BIOINFORMATICS

Vol. 00 no. 00 2015
Pages 1–2

BGT: efficient and flexible genotype query across many samples

Heng Li

Broad Institute, 7 Cambridge Center, Cambridge, MA 02142, USA

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Associate Editor: XXXXXXX

ABSTRACT

Summary: BGT is a compact format, a fast command line tool and a simple web application for efficient and convenient query of whole-genome genotypes and frequencies across tens to hundreds of thousands of samples. On real data, it encodes the haplotypes of 32,488 samples across 39.2 million SNPs into a 7.4GB database and decodes a couple of hundred million genotypes per CPU second. The high performance enables real-time responses to complex queries.

Availability and implementation: https://github.com/lh3/bgt

Contact: hengli@broadinstitute.org

1 INTRODUCTION

VCF/BCF (Danecek et al., 2011) is the primary format for storing and analyzing genotypes of multiple samples. It however has a few issues. Firstly, as a streaming format, VCF compresses all types of information together. Retrieving site annotations or the genotypes of a few samples usually requires to decode the genotypes of all samples, which is unnecessarily expensive. Secondly, VCF does not take advantage of linkage disequilibrium (LD), while using this information can dramatically improve compression ratio (Durbin, 2014). Thirdly, a VCF record is not clearly defined. Each record may consist of multiple alleles with each allele composed of multiple SNPs and INDELs. This ambiguity complicates annotations, query of alleles and integration of multiple data sets. At last, most existing VCF-based tools do not support expressive data query. We frequently need to write scripts for advanced queries, which costs both development and processing time. GQT (Layer et al., 2015) attempts to solve some of these issues. While it is very fast for selecting a subset of samples and for traversing all sites, it discards phasing, is inefficient for region query and is not compressed well. The observations of these limitations motivated us to develop BGT.

2 METHODS

2.1 Design

2.1.1 Data model BGT keeps diploid genotypes as a 2-bit integer matrix (H_{ki}) with row indexed by a pair of overlapping reference/non-reference alleles and column by a sample haplotype (thus for m' samples, the matrix has $2m'$ columns). H_{ki} takes value 0 if the i-th haplotype has the reference allele in the allele pair at row k, takes 1 if the haplotype has the non-reference allele, 2 if unknown and 3 if the haplotype has a different non-reference allele. BGT arbitrarily phases unphased genotypes and always breaks complex variants in VCF down to the smallest possible variants. It keeps the allele pairs (i.e. rows) in a site-only BCF, disallowing multiple alleles per VCF line, and stores the integer matrix as two positional BWTs (PBWTs), one for the lower bit and the other for the higher bit.

2.1.2 PBWT overview PBWT (Durbin, 2014) is a generic way to encode binary matrices. Let $(A_k)_{0\le k<n} = (A_0, \ldots, A_{n-1})$ be a list of m-long binary strings. $(A_k)_k$ can be regarded as an $n \times m$ binary matrix with $A_k[i]$ representing the cell at row k and column i. For simplicity, introduce a sentinel row $A_{-1} = \$_0\$_1 \cdots \$_{m-1}$ with a lexicographical order $\$_0 < \cdots < \$_{m-1}$.

Define binary string:

$$P_{ki} = A_k[i]A_{k-1}[i] \ldots A_0[i]A_{-1}[i]$$

to be the reverse of the column prefix ending at (k, i) and define $S_k(i)$ to be the column index of the i-th smallest prefix among set $\{P_{kj}\}_j$. $S_k(i)$ is a bijection on $\{0, \ldots, m-1\}$ and thus invertible. In a special case, $S_{-1}(i) = i$ because $P_{-1,i} = A_{-1}[i] = \$_i$.

The PBWT of $(A_k)_{0\le k<n}$ is $(B_k)_{0\le k<n}$, which is calculated by

$$B_k[i] = A_k[S_{k-1}(i)]$$

An important use of $(B_k)_k$ is to compute S_k. Define

$$\phi_k(i) = C_k(B_k[i]) + \mathrm{rank}_k(B_k[i], i)$$

where $C_k(b)$ gives the number of symbols in B_k that are lexicographically smaller than b and $\mathrm{rank}_k(b, i)$ the number of b symbols in B_k before position i. The i-th smallest column in row $k-1$ is ranked $\phi_k(i)$ in row k. Thus

$$S_k(\phi_k(i)) = S_{k-1}(i)$$

Given A_k and S_{k-1}, we can compute S_k and B_k in the order of $B_k \to \phi_k \to S_k$, starting from $k = 0$. Conversely, given B_k and S_{k-1}, computing $\phi_k \to S_k \to A_k$ derives A_k from B_k.

When there are strong correlations between adjacent rows, which is true for haplotype data due to LD, 0s and 1s tend to form long runs in B_k. This usually makes B_k much more compressible than A_k under run-length encoding. For our test data set, 32 thousand genotypes in a row can be compressed to less than 200 bytes in average.

2.2 Query with BGT

2.2.1 Flat Metadata Format (FMF) BGT introduces a new but simple text format, FMF, to manage meta data. FMF has a similar data model to wide-column stores. It is TAB-delimited with the first column showing the row name and following columns giving typed key-value pairs. An example looks like:

```
sample1   gender:Z:M   height:f:1.73   foo:i:10
sample2   gender:Z:F   height:f:1.64   bar:i:20
```

where rows can be retrieved by an arbitrary expression such as "height>1.65". BGT uses FMF to keep and query sample phenotypes and variant annotations.

