SeqArray: an R/Bioconductor Package for Big Data Management of Genome-Wide Sequencing Variants

Xiuwen Zheng
Department of Biostatistics
University of Washington – Seattle

Jan 14, 2015

Contents

1 Overview

In the era of big data, thousands of gigabyte-size data sets are challenging scientists for data management, even on well-equipped hardware. Currently, next-generation sequencing techniques are being adopted to investigate common and rare variants, making the analyses of large-scale genotypic data challenging. For example, the 1000 Genomes Project has identified approximately 38 million single nucleotide polymorphisms (SNPs), 1.4 million short insertions and deletions, and more than 14,000 larger deletions from whole-genome sequencing technologies [1]. In the near future, new technologies, like third-generation whole-genome sequencing, will be enabling data to be generated at an unprecedented scale [2]. The Variant Call Format (VCF) was developed for the 1000 Genomes Project, which is a generic text format for storing DNA polymorphism data such as SNPs, insertions, deletions and structural variants, together with rich annotations [3]. However, this format is less efficient for large-scale analyses since numeric data have to be parsed from a text VCF file before further analyses. The computational burden associated with sequencing variants is especially evident with large sample and variant sizes, and it requires efficient numerical implementation and data management.

Here I introduce a high-performance C++ computing library CoreArray (http://corearray.sourceforge.net) for big-data management of genome-wide variants [4]. CoreArray was designed for developing portable and scalable storage technologies for bioinformatics data, allowing parallel computing at the multicore and cluster levels. It provides the genomic data structure (GDS) file format for array-oriented data: this is a universal data format to store multiple data variables in a single file. The CoreArray library modules are demonstrated in Figure 1. A hierarchical data structure is used to store multiple extensible data variables in the GDS format, and all datasets are stored in a single file with chunked storage layout. A document about CoreArray can be found: http://cran.r-project.org/web/packages/gdsfmt/vignettes/CoreArrayTutorial.pdf.

Here, I focus on the application of CoreArray for statisticians working in the R environment, and developed R packages gdsfmt and SeqArray to address or reduce the computational burden associated with data management of sequencing variants. Gdsfmt provides a general R interface for the CoreArray library, and SeqArray is specifically designed for data management of sequencing variants. The kernels of gdsfmt and SeqArray were written in C/C++ and highly optimized. Genotypic data and annotations are stored in a binary and array-oriented manner, offering efficient access of genetic variants using the R language. There are six key functions in SeqArray (Table 1), and most of data analyses could be done using these six functions. The 1000 Genomes Project released 39 million genetic variants for 1092 individuals, and a 26G data file was created by SeqArray to store sequencing variants with phasing information, where 2 bits were used as a primitive data type. The file size can be further reduced to 1.5G by compression algorithms without sacrificing access efficiency, since it has a large proportion of rare variants.

SeqArray will be of great interest to scientists involved in data analyses of large-scale genomic sequencing data using R environment, particularly those with limited experience of low-level C programming and

