Received in revised form
20 January 2014
Accepted 29 January 2014
Available online 12 February 2014

Keywords:
Molecular simulation
HDF5

ABSTRACT

We propose a new file format named "H5MD" for storing molecular simulation data, such as trajectories of particle positions and velocities, along with thermodynamic observables that are monitored during the course of the simulation. H5MD files are HDF5 (Hierarchical Data Format) files with a specific hierarchy and naming scheme. Thus, H5MD inherits many benefits of HDF5, e.g., structured layout of multi-dimensional datasets, data compression, fast and parallel I/O, and portability across many programming languages and hardware platforms. H5MD files are self-contained, and foster the reproducibility of scientific data and the interchange of data between researchers using different simulation programs and analysis software. In addition, the H5MD specification can serve for other kinds of data (e.g. experimental data) and is extensible to supplemental data, or may be part of an enclosing file structure.

© 2014 Elsevier B.V. All rights reserved.

1. Introduction

Storing molecular data, such as particle coordinates, is a problem faced by simulation scientists in statistical physics, physical chemistry, structural biology, and crystallography, among others. Although there are currently various simulation packages that are related to specific fields of research and that define their own file formats, there is no standard file format for the storage and communication of molecular data.

The popular PDB file format of the Protein Data Bank [1] formats is accepted as input to several molecular dynamics programs (Gromacs [2], NAMD [3], CHARMM [4]); the original purpose of PDB, however, was to store reference crystallographic structures from experiments. PDB is not very practical as it is text-file based and uses a rigid organization. While a text file is simple to write, it uses more disk space than a binary file, it imposes a sequential reading of the data which can be time and memory intensive, and it prevents modularity of the stored data. Such technical considerations did not prevent researchers to base their work and software on PDB for running high-performance simulations and performing data analysis, in addition to using PDB for crystallographic data. For example, PDB focuses on the storage of simulation data at a single instance in time, but non-standard extensions of PDB are sometimes used and allow one to store several time frames. The documentations of the MDAnalysis software [5, Section 5.8] and of trjconv, a utility of Gromacs [2], mention explicitly this possibility. Other molecular dynamics packages use differently structured text files (ESPResSo [6,7], LAMMPS [8]) or resort to XML as a structured, but still text-based format (HOOMD [9]).

Many technical features are nowadays available for file formats that are highly desirable for storing scientific data; examples include self-descriptiveness, native support of data arrays, portability across hardware platforms, and availability of computer libraries for many programming languages. The absence of a reference file format of satisfactory technical quality does a disservice to the various communities that generate and analyze molecular data. Insights on this problem are presented in a recent publication by Hinsen [10]. Discussions on file formats for molecular data have also been held on the mailing list of Gromacs [11], suggesting that the need for technical solutions is recognized among experts in the field of molecular simulation.

Designing a new file format provides the opportunity to take into account progress in software engineering. Metadata on the person that authored a file, on the software that wrote it, and on the creation date can be included, for instance. This supports an approach of reproducible research with clear data provenance and definition [12]. On the way towards reproducible simulation-based

* Correspondence to: Department of Chemistry, Katholieke Universiteit Leuven, Celestijnenlaan 200F, B-3001 Heverlee, Belgium. Tel.: +32 16 3 27355.
E-mail address: pdebuyl@ulb.ac.be (P. de Buyl).

http://dx.doi.org/10.1016/j.cpc.2014.01.018
0010-4655/© 2014 Elsevier B.V. All rights reserved.

research, the availability of source code is only one part of an ensemble of requirements [13]. Another necessary building block in this process is the design of well-documented, published, royalty-free file formats.

Here we propose a file format named H5MD to store molecular data. It is based on the Hierarchical Data Format version 5 (HDF5) [14] as an underlying technology. H5MD specifies how to store and organize data for the trajectory of a complex many-particle system and for time-dependent or -independent observables. In addition, H5MD metadata are defined, and there is scope to store custom data such as flow fields of an embedding solvent, control parameters, or simulation scripts. The H5MD specification is available under the terms of the GNU General Public License version 3 (or any later version) [15], the development is hosted on the open-source platform Savannah [16], and discussions have been held from the early stages of the project on a public mailing list [17]. With the objectives to publish the specification, to ease the use of H5MD, to share good practice and practical information on the development of software using H5MD, we also use the website feature of the Savannah platform, see Ref. [18]. H5MD could find applications in simulation programs (molecular dynamics, fluid dynamics, etc.), experimental data acquisition (e.g., particle tracking, particle imaging velocimetry), and the analysis tools related to these data.

An outline of the article follows. Section 2 discusses the requirements faced with regard to storage of scientific data, leading to the choice of HDF5 as the basis of H5MD. Section 3 presents the H5MD format specification version 1.0.0. Section 4 presents use cases from simulation software that implement H5MD and a Python library that is accompanied by illustrative examples.

2. Design considerations

H5MD is developed with several design goals in mind:

Versatility: H5MD does not adhere to a specific application or research project. Rather it adopts an abstract language to define generic data elements and structures common to molecular data. H5MD files can be used to store data from particle-based simulations, but also molecular structures from experimental measurements, or they can serve as input for subsequent investigations.