2.2.2 Query genotypes The BGT query system is inspired by SQL. Notably, BGT can select variants and samples with arbitrary conditions on meta data (analogy: the WHERE clause in SQL). It has a few built-in aggregate variables such as the allele count of selected samples (analogy: the aggregate functions) and can filter output with these variables (analogy: the HAVING clause). BGT outputs VCF/BCF by default, but it may optionally output user requested fields only (analogy: fields in the SELECT statement). BGT also allows to create multiple sample groups with allele count calculated for each group and to set conditions on them (see Results), which is inspired by SQL's joining multiple instances of the same table in the FROM clause.

On the other hand, the query syntax of BGT is very different from SQL. It tends to be simpler as BGT only has a few 'tables' with fixed relationship. BGT can also count haplotypes across a set of variants, which is not easy to implement with a pure SQL backend.

2.3 BGT server

BGT comes with a standalone web server frontend implemented in the Go programming language. The server has a similar interface to the command line tool, but with additional consideration of sample anonymity. With BGT, each sample has an attribute 'minimal group size' or MGS. On query, the server refuses to create a sample group if the size of this group is smaller than the MGS of one sample in this group. In particular, if a sample has MGS larger than one, users cannot access its sample name and individual genotypes, but can retrieve allele counts computed together with other samples. This prevents users to access data at the level of a single sample.

3 RESULTS

We generated the BGT database for the first release of Haplotype Reference Consortium (HRC; http://bit.ly/HRC-org). The input is a BCF containing 32,488 samples across 39.2 million SNPs on autosomes. The BGT file size is 7.4GB, 11% of the genotype-only BCF, or 8% of GQT. Decoding the genotypes of all samples across 142k sites in a 10Mbp region takes 11 CPU seconds, which amounts to decoding 420 million genotypes per second. This speed is even faster than computing allele counts and outputting VCF.

We use the following command line to demonstrate the query syntax of BGT:

```
bgt view -G -d var.fmf.gz -a'gene=="BRCA1"' \
  -s 'source=="IBD"' -s 'source=="1000G"' \
  -f 'AC1/AN1>=0.001&&AC2/AN2>=0.001' \
 HRC-r1.bgt
```

It finds BRCA1 variants annotated in 'var.fmf.gz' that have $\geq$0.1% frequency in both the IBD data set (http://www.ibdresearch.co.uk) and 1000 Genomes (1000 Genomes Project Consortium, 2012). In this command line, `-G` disables the output of genotypes. Option `-a` selects variants with the 'gene' attribute equal to 'BRCA1' according to the variant database specified with `-d`. This condition is independent of sample genotypes. Each option `-s` selects a group of samples based on phenotypes. For the #-th sample group/`-s`, BGT counts the total number of called alleles and the number of non-reference alleles and writes them to the AN# and AC# aggregate variables, respectively. Option `-f` then use these aggregate variables to filter output.

The command line above takes 12 CPU seconds with most of time spent on reading through the variant annotation file to find matching alleles. The BGT server reads the entire file into memory to alleviate the overhead, but a better solution would be to use a proper database for variant annotations.

To demonstrate the server frontend, we have also set up a public BGT server at http://bgtdemo.herokuapp.com. It hosts 1000 Genomes haplotypes for chromosome 11 and 20.

4 DISCUSSIONS

Given a multi-sample VCF, most BGT functionalities can be achieved with small scripts, but as a command line tool, BGT has a few advantages. Firstly, it saves development time. Extracting information from multiple files can be done with a command line instead of a script. Secondly, BGT saves processing time. With high-performance C code at the core, BGT is much faster than processing VCF in a scripting language such as Perl or Python. For example, deriving allele counts in a 10Mbp region for the HRC data takes 30 seconds with BGT, but doing the same with a Perl script takes 40 minutes, a 80-fold difference. Thirdly, the design of one non-reference allele per record makes BGT merge much simpler and faster than generic VCF merge. This enables efficient query across multiple BGT databases which is not practical with VCF.

The BGT server tries to solve a bigger problem: data sharing. Instead of always delivering full data in VCF, projects could have a new option to serve data publicly with the BGT server, letting users select the summary statistics of interest on the fly while keeping samples unidentifiable. This is an improvement to Stade et al. (2014) which only provide precomputed summary.

We acknowledge that our MGS-based data sharing policy might have oversimplified real scenarios, but we believe this direction, with proper improvements and more importantly the approval of ethical review boards, will be more open, convenient, efficient and secure than our current share-everything-with-trust model.

ACKNOWLEDGEMENT

We are grateful to HRC for granting the permission to use the data for evaluating the performance of BGT and thank the Global Alliance Data Working Group for the helpful suggestions.

Funding: NHGRI U54HG003037; NIH GM100233

REFERENCES

1000 Genomes Project Consortium (2012). An integrated map of genetic variation from 1,092 human genomes. *Nature*, 491:56–65.

Danecek, P. et al. (2011). The variant call format and VCFtools. *Bioinformatics*, 27:2156–8.

Durbin, R. (2014). Efficient haplotype matching and storage using the positional Burrows-Wheeler transform (PBWT). *Bioinformatics*, 30:1266–72.

Layer, R. M. et al. (2015). Efficient compression and analysis of large genetic variation datasets. *bioRxiv*.

Stade, B. et al. (2014). GrabBlur–a framework to facilitate the secure exchange of whole-exome and -genome SNV data using VCF files. *BMC Genomics*, 15 Suppl 4:S8.