BIOINFORMATICS APPLICATIONS NOTE Vol. 26 no. 11 2010, pages 1458–1459
doi:10.1093/bioinformatics/btq164

Genome analysis

The Genomedata format for storing large-scale functional genomics data

Michael M. Hoffman[1], Orion J. Buske[1] and William Stafford Noble[1,2,*]

[1]Department of Genome Sciences, University of Washington, PO Box 355065, Seattle, WA 98195-5065 and [2]Department of Computer Science and Engineering, University of Washington, PO Box 352350, Seattle, WA 98195-2350, USA

Associate Editor: John Quackenbush

ABSTRACT

Summary: We present a format for efficient storage of multiple tracks of numeric data anchored to a genome. The format allows fast random access to hundreds of gigabytes of data, while retaining a small disk space footprint. We have also developed utilities to load data into this format. We show that retrieving data from this format is more than 2900 times faster than a naive approach using wiggle files.

Availability and Implementation: Reference implementation in Python and C components available at http://noble.gs.washington.edu/proj/genomedata/ under the GNU General Public License.

Contact: william-noble@uw.edu

Received on December 17, 2009; revised on March 25, 2010; accepted on April 9, 2010

1 INTRODUCTION

The advent of functional genomics assays based on next-generation sequencing (Brunner *et al.*, 2009; Hesselberth *et al.*, 2009; Park, 2009; Wold and Myers, 2008) finally allows the high-throughput acquisition of data at 1-bp resolution across entire genomes. Processing this information, however, provides a challenge for several orders of magnitude beyond that of previous genomic analyses and demands new techniques for efficient operation. We introduce the Genomedata format for genome-scale numerical data, which uses an HDF5 (Hierarchical Data Format; http://hdfgroup.org/HDF5/) container for efficient, random access to huge genomic datasets. We also provide a Python interface to this format.

Traditional data interchange formats such as the wiggle (http://genome.ucsc.edu/goldenPath/help/wiggle.html) and bedGraph (http://genome.ucsc.edu/goldenPath/help/bedgraph.html) formats provide excellent means of disseminating genome-wide datasets but suffer from several disadvantages in the repeated processing of this data. Storing numerical data as ASCII text is inefficient and impedes random access to the data. This problem becomes even more apparent when processing the data in scripting languages such as Python and R, which provide high-performance methods for bulk numerical operations on arrays, but no method for reading in data in interchange formats quickly. It is also necessary to validate this data before use, checking that there is exactly one data point per position and that data are not defined outside the boundaries of the underlying sequence. Genomedata provides an intermediate format and off-loads the frustrations of parsing and validating the data from an analysis programmer. It provides the conveniences of an application programming interface for reading a binary file format, akin to the programmatic access to sequence and alignment data provided by BAM (Li *et al.*, 2009) and BioHDF (Mason *et al.*, 2010), while being suited for dense numeric data such as bigWig (Rhead *et al.*, 2010).

In many workflows, Genomedata allows the user to parse, validate and convert the data into a binary format once, eliminating the computational expense of doing this repeatedly. The data are stored as 32-bit IEEE floating point numbers to allow minimal processing when loading into memory. Not a number entries are used where data are missing or unassigned. HDF5 transparently breaks the data into chunks aligned with data columns, so that it minimizes work during loading. Genomedata compresses these chunks when stored on disk to save space, especially when values are repeated within a column, but in a way that still facilitates efficient random access. We also store some metadata in the archive such that simple summary statistics may be accessed quickly.

To ease the memory requirements of subsequent analysis, Genomedata may optionally break chromosomes into 'supercontigs,' which avoid the allocation of empty space in the observation matrix at large assembly gaps (by default, >100 000 bp). This is not necessary for efficient performance on disk, but it is convenient for programmers who wish to process the whole genome.

The reference implementation includes several programs for loading data. The software requires Python 2.5.1, HDF5 1.8 and PyTables 2.1.

2 USING GENOMEDATA

Genomedata supplies command-line utilities that make it easy to create archives and load data. The `genomedata-load` command loads the genome sequence and a number of tracks in wiggle, BED or bedGraph formats, and stores metadata that allow one to rapidly calculate summary statistics such as minimum, maximum, mean or SD. The package also contains utilities to complete only parts of the loading process so that one may load tracks for different chromosomes in parallel.

It is easy to access data in a Genomedata archive using the supplied Python interface. A programmer may retrieve a matrix of data by specifying individual coordinate ranges to the Genomedata interface. Alternatively, one can iterate through the entire dataset

*To whom correspondence should be addressed.

chromosome by chromosome. Programmers can accomplish tasks such as reporting the average data value in a number of tracks for specified genomic regions easily, allowing a greater focus on more interesting areas of analysis.