Modularity: H5MD allows one to store heterogeneous data independently of each other within one file to reflect several facets of a complex simulation. For example, the particle trajectory may be combined with the sampling of thermodynamic observables during the simulation. These data may be split further into subsets of a complex system, e.g., the atoms forming colloidal nanoparticles and the surrounding solvent molecules.

Extensibility: H5MD files may store additional data that is not part of the specification without breaking its H5MD compliance. Storing application-specific data such as control parameters or simulation scripts fosters the reproducibility of scientific results.

Self-descriptiveness: A data element stored within H5MD carries a descriptive name and specifies its data type and array shape. Creation and modification of data elements can be tracked. Data elements can be annotated further by custom metadata.

Efficiency: H5MD files allow for high-throughput I/O and compression to reduce storage requirements. Further, random access to data subsets is fast and (nearly) constant in time.

Portability: H5MD files can be read and written across many different operating systems and computer architectures. Access to H5MD files can be achieved easily by a large variety of programming languages. In particular, H5MD can be integrated natively into existing software.

The goals "Modularity", "Extensibility", and "Self-descriptiveness" call for a structured file format. A binary and structured file format is demanded by the requirement "Efficiency", and "Portability" is ensured by resorting to an I/O library that is well supported by and established in the scientific community.

The HDF5 file format [14] is an ideal technology to meet these requirements. HDF5 is a mature, structured, binary, and portable file format for storing scientific data in a self-describing manner and consists of a specification and an I/O library. Data is stored in named datasets that are organized in groups similarly to the tree structure of a filesystem. Another significant benefit over non-binary or non-structured file formats are the facilities of HDF5 for parallel input/output, for data compression, and for random access to parts of a multi-dimensional dataset without reading the full dataset (array slicing). The HDF5 library provides programming interfaces for C, C++, and Fortran; Java and Python bindings are available, and some commercial packages (e.g., Matlab, Mathematica, IDL) support HDF5. Still there remains a great deal of choices to be made with respect to the organization of the file, the naming scheme, the coherence of time-dependent information, etc. Those choices form the H5MD specification.

The authors took care to respect HDF5 technicalities [19,20]. Creation and modification times in a H5MD file are recorded implicitly using the object tracking feature of HDF5. Compact Datasets are used for small amounts of data (i.e., when the data size is independent of the system size). Data forming a time series (e.g., particle positions in the course of a simulation) is stored as a single dataset, where the time axis is accommodated by an extra, extensible dimension prepended to the array shape. This approach is preferred over storing the sequence of frames in a sequence of differently named datasets (or groups), avoiding *inter alia* the clobbering of the file structure for a large number of frames. Since molecular data are sometimes time-dependent and sometimes time-independent, H5MD users may choose for any data item among two well-defined, generic data structures.

The design of H5MD also strives to be future-proof in that it allows for future extensions and avoids setting in stone features that appear not yet mature or generic enough. For instance, the storage of protein topology or crystallographic symmetry groups are not part of the specification though the authors recognize their importance. The self-describing and organized nature of HDF5 allows additions in the form of new (root) groups or datasets without breaking the compliance of the file with H5MD. On the other hand, the H5MD format is defined as an HDF5 group within an HDF5 file and a single HDF5 file may thus contain several H5MD root groups, allowing for the collection of data in a single file and for making H5MD part of an enclosing file structure, e.g., in the Mosaic data model [21,22].

Some features or constraints are included in H5MD by so-called *modules*. Modules are aimed at specific applications or domains in which agreed upon conventions exist and should be reflected in the data. Two such modules are presented in the Appendix. The module "Units", for instance, describes how physical units can be meaningfully attached to H5MD data.

3. H5MD format specification version 1.0.0

3.1. Objective

H5MD stands for "HDF5 for molecular data". H5MD is a specification to store molecular simulation data and is based on the HDF5 file format [14]. The primary goal is to facilitate the portability of said data amongst scientific simulation and analysis programs.

3.2. File format

H5MD structures are stored in the HDF5 file format version 0 or later. It is recommended to use the HDF5 file format version 2,

which includes the implicit tracking of the creation and modification times of the file and of each of its objects.

3.3. Notation and naming

HDF5 files are organized into groups and datasets, summarized as *objects*, which form a tree structure with the datasets as leaves. Attributes can be attached to each object. The H5MD specification adopts this naming and uses the following notation to depict the tree or its subtrees:

`\-- item` An object within a group, that is either a dataset or a group. If it is a group itself, the objects within the group are indented by five spaces with respect to the group name.

`+-- attribute` An attribute, that relates either to a group or a dataset.

`\-- data: <type>[dim1][dim2]` A dataset with array dimensions `dim1` by `dim2` and of type `<type>`. The type is taken from `Enumeration`, `Integer`, `Float` or `String` and follows the HDF5 Datatype classes. If the type is not mandated by H5MD, `<type>` is indicated. A scalar dataspace is indicated by `[]`.

`(identifier)` An optional item.

`<identifier>` An optional item with unspecified name.

H5MD defines a structure called *H5MD element* (or *element* whenever there is no confusion). An element is either a time-dependent group or a single dataset (see time-dependent data below), depending on the situation.

