Bioinformatics, 36(1), 2020, 311–316
doi: 10.1093/bioinformatics/btz540
Advance Access Publication Date: 10 July 2019
Original Paper

OXFORD

Data and text mining

Cooler: scalable storage for Hi-C data and other genomically labeled arrays

Nezar Abdennur [1,*] **and Leonid A. Mirny** [1,2,*]

[1]Institute for Medical Engineering and Science and [2]Department of Physics, Massachusetts Institute of Technology, Cambridge, MA 02139, USA

*To whom correspondence should be addressed.

Associate Editor: Jonathan Wren

Received on March 11, 2019; revised on May 27, 2019; editorial decision on June 30, 2019; accepted on July 9, 2019

Abstract

Motivation: Most existing coverage-based (epi)genomic datasets are one-dimensional, but newer technologies probing interactions (physical, genetic, etc.) produce quantitative maps with two-dimensional genomic coordinate systems. Storage and computational costs mount sharply with data resolution when such maps are stored in dense form. Hence, there is a pressing need to develop data storage strategies that handle the full range of useful resolutions in multidimensional genomic datasets by taking advantage of their sparse nature, while supporting efficient compression and providing fast random access to facilitate development of scalable algorithms for data analysis.

Results: We developed a file format called cooler, based on a sparse data model, that can support genomically labeled matrices at any resolution. It has the flexibility to accommodate various descriptions of the data axes (genomic coordinates, tracks and bin annotations), resolutions, data density patterns and metadata. Cooler is based on HDF5 and is supported by a Python library and command line suite to create, read, inspect and manipulate cooler data collections. The format has been adopted as a standard by the NIH 4D Nucleome Consortium.

Availability and implementation: Cooler is cross-platform, BSD-licensed and can be installed from the Python package index or the bioconda repository. The source code is maintained on Github at `https://github.com/mirnylab/cooler`.

Contact: nezar@mit.edu or leonid@mit.edu

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

1 Introduction

Recent years have seen a ramp in production of large datasets that map associations between genomic loci. Of note are high-throughput chromosome conformation capture (3C) technologies (Dekker *et al.*, 2002; Denker and De Laat, 2016), such as Hi-C (Lieberman-Aiden *et al.*, 2009) and its variants, which produce two dimensional maps of chromosomal contacts. These technologies have undergone incremental improvements in technological resolution (cutting frequency, capture radius), biological sampling (cell numbers, library complexity) and technical sampling (sequencing depth), making it possible to resolve features at increasingly finer scales. Hi-C and related experiments also span a growing range of experimental scales, e.g. from single cells to large cell populations, unbiased versus specific enrichment methods; for a review, see Davies *et al.* (2017). As a result, there is a need for data structures that are flexible enough to accommodate data of massive size and varying degrees and patterns of sparsity, and easily adapt to new experimental techniques and novel metadata.

In the case of 3C-based experiments, pairs of sequence tags identify chimeric ligation junctions between DNA fragments. It is natural to subject these paired tags to *binning*, either by assigning them to putative restriction fragments or more commonly, by aggregating them with respect to genomic intervals of some fixed size. Such gridded binning also suppresses count noise and increases effective

coverage (Lajoie *et al.*, 2015). The result is a quantitative genomic *matrix*, whose dimensional axes comprise a series of fixed or variable-length genomic intervals.

Today, processed Hi-C data and similar two-dimensional datasets are often still persisted using flat text files. For large and high-resolution datasets, this poses bottleneck challenges for basic processing, analysis and visualization. There exist compression and indexing strategies for tabular text files that mitigate these challenges to some degree by enabling random access (Li, 2011). However, binary formats can provide more efficient and compressible storage, faster I/O, and preserve numerical precision. Several custom binary formats have been developed for Hi-C data, including butlr (Wang *et al.*, 2018), hic (Durand *et al.*, 2016) and MRH (Sauria *et al.*, 2015). They are useful in that they organize the data more efficiently and permit random access, but their strict byte layouts make them rather inflexible for accommodating different data types, metadata or additional information.