3 PERFORMANCE

Genomedata can quickly load large amounts of data. We measured the time to load a Genomedata archive with the complete human genome sequence (build NCBI36) and from one to 11 ChIP-seq data tracks on a 2.33-GHz Intel Xeon E5345 processor, and performed a linear regression on the timing results with the statistical computing environment R. This yielded a model with the coefficient of determination $R^2 = 0.98$, where loading the sequence and other constant overhead took $5.0 \pm 2.5 \times 10^3$ s, and each track took an additional $7.5 \pm 0.4 \times 10^3$ s.

One may retrieve functional genomics data from Genomedata archives much more quickly than the text-based formats commonly used for this data. We measured the time to retrieve data from a whole-genome 1-bp-resolution DNase-seq data track at each of a randomly generated list of genomic positions using a method that accessed the original gzip-compressed wiggle file and two different methods that access a Genomedata archive loaded from that file (Fig. 1). The offline (sequential access) wiggle algorithm first sorts the list and then iterates through the original wiggle files until it finds the specified positions. The offline Genomedata algorithm works in a similar way, but iterates through a Genomedata archive instead. The online (random access) Genomedata algorithm retrieves the data at each position in the random order specified by the list. We repeated this process with nine different list sizes to examine the dependence of retrieval time on the number of positions.

Because the offline algorithms read data sequentially rather than randomly, their run times are mostly independent of the number of genomic positions. After creation of the Genomedata archive, the offline Genomedata algorithm ran 2900 times faster than the comparable offline wiggle approach, suggesting a considerable advantage for the use of Genomedata when repeatedly accessing a dataset. Even when including the one-time cost of creating the archive (4 h), the Genomedata approach still ran 10 times faster, because we wrote the Genomedata track loader in C. The advantage for an online Genomedata approach is even greater when retrieving fewer than ~10 000 positions at once. Genomedata is especially suited for whole-genome, dense datasets, so it has less of a comparative advantage in cases of sparse datasets with data at only a limited number of genomic positions. Genomedata should still perform as well, however, in an absolute sense.

Not only does using Genomedata improve performance, but it also makes programming against this type of data easier, resulting in less boilerplate code for data retrieval. According to SLOCCount (http://www.dwheeler.com/sloccount/), which counts the physical source lines of code in a program, it took 70 source lines of code to implement the wiggle method, while only 44 (37% fewer) to implement the offline Genomedata method and 16 (77% fewer) to implement the online Genomedata method.

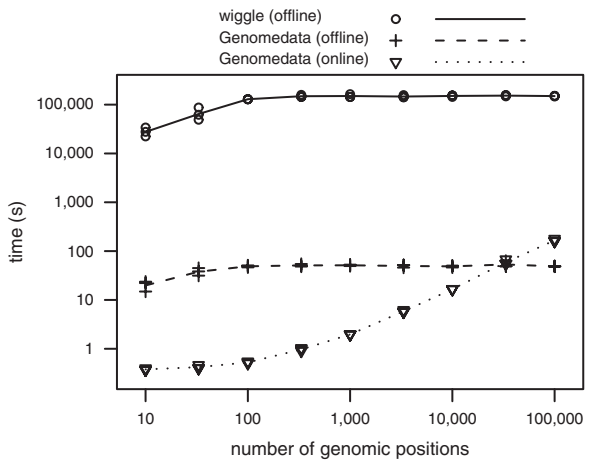

Fig. 1. Scatter plot of the time to retrieve data from a list of random genomic positions against the number of positions for different algorithms. Each point represents the average run time of the last three of four sequential trials (to eliminate caching effects) with a specific algorithm and a particular list of random positions. We used three different random lists of nine different sizes on three different algorithms, resulting in 81 plotted data points. The wiggle (circles) and offline Genomedata (crosses) algorithms ran in approximately constant time for greater than 100 positions, averaging 140 000 s (39 h) and 48 s, respectively. The online Genomedata algorithm (triangles) ran in approximately linear time for greater than 1000 random positions, averaging 1.7 ms per random access.

Funding: National Institutes of Health (HG004695).

Conflict of Interest: none declared.

REFERENCES

Brunner,A.L. *et al*. (2009)Distinct DNA methylation patterns characterize differentiated human embryonic stem cells and developing human fetal liver. *Genome Res*., **19**, 1044–1056.

Hesselberth,J.R. *et al*. (2009) Global mapping of protein-DNA interactions *in vivo* by digital genomic footprinting. *Nat. Methods*, **6**, 283–289.

Li,H. *et al*. (2009) The sequence alignment/map format and SAMtools. *Bioinformatics*, **25**, 2078–2079.

Mason,C.E. *et al*. (2010) Standardizing the Next Generation of Bioinformatics Software Development With BioHDF (HDF5).*Advances in Computational Biology*. Springer, in press.

Park,P.J. (2009) ChIP-seq: advantages and challenges of a maturing technology. *Nat. Rev. Genet*., **10**, 669–680.

Rhead,B. *et al*. (2010) The UCSC Genome Browser database: update 2010. *Nucleic Acids Res.*, **38**, D613–D619.

Wold,B. and Myers,R.M. (2008) Sequence census methods for functional genomics. *Nat. Methods*, **5**, 19–21.