3.4. General organization

H5MD defines an organization of the HDF5 file or a part thereof into groups, datasets, and attributes. The root level of the H5MD structure may coincide with the root of the HDF5 file or be an arbitrary group inside the HDF5 tree. A number of groups are defined at the H5MD root level. Several levels of subgroups may exist inside the H5MD structure, allowing the storage and description of subsystems.

The H5MD structure is allowed to possess non-specified groups, datasets, or attributes that contain additional information such as application-specific parameters or data structures, leaving scope for future extensions. Only the `h5md` group is mandatory at the H5MD root level. All other root groups are optional, allowing the user to store only relevant data. Inside each group, every group or dataset is again optional, unless specified differently.

H5MD supports equally the storage of time-dependent and time-independent data, i.e., data that change in the course of the simulation or that do not. The choice between those storage types is not made explicit for the elements in the specification, it has to be made according to the situation. For instance, the species and mass of the particles are often fixed in time, but in chemically reactive systems this might not be appropriate.

3.4.1. Time-dependent data

Time-dependent data consist of a series of samples (or frames) referring to multiple time steps. Such data are found inside a single dataset and are accessed via dataset slicing. In order to link the samples to the time axis of the simulation, H5MD defines a *time-dependent H5MD element* as a group that contains, in addition to the actual data, information on the corresponding integer time step and on the physical time. The structure of such a group is:

```
<element>
 \-- step: Integer[variable]
 \-- time: Float[variable]
 \-- value: <type>[variable][...]
```

`step` A dataset with dimensions `[variable]` that contains the time steps at which the corresponding data were sampled. It is of `Integer` type to allow exact temporal matching of data from one H5MD element to another. The values of the dataset are in monotonically increasing order.

`time` A dataset that is the same as the `step` dataset, except it is `Float`-valued and contains the simulation time in physical units. The values of the dataset are in monotonically increasing order.

`value` A dataset that holds the data of the time series. It uses a simple dataspace whose rank is given by 1 plus the tensor rank of the data stored. Its shape is the shape of a single data item prepended by a `[variable]` dimension that allows the accumulation of samples during the course of time. For instance, the data shape of scalars has the form `[variable]`, `D`-dimensional vectors use `[variable][D]`, etc. The first dimension of `value` must match the unique dimension of `step` and `time`.

If several H5MD elements are sampled at equal times, `step` and `time` of one element may be hard links to the `step` and `time` datasets of a different element. If two elements are sampled at different times (for instance, if one needs the positions more frequently than the velocities), `step` and `time` are unique to each of them.

3.4.2. Time-independent data

H5MD defines a *time-independent H5MD element* as a dataset. As for the `value` dataset in the case of time-dependent data, data type and array shape are implied by the stored data, where the `[variable]` dimension is omitted.

3.4.3. Storage order of arrays

All arrays are stored in C-order as enforced by the HDF5 file format (see Section 3.2.5 in [23]). A C or C++ program may thus declare `r[N][D]` for the array of particle coordinates while the Fortran program will declare a `r(D,N)` array (appropriate index ordering for a system of `N` particles in `D` spatial dimensions), and the HDF5 file will be the same.

3.5. H5MD root level

The root level of an H5MD structure holds a number of groups and is organized as follows:

```
H5MD root
 \-- h5md
 \-- (particles)
 \-- (observables)
 \-- (parameters)
```

`h5md` A group that contains metadata and information on the H5MD structure itself. It is the only mandatory group at the root level of H5MD.

`particles` An optional group that contains information on each particle in the system, e.g., a snapshot of the positions or the full trajectory in phase space. The size of the stored data scales linearly with the number of particles under consideration.

`observables` An optional group that contains other quantities of interest, e.g., physical observables that are derived from the system state at given points in time. The size of stored data is typically independent of the system size.

`parameters` An optional group that contains application-specific, custom data such as control parameters or simulation scripts.

In subsequent sections, the examples of HDF5 organization may start at the group level, omitting the display of `H5MD root`.

3.6. H5MD metadata

A set of global metadata describing the H5MD structure is stored in the `h5md` group as attributes. The contents of the group is:

```
h5md
 +-- version: Integer[2]
 \-- author
 |    +-- name:  String[]
 |    +-- (email:  String[])
 \-- creator
      +-- name:  String[]
      +-- version:  String[]
```

`version` An attribute, of `Integer` datatype and of simple dataspace of rank 1 and size 2, that contains the major version number and the minor version number of the H5MD specification the H5MD structure conforms to.

The version *x.y.z* of the H5MD specification follows semantic versioning [24]: A change of the major version number *x* indicates backwards-incompatible changes to the file structure. A change of the minor version number *y* indicates backwards-compatible changes to the file structure. A change of the patch version number *z* indicates changes that have no effect on the file structure and serves to allow for clarifications or minor text editing of the specification.

As the *z* component has no impact on the content of a H5MD file, the `version` attribute contains only *x* and *y*.

`author` A group that contains metadata on the person responsible for the simulation (or the experiment) as follows:

- `name` An attribute, of fixed-length string datatype and of scalar dataspace, that holds the author's real name.
- `email` An optional attribute, of fixed-length string datatype and of scalar dataspace, that holds the author's email address of the form `email@domain.tld`.