A popular alternative is the HDF5 container format (Koziol and Robinson, 2018), which provides the freedom to organize collections of editable multidimensional array data and metadata in binary form in a hierarchy similar to that of a file system. HDF5 is referred to as 'self-describing' because objects can be inspected for their storage metadata, such as type, compression and array shape. This enables it to serve as a flexible container for specific applications without constraining users to a strict preordained data organization. These features and its performance have made HDF5 very popular for storing large scientific datasets, and it has been made its mark in the Hi-C field for some time, in software packages such as hiclib (Imakaev *et al.*, 2012), hifive (Sauria *et al.*, 2015), gcMapExplorer (Kumar *et al.*, 2017), HiCExplorer (Wolff *et al.*, 2018) and cworld (`https://github.com/dekkerlab/cworld-dekker`).

All the HDF5-based Hi-C formats in the tools mentioned, with the exception of HiCExplorer, use dense representations, i.e. full two-dimensional arrays of counts or transformed counts—including zeros for unobserved interactions; however, this strategy scales poorly with finer binning, whereby both size and sparsity of the data increase (Supplementary Tables S1 and S2). For example, if we obtain as many as one billion contacts from a population of cells mapped to the human reference genome and bin them at kilobase resolution, we are guaranteed to fill less than 0.03% of the trillions of available matrix elements (Supplementary Data). Moreover, DNA contact frequency also exhibits a characteristic density pattern whereby contacts are much more densely sampled near the diagonal in *cis*. Sparse representations are not only critical for scalable storage: algorithms such as matrix balancing and principal component analysis can be adapted to operate using only non-zero elements, and very finely binned maps can be used to look at patterns averaged over many genomic loci. Storage that accommodates a wide range of data resolutions is also necessary to visually explore the full depth of scale of such large datasets.

Here we present a data model, an implementation and a support library for a sparse, scalable HDF5-based genomic array format called cooler. We first describe a general sparse data model for genomically labeled arrays, designed with Hi-C data in mind, but flexible enough to accommodate any binned genomic data describing associations, correlations, or interactions, such as linkage disequilibrium statistics. A very similar model is already employed in the text format of the popular HiCPro pipeline (Servant *et al.*, 2015). We then present the implementation of the model as a file format in HDF5 that can support genomic matrices at any resolution as well as multi-resolution files, with a support software library in Python, also called cooler. The library provides the functionality to create, aggregate and manipulate the contents of cooler files, provides an application programming interface (API) to materialize genomic range queries in both tabular and array forms. Its design supports both sequential and random access, ideal for the development of out-of-core data processing algorithms. A command line interface (CLI) is shipped with the cooler package for convenient scripting, application and pipeline integration.

The cooler format combines the strengths of a sparse data model with the optimized storage and querying features of HDF5. Furthermore, we provide a full high-level specification of cooler files as implemented in HDF5 (the cooler schema) to facilitate its adoption as an interoperable standard for Hi-C analysis tools.

2 Materials and methods

We outline a simple but flexible data model for representing multidimensional binned genomic data termed genomically labeled sparse arrays (GLSAs). In the context of data structures, matrices and tensors are normally referred to as *arrays*, their individual shape dimensions are sometimes also termed *axes*. We will refer to the discrete units along the array axes as *bins*. Throughout we assume that the array axes are *homogeneous*, i.e. correspond to the same genome assembly, binned in the same way. A two-dimensional genomically labeled array is a data structure that assigns unique quantitative values to pairs of bins obtained from an interval partition of a reference genome assembly. These pairs of bins make up the *coordinates* of the array's elements. Because they are often visualized as heatmaps, we also use the term 'pixels' to refer to elements of 2D arrays. By omitting pixels possessing zero or no value, the representation becomes sparse.

A genomically labeled array of dimension two or greater can be represented with a single table similar to the BEDPE format, where each non-zero array element is described by a record listing the genomic coordinates of its bins and additional bin-related attributes alongside the element's quantitative value(s) (Fig. 1b, right). In Hi-C, for example, additional bin-related quantities include normalization weights or A/B compartment signal. However, at the scale of Hi-C data, this representation is limited by the fact that bin-related attributes (coordinates, weights, etc.) can be duplicated many times throughout the table, in numbers greatly exceeding the total number of bins (Supplementary Table S2).

We eliminate such redundancy by replacing the single table with two separate ones (Fig. 1a). The first is a *bin table* that describes the complete genomic bin segmentation on both axes of the matrix, such that each bin-related attribute is fully described by a column, or one-dimensional array. The second table contains distinct axis columns (`bin1_id, bin2_id`) that reference the rows of the bin table: the resulting *pixel table* is a condensed representation of the nonzero elements of the array. Conveniently, this corresponds to the classic coordinate list (COO) representation for sparse arrays (Saad, 1990). For completeness, we include a third *chromosome table* to list the chromosomes (or other scaffolds) of the genome assembly, their genomic lengths and any other chromosome-specific attributes. Note that while the bin and chromosome tables tend to have a relatively small memory footprint, it is the pixel table that can greatly exceed memory storage for large datasets.