`creator` A group that contains metadata on the program that created the H5MD structure as follows:

- `name` An attribute, of fixed-length string datatype and of scalar dataspace, that stores the name of the program.
- `version` An attribute, of fixed-length string datatype and of scalar dataspace, that yields the version of the program.

3.6.1. Modules

The H5MD specification can be complemented by modules specific to a domain of research. A module may define additional data elements within the H5MD structure, add conditions that the data must satisfy, or define rules for their semantic interpretation. Multiple modules may be present, as long as their prescriptions are not contradictory. Each module is identified by a name and a version number.

The modules that apply to a specific H5MD structure are stored as subgroups within the group `h5md/modules`. Each module holds its version number as an attribute, further module-specific information may be stored:

```
h5md
 \-- (modules)
      \-- <module1>
      |    +-- version: Integer[2]
      \-- <module2>
      |    +-- version: Integer[2]
      \-- ...
```

`version` An attribute, of `Integer` datatype and of simple dataspace of rank 1 and size 2, that contains the major version number and the minor version number of the module.

The version *x.y.z* of an H5MD module follows semantic versioning [24] and again only the components *x* and *y* are stored, see `h5md/version` in "H5MD metadata".

3.7. Particles group

Information on each particle, i.e., particle trajectories, is stored in the `particles` group. The `particles` group is a container for subgroups that represent different subsets of the system under consideration, and it may hold one or several subgroups, as needed. These subsets may overlap and their union may be incomplete, i.e., not represent all particles of the simulation volume. The subgroups contain the trajectory data for each particle as time-dependent or time-independent data, depending on the situation. Each subgroup contains a specification of the simulation box, see below. For each dataset, the particle index is accommodated by the second (first, in the case of time-independence) array dimension.

The contents of the `particles` group assuming `N` particles in `D`-dimensional space could be the following:

```
particles
 \-- <group1>
      \-- box
      \-- (position)
      |    \-- step: Integer[variable]
      |    \-- time: Float[variable]
      |    \-- value: <type>[variable][N][D]
      \-- (image)
      |    \-- step: Integer[variable]
      |    \-- time: Float[variable]
      |    \-- value: <type>[variable][N][D]
      \-- (species: Enumeration[N])
      \-- ...
```

The following identifiers for H5MD elements are standardized:

`position` An element that describes the particle positions as coordinate vectors of `Float` or `Integer` type.

If the component *k* of `box/boundary` (see below) is set to `none`, the data indicate for each particle the component *k* of its absolute position in space. If the component *k* of `box/boundary` is set to `periodic`, the data indicate for each particle the component *k* of the absolute position in space of an *arbitrary* periodic image of that particle.

`image` An element that represents periodic images of the box as coordinate vectors of `Float` or `Integer` type and allows one to compute for each particle its absolute position in space. If `image` is present, `position` must be present as well. For time-dependent data, the `step` and `time` datasets of `image` must equal those of `position`, which must be accomplished by hard-linking the respective datasets.

If the component *k* of `box/boundary` (see below) is set to `none`, the values of the corresponding component *k* of `image` serve as placeholders. If the component *k* of `box/boundary` is set to `periodic`, for a cuboid box, the component *k* of the absolute position of particle *i* is computed as $R_{ik} = r_{ik} + L_k a_{ik}$, where $\mathbf{r}_i$ is taken from `position`, $\mathbf{a}_i$ is taken from `image`, and $\mathbf{L}$ from `box/edges`.

`velocity` An element that contains the velocities for each particle as a vector of `Float` or `Integer` type.

`force` An element that contains the total forces (i.e., the accelerations multiplied by the particle mass) for each particle as a vector of `Float` or `Integer` type.

`mass` An element that holds the mass for each particle as a scalar of `Float` type.

`species` An element that describes the species for each particle, i.e., its atomic or chemical identity, as a scalar of `Enumeration` or `Integer` data type. Particles of the same species are assumed to be identical with respect to their properties and unbonded interactions.

`id` An element that holds a scalar identifier for each particle of `Integer` type, which is unique within the given particle subgroup. The `id` serves to identify particles over the course of the simulation in the case when the order of the particles changes, or when new particles are inserted and removed. If `id` is absent, the identity of the particles is given by their index in the `value` datasets of the elements within the same subgroup.

A *fill value* (see Section 6.6 in [23]) may be defined for `id/value` upon dataset creation. When the identifier of a particle is equal to this user-defined value, the particle is considered non-existing, the entry serves as a placeholder. This permits the storage of subsystems whose number of particles varies in time. For the case of varying particle number, the dimension denoted by `[N]` above may be variable.

3.8. Simulation box

The specification of the simulation box is stored in the group `box`, which must be contained within each of the subgroups of the `particles` group. Storing the box information at several places reflects the fact that different subgroups may be sampled at different time grids. This way, the box information remains associated to a group of particles. A specific requirement for `box` groups inside `particles` is that the `step` and `time` datasets exactly match those of the corresponding `position` groups, which must be accomplished by hard-linking the respective datasets.

The spatial dimension and the boundary conditions of the box are stored as attributes to the `box` group, e.g., :

```
particles
 \-- <group1>
      \-- box
           +-- dimension: Integer[]
           +-- boundary:  String[D]
           \-- (edges)
```

`dimension` An attribute that stores the spatial dimension `D` of the simulation box and is of `Integer` datatype and scalar dataspace.