Furthermore, for symmetric matrices, such as Hi-C, we reduce data requirements and preserve a unique representation by keeping only unique pixels and orienting them to lie within the upper triangle of the matrix, discarding their lower triangular transpose elements (Fig. 1c). Altogether, for a given ordering of the chromosomes

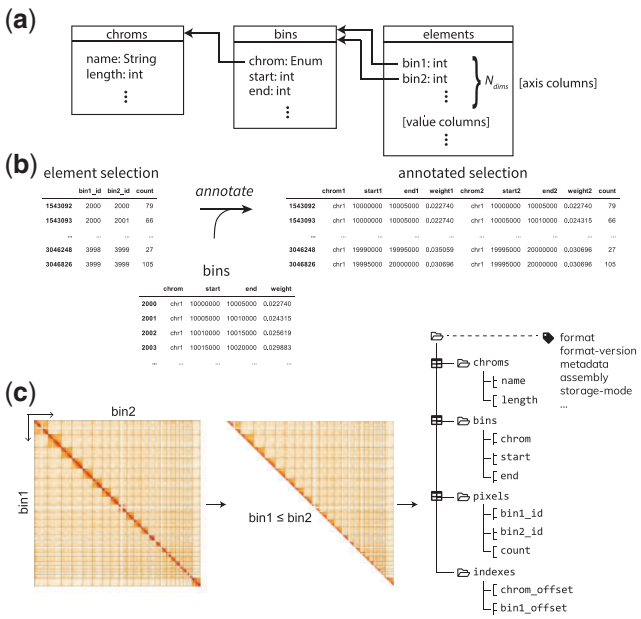

Fig. 1. Data model for GLSAs and cooler format. (**a**) Diagram of the GLSA data model. A multidimensional genomically labeled array can be represented via a decomposition that distinguishes the attributes describing the genomic intervals (table labeled *bins*) that make up the coordinates of the array's axes from the actual non-zero elements of the array (table labeled *elements*). The element table contains one or more numerical *value columns* and simple integer coordinates that reference rows of the bin table (depicted using arrows). The bin table's records describe a sequence of ordered, non-overlapping genomic intervals, minimally described by the reference sequence (*chrom*), and *start*, and *end* positions. The *chrom* column is further encoded as an integer enumeration to reference a third table labeled *chroms*, which contains attributes describing the reference sequences themselves, such as their genomic lengths. (**b**) Any selection of rows of the element table can be annotated by joining with the appropriate columns of the bin table. (**c**) For symmetric matrices, such as Hi-C maps, only upper triangular pixels are stored to eliminate duplication. Right, a diagram of a cooler data collection's hierarchical structure. The three tables are modeled as HDF5 groups (depicted as folders) while the table columns are stored as 1D arrays, which are chunked and compressed internally by HDF5. A reserved set of metadata HDF5 attributes are associated with the root group of the data collection, including a property indicating whether the matrix is to be interpreted as symmetric

of an assembly, this decomposition uniquely represents a given genomically labeled 2D array.

Given any collection of rows from the pixel table, its full representation (or any subset of bin-related attributes) can be recovered inexpensively at the application level by performing relational joins between the axis columns of the pixel table and the desired bin attributes in the bin table. We refer to this as element *annotation* (Fig. 1b).

3 Cooler format

We implemented the GLSA data model for 2D arrays using HDF5, defining a file format called cooler, with the recommended extension `.cool`. A full specification can be found in the online documentation. We outline the key elements in this section.

3.1 Specification

HDF5 files are organized hierarchically, akin to a file system within a file, with two primary structures, *groups* and *datasets*. Groups serve the role of directories and contain zero or more groups or datasets. Datasets are multidimensional arrays, which can be flexibly sized, chunked and passed through various I/O filters, including checksumming and compression. Both groups and datasets can be assigned key-value user metadata, called *attributes*, and like POSIX file systems can be referenced with relative or absolute paths separated by slashes.

HDF5 does not natively support sparse arrays or relational data structures: its datasets are dense multidimensional arrays. Therefore, we model a table as a group of equal-length 1D arrays representing columns. Although HDF5 does support row-oriented storage using compound data types (i.e. structured arrays), we chose to implement column-oriented storage because of the advantages it provides, including cheap addition or removal of columns, efficient slicing along columns and more efficient compression (Abadi *et al.*, 2008). Our simple table model thus allows for easy addition of columns and appending of rows, but not random insertion of rows. This was deemed reasonable since the raw datasets are normally write-once, but global data transformations are not uncommon. Moreover, it does not enforce a specific ordering on the columns, though a conventional order for required columns is provided in the specification.

A schematic of the array hierarchy representing a single matrix is shown in Figure 1c. Three HDF5 groups representing the *chrom*, *bin* and *pixel* tables live directly beneath the collection's root group. Our specification permits any number of additional columns in each of the three tables. For example, continuous 1D signal tracks that align with the genomic bins (e.g. normalization weights, eigenvector scores) can be appended as columns to the bin table, and pixel tables can contain multiple value columns (e.g. the color bands of an image). An additional group called *indexes* lives alongside the three tables and contains two index arrays described below. Any additional group hierarchies and metadata are also permitted as long as they are not nested within the group of a table.

We term the complete object hierarchy representing a matrix a *data collection*. The specification requires that some standard metadata be provided in the attributes of the data collection's root group (Fig. 1c). We reserve an additional attribute slot for storing serialized custom user metadata in JSON format: for example, experimental details, processing logs or mapping statistics. Since all versions of the HDF5 library ship with gzip compression, for maximum portability, it was chosen as the default compression filter for all columns.

3.2 Indexing

We further stipulate that the records of the pixel table must be sorted lexicographically by the bin ID along the first axis, then by the bin ID along the second axis. This way, the *bin1_id* column can be substituted with an array of offsets that serves as a lookup index for the rows of the matrix, stored under *indexes/bin1_offset* (Fig. 1c). With this index, we obtain a compressed sparse row (CSR) sparse matrix representation (Saad, 1990), where the *bin1_offset* index corresponds what is often named `indptr` in CSR data structures. Given an enumeration of chromosomes, the bin table must also be lexicographically sorted by chromosome then by start coordinate. Then similarly, the *chrom* column of the bin table will reference the rows of the chrom table, and can also be substituted with an offset array, stored under *indexes/chrom_offset* (Fig. 1c). Because of the efficient compression of sorted columns, we preserve the original *bin1_id* and *chrom* columns so that the indexes may be

dispensable for a reader that does not wish to use them. Cooler can therefore be thought of as a hybrid COO-CSR format.

3.3 Flavors

Although the standard cooler file contains a single data collection located at the file's root group, we allow a cooler file to store multiple data collections anywhere in the group hierarchy of an HDF5 file, as long as they are not sequentially nested. Any number of data collections is permitted, and each individual data collection can be referenced using its qualified group path. We therefore support a flexible syntax to locate data collections within a file (see below) and provide two conventions or 'flavors' of the file layout: the single resolution and multi-resolution or 'zoomified' cooler (often suffixed `.mcool`), which is ideal for interactive multiscale visualization, as exemplified by HiGlass (Kerpedjiev *et al.*, 2018) (See Supplementary Fig. S1 and `https://higlass.io/app?config=W4DNgqjXRNWPQ7Nbz7NLnQ`).

4 Cooler package

We provide a Python-based convenience library to manage cooler data collections. It provides tools to create and append data to collections, to merge, aggregate and normalize them, and to and query their contents and metadata. To identify cooler data collections that may lie at any level of the group hierarchy of a file, we support a resource path syntax consisting of a file path, optionally followed by double colon :: followed by a fully qualified HDF5 group path. If the double colon and group path are omitted from the resource path, the data collection is interpreted as being located in the file's root group.

4.1 CLI

The cooler Python package ships with a CLI for most common manipulations (Fig. 2a). They are provided as subcommands under a top-level `cooler` command namespace. We briefly describe the main commands below, grouped by theme. All these actions are also available and further customizable programmatically through the Python API.

4.1.1 Ingestion

There are several commands used to create cooler data collections from input data. The first required input is a definition of the bin segmentation of the genome assembly to which the interaction data was mapped. In the case of a uniform bin size, the user may simply provide a file listing the chromosome lengths (a chrom-sizes file) along with the bin size (resolution) in base pairs. In the case of variable length bins, it can be specified using a BED file. The second required input is either (1) a list of records describing aggregated data (i.e. a matrix) or (2) a list of unaggregated paired tags.