`boundary` An attribute, of fixed-length string datatype and of simple dataspace of rank 1 and size `D`, that specifies the boundary condition of the box along each dimension. The values in `boundary` are either `periodic` or `none`:

`periodic` The simulation box is periodically continued along the given dimension and serves as the unit cell for an infinite tiling of space.

`none` No boundary condition is imposed. This summarizes the situations of open systems (i.e., an infinitely large box) and closed systems (e.g., due to an impenetrable wall). For those components where `boundary` is set to `none`, the corresponding values of `edges` serve as placeholders.

Information on the geometry of the box edges is stored as an H5MD element, allowing for the box to be fixed in time or not. Supported box shapes are the cuboid and triclinic unit cell, for other shapes a transformation to the triclinic shape may be considered [25]. If all values in `boundary` are `none`, `edges` may be omitted.

`edges` A `D`-dimensional vector or a $D \times D$ matrix, depending on the geometry of the box, of `Float` or `Integer` type. If `edges` is a vector, it specifies the space diagonal of a cuboid-shaped box. If `edges` is a matrix, the box is of triclinic shape with the edge vectors given by the rows of the matrix.

For a time-dependent box, a cuboid geometry is encoded by a dataset `value` (within the H5MD element) of rank 2 (1 dimension for the time and 1 for the vector) and a triclinic geometry by a dataset `value` of rank 3 (1 dimension for the time and 2 for the matrix).

For a time-independent box, a cuboid geometry is encoded by a dataset `edges` of rank 1 and a triclinic geometry by a dataset of rank 2.

For instance, a cuboid box that changes in time would appear as:

```
particles
 \-- <group1>
      \-- box
           +-- dimension: Integer[]
           +-- boundary:  String[D]
           \-- edges
                \-- step: Integer[variable]
                \-- time: Float[variable]
                \-- value: <type>[variable][D]
```

where `dimension` is equal to `D`. A triclinic box that is fixed in time would appear as:

```
particles
 \-- <group1>
      \-- box
           +-- dimension: Integer[]
           +-- boundary:  String[D]
           \-- edges: <type>[D][D]
```

where `dimension` is equal to `D`.

3.9. Observables group

Macroscopic observables, or more generally, averages of some property over many particles, are stored in the root group `observables`. Observables representing only a subset of the particles may be stored in appropriate subgroups similarly to the `particles` tree. Each observable is stored as an H5MD element. The shape of the corresponding dataset (the element itself for time-independent data and `value` for time-dependent data) is the tensor shape of the observable prepended by a `[variable]` dimension for time-dependent data.

The contents of the observables group has the following structure:

```
observables
 \-- <observable1>
 |    \-- step: Integer[variable]
 |    \-- time: Float[variable]
 |    \-- value: <type>[variable]
 \-- <observable2>
 |    \-- step: Integer[variable]
 |    \-- time: Float[variable]
 |    \-- value: <type>[variable][D]
 \-- <group1>
 |    \-- <observable3>
 |         \-- step: Integer[variable]
 |         \-- time: Float[variable]
 |         \-- value: <type>[variable][D][D]
 \-- <observable4>: <type>[]
 \-- ...
```

3.10. Parameters group

The `parameters` group stores application-specific, custom data such as control parameters or simulation scripts. The group consists of groups, datasets, and attributes; the detailed structure, however, is left unspecified.

The contents of the `parameters` group could be the following:

```
parameters
 +-- <user_attribute1>
 \-- <user_data1>
 \-- <user_group1>
 |    \-- <user_data2>
 |    \-- ...
 \-- ...
```

4. Uses and tools for H5MD

H5MD evolved from custom HDF5 file structures that were used by the authors in their simulation-based research. After discussing the common need for a file format and deciding to join efforts, the design of H5MD started from the experience of "HAL's MD package" and the RMPCDMD software.

"HAL's MD package" [26] exploits the vastly parallel architecture of modern graphics processors (GPUs) and the CUDA framework (Nvidia Corp., Santa Clara, CA) to deliver accurate and efficient, highly-accelerated large-scale molecular dynamics simulations [27]. The software has been written in C++ and Lua by P.H.C. and F.H., and the package is now maintained by F.H. It features a modular design along with a scripting user interface and permits the simulation of some 10^6 particles on a single GPU. The focus on the online evaluation of thermodynamic observables and of dynamic correlation functions minimizes the need to dump particle-resolved data to the disk. The software targets problems in statistical physics, applications include the slow glassy dynamics of binary mixtures [27] and the structure of liquid–vapor interfaces [28].

RMPCDMD is a Fortran hydrodynamics simulation code developed by P.d.B. based on the Multi-particle Collision Dynamics algorithm used for solvent dynamics, and the code is able to perform molecular dynamics of embedded colloids. In addition, RMPCDMD allows the simulation of reactive fluids. RMPCDMD uses H5MD to store the particle coordinates of the colloids and the solvent (for checkpointing only, up to 2 millions solvent particles tested) and all time-dependent data in the simulation. The H5MD code of RMPCDMD is separate from the simulation code and is provided as the f90h5md [29] Fortran module, by the same author.

In order to demonstrate the use of H5MD in a minimal self-contained example, we provide a code for the simulation of a 1D random walk of an ensemble of particles. The example is written in Python using the packages NumPy, h5py [30], and matplotlib, which are shipped with major GNU/Linux distributions or scientific Python distributions. The code is composed of the module pyh5md [31] (available as supplementary material can be found online at http://dx.doi.org/10.1016/j.cpc.2014.01.018 and at the Python Package Index, http://pypi.python.org/) that provides H5MD functionality on top of h5py, and the simulation script `random_walk_1d.py`. The management of H5MD data is handled by pyh5md, which results in a very small number of instructions in the actual program: opening the file, declaring each data element (particles group with position dataset, and observables), and writing the data. Excluding comments and blank lines, `random_walk_1d.py` contains 18 lines of code. The code propagates particles and stores their positions in the course of time in a H5MD file. The center of mass of the system is stored as an observable in the file. After running the simulation, an analysis of the mean-squared displacement is presented in `random_walk_1d_analysis.py`. It illustrates the opening of a file and the acquisition of trajectory data in about 30 lines of code.

5. Conclusions and perspectives

To summarize, we propose H5MD as a new file format to store molecular data. We have also presented a simple implementation of H5MD to serve as an illustration and a starting point for potential users. This simple implementation, along with the fact that H5MD is employed by all authors in their simulation codes, shows the applicability of the file format to real-world use cases.

It is our hope that scientists from related fields will participate in the further development of H5MD. The modular nature of H5MD and its open development process allow for future extensions that make the file format suitable for a wide range of applications. In particular, we believe that H5MD has the potential to provide a superb alternative to the PDB file format. H5MD provides the basis for storing particle-based data that form an important part of PDB (specifically, the "ATOM" records). Additional data, many of which consist in named parameters, would fit easily as named datasets or attributes in an HDF5 file. Other interesting extensions are related to the storage of grid data, for example when mesoscopic particles are coupled to flow fields as in computational fluid dynamics or lattice-Boltzmann approaches.

Acknowledgments

We are grateful to Konrad Hinsen for his numerous contributions on the H5MD mailing list and to Olaf Lenz for correspondence, discussions, and comments on the manuscript. We acknowledge Savannah (the free software forge) for hosting the H5MD project. We thank Bryan Robertson and Cyrille Lavigne for critically reading the manuscript. P.d.B. and P.H.C. wish to thank Raymond Kapral for his support.

Appendix. Modules

A.1. Thermodynamic observables

A.1.1. Objective

This module defines a set of thermodynamic observables commonly output by molecular simulation programs.

A.1.2. Module name and version

The name of this module is `thermodynamics`. The module version is 1.0.0.

A.1.3. Observables

Thermodynamic observables are stored in the `observables` group for global properties or in `observables/<group>` for subsystems, similarly to the `particles` group. The groups have the following contents:

```
observables
 \-- <group>
      +-- dimension: Integer[]
      \-- particle_number
      \-- (pressure)
      \-- (temperature)
      \-- (density)
      \-- ...
```

`dimension` A scalar attribute of `Integer` type that gives the dimension of the space embedding the subsystem.

`particle_number` The number of particles in the subsystem stored as scalar H5MD element of `Integer` datatype.

The following H5MD elements are optional, of scalar character, and use the `Float` datatype.

`pressure` The pressure of the subsystem.
`temperature` The (instantaneous) temperature of the subsystem as inferred from the kinetic energy.
`density` The number density of the subsystem stored as `Float` or `Integer` datatype.
`potential_energy` The potential energy of the subsystem.
`kinetic_energy` The kinetic energy of the subsystem.
`internal_energy` The internal energy of the subsystem, typically the sum of potential and kinetic energy.
`enthalpy` The enthalpy of the subsystem.

The latter 4 quantities are stored as per-particle averages. The corresponding extensive quantities (scaling linearly with the system size) are obtained by multiplication with the number of particles. Per-volume averages follow by multiplication with the number density if present.

A.2. Units

A.2.1. Objective

This module defines how physical units are attached to dimensionful H5MD elements.

A.2.2. Module name and version

The name of this module is `units`. The module version is 1.0.0.

A.2.3. Unit system definition

The `units` group possesses, in addition to the `version` attribute, a `system` attribute that defines the unit system in use. `system` is of scalar dataspace and fixed-length string datatype.

A.2.4. Unit attribute

The datasets of any H5MD element that have a physical dimension may carry an attribute `unit` to indicate the physical unit of the respective data. In general, this refers to the dataset itself for time-independent elements, or to the datasets `value` and `time` in the time-dependent case:

```
<element>
 \-- step: Integer[variable]
 \-- time: Float[variable]
 |     +-- (unit:  String[])
 \-- value: <type>[variable][...]
       +-- (unit:  String[])
```

The attribute `unit` is of scalar dataspace and fixed-length `String` datatype using the ASCII character set.

A.2.5. Unit string

The `unit` string consists of a sequence of unit factors separated by a space. A unit factor is either a number (an integer or a decimal fraction) or a unit symbol optionally followed by a non-zero, signed integer indicating the power to which this factor is raised. Each unit symbol may occur only once. There may also be at most one numeric factor, which must be the first one.