If the input data are unaggregated paired tags, the command `cooler cload` will aggregate the pairs according to the provided bin segmentation. If the input is already binned, the `cooler load` command will convert the input matrix into the cooler format.

4.1.2 Aggregation

Cooler data collections at one or more base resolutions can be aggregated or *coarsened* to lower resolutions using the `cooler coarsen` command. Moreover, recursively aggregated multi-resolution cooler files can be generated using the `cooler zoomify` command. The resulting files are suitable both for data analysis and for multiscale visualization (Waldispühl *et al.*, 2018).

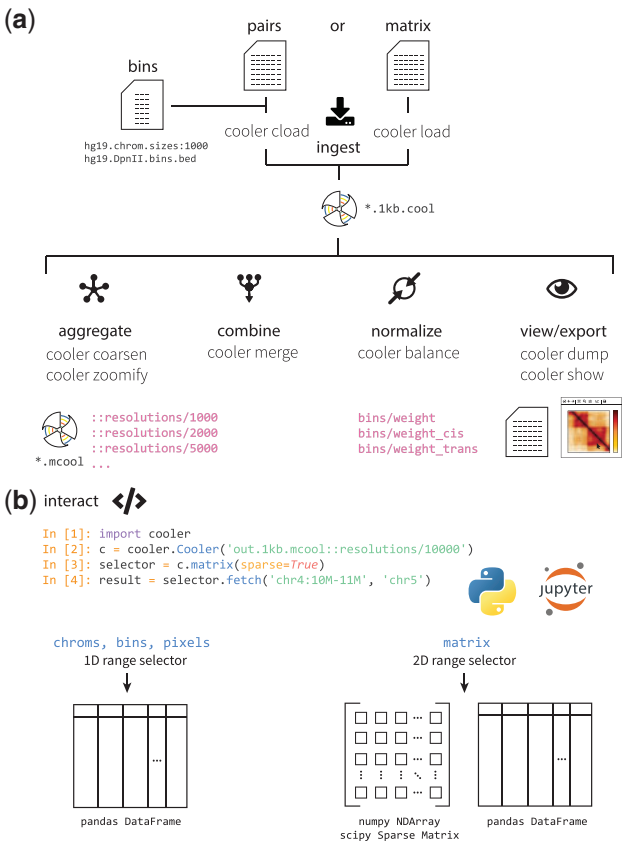

Fig. 2. Cooler CLI and Python library. (**a**) Summary of the main categories of `cooler` commands available with the cooler Python package, illustrating the flow of data. The main operations include the ingestion of file or text streams to create new coolers, aggregation and coarsening of existing coolers to lower resolutions, merging of axis-compatible matrices, normalization of cooler matrices by iterative correction, utilities to serialize and stream out the data and metadata inside a cooler file and to process range queries and a lightweight viewer to visually inspect a matrix. For example, one uses either the `load` command to ingest pre-aggregated data already in matrix form or the `cload` command to aggregate paired tag records into a matrix. The genomic bin segmentation defining the axes of the matrix must be provided separately by providing either a path to a BED file or a path to a chromosome sizes file along with a specified fixed bin size. (**b**) The cooler Python library provides a `Cooler` class that exposes data *range selectors* to facilitate data retrieval and analysis. The individual *chrom*, *bin* and *pixel* tables are accessible using 1D range selectors that accept column and row-range selections and yield pandas data frame output. A cooler's matrix values are also exposed using a 2D range selector that processes rectangular range queries specified either by a pair of genomic coordinate intervals in UCSC-style notation (using the `fetch` method) or as integer matrix coordinates (using Python slice syntax). The retrieved 2D range data may be materialized as dense NumPy arrays, sparse matrices or data frames. For symmetric coolers, the file's upper triangular data will be appropriately mirrored in the array and sparse matrix outputs

4.1.3 Merging

Any combination of cooler data collections with compatible bin axes, such as technical or biological replicates of an experiment, can be pooled together in a memory-efficient manner using the `cooler merge` command.

4.1.4 Balancing

The *de facto* standard method for normalizing Hi-C data is matrix balancing, also known as iterative correction (Imakaev *et al.*, 2012).