Examples:

- "nm+3" stands for cubic nanometers
- "um+2 s−1" stands for micrometers squared per second
- "60 s" stands for a minute
- "10+3m" stands for a kilometer

A.2.6. The "SI" unit system

The "SI" unit system [32] defines SI base units (Table A.1), SI derived units (Table A.2), and SI prefixes (Table A.3).

Table A.1
SI base unit symbols and names.

Dimension	Symbol	Unit name
Length	m	meter
Mass	kg	kilogram
Time	s	second
Electric current	A	ampere
Temperature	K	kelvin
Amount of substance	mol	mole
Luminous intensity	cd	candela

Table A.2
SI derived unit symbols, names and conversion rules.

Dimension	Symbol	Unit name	Conversion
Plane angle	rad	radian	$1\ \mathrm{rad} = 1\ \mathrm{m\ m^{-1}}$
Solid angle	sr	steradian	$1\ \mathrm{sr} = 1\ \mathrm{m^2\ m^{-2}}$
Frequency	Hz	hertz	$1\ \mathrm{Hz} = 1\ \mathrm{s^{-1}}$
Force	N	newton	$1\ \mathrm{N} = 1\ \mathrm{m\ kg\ s^{-2}}$
Pressure/stress	Pa	pascal	$1\ \mathrm{Pa} = 1\ \mathrm{N\ m^{-2}}$
Energy/work	J	joule	$1\ \mathrm{J} = 1\ \mathrm{N\ m}$
Power	W	watt	$1\ \mathrm{W} = 1\ \mathrm{J\ s^{-1}}$
Electric charge	C	coulomb	$1\ \mathrm{C} = 1\ \mathrm{A\ s}$
Voltage	V	volt	$1\ \mathrm{V} = \mathrm{W\ A^{-1}}$
Capacitance	F	farad	$1\ \mathrm{F} = \mathrm{C\ V^{-1}}$
Electric resistance	ohm	ohm	$1\ \Omega = \mathrm{V\ A^{-1}}$
Electric conductance	S	siemens	$1\ \mathrm{S} = 1\ \mathrm{A\ V^{-1}}$
Magnetic flux	Wb	weber	$1\ \mathrm{Wb} = 1\ \mathrm{V\ s}$
Magnetic flux density	T	tesla	$1\ \mathrm{T} = 1\ \mathrm{Wb\ m^{-2}}$
Inductance	H	henry	$1\ \mathrm{H} = 1\ \mathrm{Wb\ A^{-1}}$
Celsius temperature	degC	degree Celsius	$0\ ^\circ\mathrm{C} = 273.15\ \mathrm{K}$
Luminous flux	lm	lumen	$1\ \mathrm{lm} = 1\ \mathrm{cd\ sr}$
Illuminance	lx	lux	$1\ \mathrm{lx} = 1\ \mathrm{lm\ m^{-2}}$
Radioactivity	Bq	becquerel	$1\ \mathrm{Bq} = 1\ \mathrm{s^{-1}}$
Absorbed dose	Gy	gray	$1\ \mathrm{Gy} = 1\ \mathrm{J\ kg^{-1}}$
Dose equivalent	Sv	sievert	$1\ \mathrm{Sv} = 1\ \mathrm{J\ kg^{-1}}$
Catalytic activity	kat	katal	$1\ \mathrm{kat} = 1\ \mathrm{mol\ s^{-1}}$

Table A.3
SI prefixes.

Prefix	Symbol	Factor
exa-	E	10^{18}
peta-	P	10^{15}
tera-	T	10^{12}
giga-	G	10^{9}
mega-	M	10^{6}
kilo-	k	10^{3}
hecto-	h	10^{2}
deca-	da	10^{1}
deci-	d	10^{-1}
centi-	c	10^{-2}
milli-	m	10^{-3}
micro-	u	10^{-6}
nano-	n	10^{-9}
pico-	p	10^{-12}
femto-	f	10^{-15}
atto-	a	10^{-18}

References

[1] H. Berman, K. Henrick, H. Nakamura, Announcing the world wide protein data bank, Nature Struct. Biol. 10 (2003) 980. http://dx.doi.org/10.1038/nsb1203-980.
[2] D. Van Der Spoel, E. Lindahl, B. Hess, G. Groenhof, A.E. Mark, H.J.C. Berendsen, GROMACS: fast, flexible, and free, J. Comput. Chem. 26 (16) (2005) 1701–1718. http://dx.doi.org/10.1002/jcc.20291.
[3] J.C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R.D. Skeel, L. Kalé, K. Schulten, Scalable molecular dynamics with NAMD, J. Comput. Chem. 26 (16) (2005) 1781–1802. http://dx.doi.org/10.1002/jcc.20289.
[4] B.R. Brooks, C.L. Brooks III, A.D. Mackerell Jr., L. Nilsson, R.J. Petrella, B. Roux, Y. Won, G. Archontis, C. Bartels, S. Boresch, A. Caflisch, L. Caves, Q. Cui, A.R. Dinner, M. Feig, S. Fischer, J. Gao, M. Hodoscek, W. Im, K. Kuczera, T. Lazaridis, J. Ma, V. Ovchinnikov, E. Paci, R.W. Pastor, C.B. Post, J.Z. Pu, M. Schaefer, B. Tidor, R.M. Venable, H.L. Woodcock, X. Wu, W. Yang, D.M. York,