Because is it such a common transformation, we include this functionality in the cooler package. The output is a vector of balancing weights [the reciprocal of the biases described in Imakaev *et al.* (2012)]. These weights are applied by multiplying each pixel value by the weights associated with its two genomic bins. The `cooler balance` command performs bin-level filtering and balancing using a parallel and out-of-core implementation of iterative correction with extensive options. Balancing weights are commonly stored as columns in the bin table of a data collection and applied on-the-fly during querying. See Supplementary Data for further discussion.

4.1.5 View/export

The contents of the *chrom*, *bin* and *pixel* tables may be serialized as delimited text using the `cooler dump` command, which also supports genomic range queries and pixel annotation. Additionally, the `cooler show` command provides a lightweight interactive matplotlib visualization to inspect and explore the data.

4.2 Python library

The cooler library provides programmatic access to all of the above functions as well as an API to select arbitrary ranges of data from the tables and perform 2D range queries on matrices (Fig. 2b), powered by the *h5py* (Collette, 2013) Python interface to HDF5. It provides both tabular and 2 D array-based interfaces to the sparse (and, for symmetric matrices, upper triangular) data representation in the file. It understands both global array indices and genomic coordinate ranges.

To integrate seamlessly into the Python data ecosystem, the cooler package's API materializes queries in several common scientific data structures, including dense *NumPy* arrays, *SciPy* sparse matrices and *Pandas* data frames. Furthermore, selections of pixels in data frame form can be annotated with genomic bin columns as depicted in Figure 1b using the *annotate* function.

The combined CLI and API of the cooler package makes it suitable for use in scripts and pipelines and for inclusion in other software libraries. Thanks to its ease of integration with Python-backed tools, it has already been successfully integrated into 3D genome browsers, pipelines and visualization tools, including HiGlass (Kerpedjiev *et al.*, 2018), the WashU Epigenome Browser (Li *et al.*, 2017), HiCExplorer (Wolff *et al.*, 2018), HiCPlotter (Akdemir and Chin, 2015) and CoolBox (Xu *et al.*, 2019). The cooler package also provides the flexibility to facilitate interactive data analysis in environments such as the Jupyter Notebook platform (Kluyver *et al.*, 2016). A comprehensive suite of tools dedicated to Hi-C data analysis using cooler files is being developed as part of a package called *cooltools* and will be presented elsewhere.

4.3 Implementation

Cooler is implemented as a Python package and supports Python versions 2.7 and 3.4 or greater and works on Linux, Mac OS X and Windows platforms. The cooler package is open source, BSD licensed, and the source code is maintained on GitHub at `https://github.com/mirnylab/cooler`. The documentation is hosted at `https://cooler.readthedocs.io`.

Cooler can be installed using Python's pip package manager,

```
$ pip install cooler
```

or from the bioconda distribution channel (Grüning *et al.*, 2018) using the conda package manager

```
$ conda install -c bioconda cooler
```

For docker users, the CLI is also available through BioContainers (da Veiga Leprevost *et al.*, 2017)

```
$ docker run quay.io/biocontainers/cooler
cooler -help
```

5 Discussion

We present a sparse data model and file format for genomically labelled arrays with minimal redundancy but enough flexibility to support a wide range of data types, data sizes and future metadata requirements. A sparse representation in particular is crucial for developing robust tools and algorithms for use on increasingly high-resolution multidimensional genomic data sets that need to operate on subsets of data at a time. While we developed a command line suite and Python API to create, inspect and manipulate cooler data collections, we also set out to define a simple enough specification that it could be easily interpreted across programming environments using the established APIs of the underlying storage layer.

In selecting a storage layer on which to implement our sparse array data model, HDF5 was chosen because it is an open-source, portable and performant format for scientific data with widespread use and whose self-describing generic data structures are well suited to data modeling. Furthermore, we required exchangeable encapsulated files, rather than sharded files or distributed databases, because the former are still indispensable for modern bioinformatic workflows and data analysis practices. Popular binary formats such as the current versions of MATLAB's mat files and Unidata's NetCDF4 are built as abstractions on top of HDF5 (Dougherty *et al.*, 2009) and it has been used to store petabytes of mission critical data, such as NASA's Earth Observation System, for decades (Folk *et al.*, 2011). Importantly, HDF5 supports chunking and compression, arrays of unlimited size and efficient array subset selection (slicing). High level APIs exist for a wide variety of programming languages, including C/C++, Java, Python, Perl and R, facilitating interoperability across tools developed in different programming languages and environments.