M. Karplus, CHARMM: the biomolecular simulation program, J. Comput. Chem. 30 (10, Sp. Iss. SI) (2009) 1545–1614. http://dx.doi.org/10.1002/jcc.21287.
[5] N. Michaud-Agrawal, E.J. Denning, T.B. Woolf, O. Beckstein, MDAnalysis: a toolkit for the analysis of molecular dynamics simulations, J. Comput. Chem. 32 (10) (2011) 2319–2327. http://dx.doi.org/10.1002/jcc.21787.
[6] A. Arnold, O. Lenz, S. Kesselheim, R. Weeber, F. Fahrenberger, D. Roehm, P. Košovan, C. Holm, ESPResSo 3.1—molecular dynamics software for coarse-grained models, in: M. Griebel, M.A. Schweitzer (Eds.), Meshfree Methods for Partial Differential Equations VI, in: Lecture Notes in Computational Science and Engineering, vol. 89, Springer, 2013, pp. 1–23. http://dx.doi.org/10.1007/978-3-642-32979-1_1.
[7] H.J. Limbach, A. Arnold, B.A. Mann, C. Holm, ESPResSo—an extensible simulation package for research on soft matter systems, Comput. Phys. Comm. 174 (9) (2006) 704–727. http://dx.doi.org/10.1016/j.cpc.2005.10.005.
[8] S. Plimpton, A. Thompson, P. Crozier, LAMMPS users manual, June 2013. http://lammps.sandia.gov/doc/Manual.pdf.
[9] J. Anderson, A. Travesset, R. Sknepnek, Documentation of HOOMD-blue, version 0.11.3, 2013. http://codeblue.umich.edu/hoomd-blue/downloads/0.11/hoomd-userdoc-0.11.3.pdf.
[10] K. Hinsen, Caring for your data, Comput. Sci. Eng. 14 (6) (2012) 70–74. http://dx.doi.org/10.1109/MCSE.2012.108.
[11] Gromacs mailing list, searching for HDF5 reveals several discussions. http://www.gromacs.org/Support/Mailing_Lists/GMX-developers_List.
[12] R.D. Peng, Reproducible research in computational science, Science 334 (6060) (2011) 1226–1227. http://dx.doi.org/10.1126/science.1213847.
[13] D.C. Ince, L. Hatton, J. Graham-Cumming, The case for open computer programs, Nature 482 (2012) 485–488. http://dx.doi.org/10.1038/nature10836.
[14] The HDF Group, Hierarchical Data Format version 5, 2000–2013. http://www.hdfgroup.org/HDF5.
[15] Free Software Foundation, Inc., GNU General Public License version 3, 2007. https://www.gnu.org/licenses/gpl.html.
[16] Savannah. http://savannah.nongnu.org/.
[17] H5MD-user mailing list. https://lists.nongnu.org/mailman/listinfo/h5md-user.
[18] The H5MD website. http://nongnu.org/h5md/.
[19] The HDF Group, Advanced topics in HDF5, 2000–2013. http://www.hdfgroup.org/HDF5/doc/Advanced.html.
[20] The HDF Group, HDF5: API specification reference manual, 2000–2013. http://www.hdfgroup.org/HDF5/doc/RM/RM_H5Front.html.
[21] K. Hinsen, Mosaic—the molecular simulation interchange conventions. http://mosaic-data-model.github.io/.
[22] K. Hinsen, MOSAIC: a data model and file formats for molecular simulations, J. Chem. Inf. Model. 54 (1) (2014) 131–137.
[23] HDF5 User's guide, 2000–2013. http://www.hdfgroup.org/HDF5/doc/UG.
[24] T. Preston-Werner, Semantic versioning. http://semver.org/spec/v2.0.0.html.
[25] H. Bekker, Unification of box shapes in molecular simulations, J. Comput. Chem. 18 (15) (1997) 1930–1942. http://dx.doi.org/10.1002/(SICI)1096-987X(19971130)18:15<1930::AID-JCC8>3.0.CO;2-P.
[26] F. Höfling, Highly accelerated large-scale molecular dynamics package, 2008–2013. http://halmd.org.
[27] P.H. Colberg, F. Höfling, Highly accelerated simulations of glassy dynamics using GPUs: caveats on limited floating-point precision, Comput. Phys. Comm. 182 (5) (2011) 1120–1129. http://dx.doi.org/10.1016/j.cpc.2011.01.009.
[28] F. Höfling, S. Dietrich, Enhanced surface tension of liquid–vapour interfaces at mesoscopic scales, preprint, 2013.
[29] P. de Buyl, f90h5md—a Fortran 90 implementation of H5MD. https://github.com/pdebuyl/f90h5md.
[30] A. Collette, HDF5 for Python, 2008. http://h5py.alfven.org/.
[31] P. de Buyl, pyh5md: Read and write H5MD files. https://github.com/pdebuyl/pyh5md.
[32] The international system of units (SI), Tech. rep., Paris, 2006. http://www.bipm.org/en/si/si_brochure/general.html.