Nevertheless, the high-level GLSA data model can be implemented using a variety of storage strategies for different goals. For example, because the model minimally fragments the data while eliminating duplication, it can be very useful for text-based interchange for multidimensional genomic arrays or parts thereof. Indeed, for Hi-C data, a two-file text format based on a bin file and upper triangular element file was introduced by the HiCPro pipeline (Servant *et al.*, 2015). Other open source backends in which GLSAs could be implemented include Apache Parquet (`https://parquet.apache.org/`), a cloud-optimized columnar binary format for big tabular data; an emerging array storage technology called TileDB that provides native high performance sparse array support (`https://tiledb.io/`) and a new format called Zarr (`https://github.com/zarr-developers/zarr`) that provides an HDF5-inspired implementation of chunked, compressed, N-dimensional arrays that can work on top of a variety of storage layers (file system hierarchies, zip files, or key-values stores). Such new technologies are poised to become more important as community genomic data migrate increasingly to cloud storage and computing environments.

Of course, our design decisions involved certain tradeoffs. The data model assigns every genomic bin a global bin ID, which is sensitive to the chromosome order chosen. This tradeoff was deemed preferable to more complicated schemes that add additional bin identifiers or that divide each scaffold-pair block into a separate group. HDF5

supports a row-based storage model for tables using compound data types (structured arrays), but the compression, append and performance benefits of columnar were deemed important enough to define a column-based storage model. The CSR indexing scheme in our HDF5 schema is very space-efficient, but not optimal for 2D range queries because the data are not serialized in a way that strongly preserves 2D locality. However, by using different sort orders on the element data, the data model we present can support more sophisticated indexing schemes, such as space-filling curves (Pascucci and Frank, 2003). Finally, the data model implemented describes two-dimensional arrays with homogeneous sets of axes. Though not yet implemented in the cooler package, handling heterogeneous axes is a matter of including separate chromosome and bin tables for each distinct axis. The data model also extends naturally to multidimensional tensors by including additional axis columns in the element table.

Cooler provides a scalable solution to tackling the analysis and visualization of big Hi-C data and other genomically labeled matrices at any data resolution, scaling to massive data sizes without incurring the read and write bottlenecks of dense storage. It is also amenable to external-memory (often called out-of-core) algorithms that can be controlled to maximally exploit multiple cores and disk I/O while not overwhelming memory resources (Vitter, 2006). The cooler package's command line tools facilitate integration into scripts and workflows, and its Python API allows users to leverage the powerful resources of the Python data ecosystem while the portable HDF5 format makes it readily accessible in other environments such as Java and R. The cooler format has adopted as a standard for Hi-C data storage, along with hic (Durand *et al.*, 2016) and the pairs text format (`https://github.com/4dn-dcic/pairix/blob/master/pairs_format_specification.md`), by the 4D Nucleome Consortium's Data Coordination and Integration Center, and is already supported by a number of genomic analysis and visualization tools, including HiGlass (Kerpedjiev *et al.*, 2018).

Acknowledgements

We greatly thank Anton Goloborodko and Maxim Imakaev for many extensive discussions, contributions to code development and providing feedback on the manuscript. We are grateful to Peter Kerpedjiev and Fritz Lekschas for helpful discussions and for HiGlass integration. We also thank Geoffrey Fudenberg, Sergey Venev, Ilya Flyamer, Joachim Wolff, members of the Mirny lab, and members of Nils Gehlenborg's and Peter Park's groups for feedback, contributions and animated participation on the public issue tracker.

Funding

This work was supported by the National Institutes of Health Common Fund 4D Nucleome Program [Center for Structure and Physics of the Genome DK107980; and 4D Nucleome Network Data Coordination and Integration Center U01 CA200059].

Conflict of Interest: none declared.

References

Abadi,D.J. *et al.* (2008) Column-stores vs. row-stores: how different are they really? In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, SIGMOD '08*, pp. 967–980. ACM, New York, NY, USA.

Akdemir,K.C. and Chin,L. (2015) HiCPlotter integrates genomic data with interaction matrices. *Genome Biol.*, **16**, 198.

Collette,A. (2013) *Python and HDF5: Unlocking Scientific Data*. O'Reilly.

da Veiga Leprevost,F. *et al.* (2017) BioContainers: an open-source and community-driven framework for software standardization. *Bioinformatics*, **33**, 2580–2582.

Davies,J.O. *et al.* (2017) How best to identify chromosomal interactions: a comparison of approaches. *Nat. Methods*, **14**, 125.

Dekker,J. *et al.* (2002) Capturing chromosome conformation. *Science*, **295**, 1306–1311.

Denker,A. and De Laat,W. (2016) The second decade of 3C technologies: detailed insights into nuclear organization. *Genes Dev.*, **30**, 1357–1382.

Dougherty,M.T. *et al.* (2009) Unifying biological image formats with HDF5. *Queue*, **7**, 20.

Durand,N.C. *et al.* (2016) Juicebox provides a visualization system for hi-c contact maps with unlimited zoom. *Cell Syst.*, **3**, 99–101.

Folk,M. *et al.* (2011) An overview of the HDF5 technology suite and its applications. In: *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, pp. 36–47. ACM, Uppsala, Sweden.

Grüning,B. *et al.* (2018) Bioconda: sustainable and comprehensive software distribution for the life sciences. *Nat. Methods*, **15**, 475.

Imakaev,M. *et al.* (2012) Iterative correction of Hi-C data reveals hallmarks of chromosome organization. *Nat. Methods*, **9**, 999.

Kerpedjiev,P. *et al.* (2018) HiGlass: web-based visual exploration and analysis of genome interaction maps. *Genome Biol.*, **19**, 125.

Kluyver,T. *et al.* (2016) Jupyter notebooks—a publishing format for reproducible computational workflows. In: Loizides,F. and Schmidt,B. (eds) *Positioning and Power in Academic Publishing: Players, Agents and Agendas, Göttingen, Germany*. IOS Press, Amsterdam, Netherlands, pp. 87–90.

Koziol,Q. and Robinson,D. (2018) The HDF Group (2000–2018) Hierarchical Data Format version 5. doi: 10.11578/dc.20180330.1.

Kumar,R. *et al.* (2017) Genome contact map explorer: a platform for the comparison, interactive visualization and analysis of genome contact maps. *Nucleic Acids Res.*, **45**, e152.

Lajoie,B.R. *et al.* (2015) The Hitchhiker's guide to Hi-C analysis: practical guidelines. *Methods*, **72**, 65–75.

Li,D. *et al.* (2017) Chromatin interaction data visualization in the WashU epigenome browser. *bioRxiv*, p. 239368.

Li,H. (2011) Tabix: fast retrieval of sequence features from generic tab-delimited files. *Bioinformatics*, **27**, 718–719.

Lieberman-Aiden,E. *et al.* (2009) Comprehensive mapping of long-range interactions reveals folding principles of the human genome. *Science*, **326**, 289–293.

Pascucci,V. and Frank,R.J. (2003) Hierarchical indexing for out-of-core access to multi-resolution data. In: Gerald,F. *et al.* (eds) *Hierarchical and Geometrical Methods in Scientific Visualization*. Springer-Verlag, Berlin Heidelberg, pp. 225–241.

Saad,Y. (1990) SPARSKIT: A basic tool kit for sparse matrix computations. *Technical Report* 90.20. Research Institute for Advanced Computer Science, NASA Ames Research Center.

Sauria,M.E. *et al.* (2015) HiFive: a tool suite for easy and efficient HiC and 5C data analysis. *Genome Biol.*, **16**, 237.

Servant,N. *et al.* (2015) HiC-Pro: an optimized and flexible pipeline for Hi-C data processing. *Genome Biol.*, **16**, 259.

Vitter,J.S. (2006) Algorithms and data structures for external memory. *Found. Trends Theor. Comput. Sci.*, **2**, 305–474.

Waldispühl,J. *et al.* (2018) Storage, visualization, and navigation of 3D genomics data. *Methods*, **142**, 74–80.

Wang,Y. *et al.* (2018) The 3D genome browser: a web-based browser for visualizing 3D genome organization and long-range chromatin interactions. *Genome Biol.*, **19**, 151.

Wolff,J. *et al.* (2018) Galaxy HiCExplorer: a web server for reproducible Hi-C data analysis, quality control and visualization. *Nucleic Acids Res.*, **46**, W11.

Xu,W. *et al.* (2019) Coolbox: a interactive genomic data explorer for Jupyter notebook. *bioRxiv*.