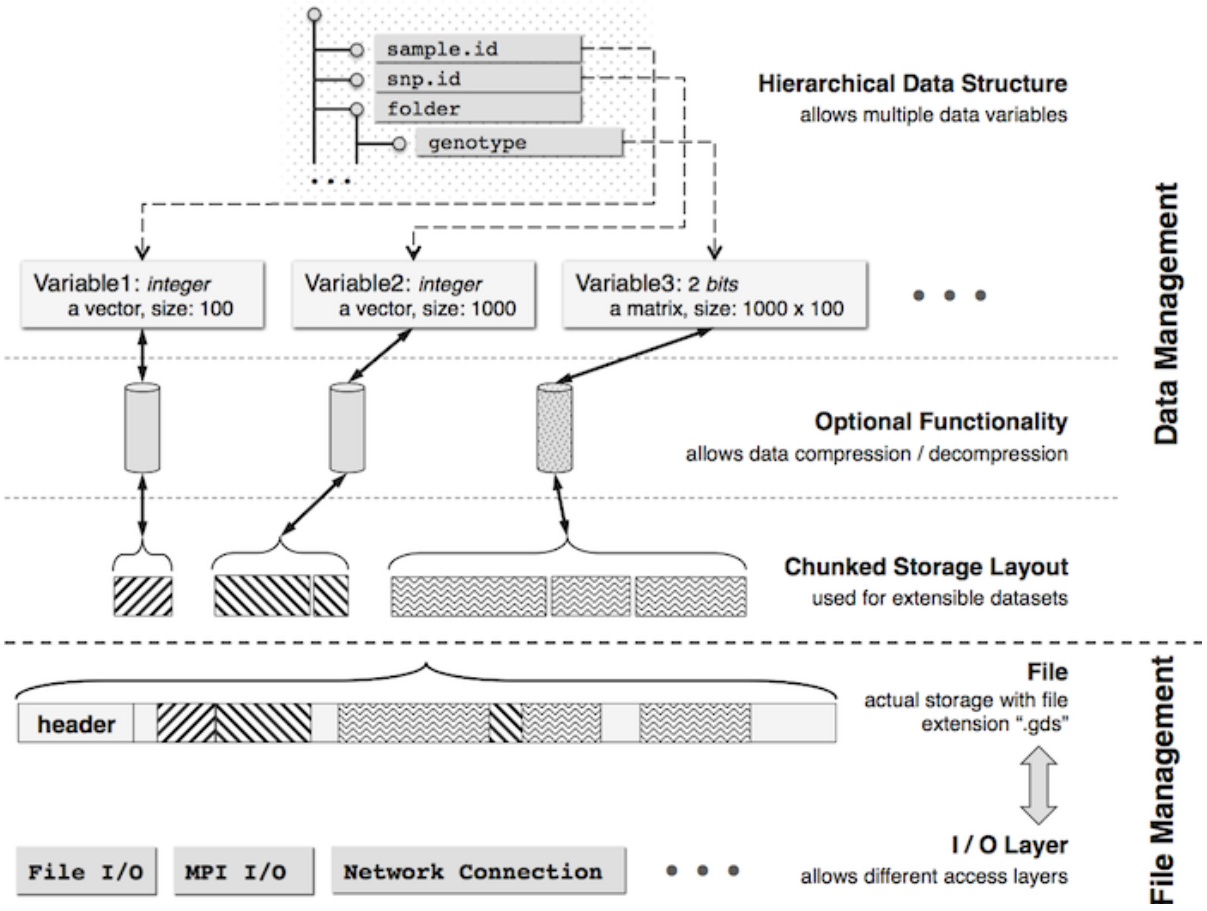


Figure 1: CoreArray library modules.

Table 1: The key functions in the SeqArray package.

Function	Description
seqVCF2GDS	Imports VCF files
seqSummary	Gets the summary of a sequencing GDS file (# of samples, # of variants, INFO/FORMAT variables, etc)
seqSetFilter	Sets a filter to sample or variant (define a subset of data)
seqGetData	Gets data from a sequencing GDS file (from a subset of data)
seqApply	Applies a user-defined function over array margins
seqParallel	Applies functions in parallel

parallel computing.

2 Preparing Data

2.1 Data formats used in SeqArray

To support efficient memory management for genome-wide numerical data, the *gdsfmt* package provides the genomic data structure (GDS) file format for array-oriented bioinformatic data based on the Core-Array library, which is a container for storing genotypic and annotation data. The GDS format supports data blocking so that only the subset of data that is being processed needs to reside in memory.

```
# load the R package SeqArray
library(SeqArray)
```

```
## Loading required package:  gdsfmt
```

Here is a typical GDS file shipped with the SeqArray package:

```
gds.fn <- seqExampleFileName("gds")
# or gds.fn <- "C:/YourFolder/Your_GDS_File.gds"
gds.fn
```

```
## [1] "E:/biocbld/bbs-3.1-bioc/tmpdir/RtmpQT6aaN/Rinst21d81b876737/SeqArray/extdata/CEU_Ex
```

```
seqSummary(gds.fn)
```

```
## File: E:\biocbld\bbs-3.1-bioc\tmpdir\RtmpQT6aaN\Rinst21d81b876737\SeqArray\extdata\CEU_E
## Sequence Variant Format: v1.0
## The number of samples: 90
## The number of variants: 1348
## The chromosomes:
##    1   10   11   12   13   14   15   16   17   18   19    2   20   21   22    3    4
##  142   70   16   62   11   61   46   84  100   54  111   59   59   23   23   81   48   6
##    6    7    8    9 <NA>
##   99   58   51   29    0
## The number of alleles per site:
##    2    3
## 1346    2
## Annotation, information variables:
##  AA, ., String, Ancestral Allele
##  AC, 1, Integer, total number of alternate alleles in called genotypes
##  AN, 1, Integer, total number of alleles in called genotypes
##  DP, 1, Integer, Total Depth
##  HM2, 0, Flag, HapMap2 membership
##  HM3, 0, Flag, HapMap3 membership
##  OR, 1, String, Previous rs number
##  GP, 1, String, GRCh37 position(s)
##  BN, 1, Integer, First dbSNP build #
## Annotation, format variables:
##  DP, ., Integer, Read Depth from MOSAIK BAM
```

`seqExampleFileName` returns the file name of a GDS file used as an example in *SeqArray*, and it is a

subset of data from the 1000 Genomes Project. `seqSummary` summarizes the genotypes and annotations stored in the GDS file.

```
# open a GDS file
genofile <- seqOpen(gds.fn)

# display the contents of the GDS file in a hierarchical structure
genofile
```

```
## Object of class "SeqVarGDSClass"
## File: E:\biocbld\bbs-3.1-bioc\tmpdir\RtmpQT6aaN\Rinst21d81b876737\SeqArray\extdata\CEU_
## +    [  ]
## |--+ description   [  ] *
## |--+ sample.id   { VStr8 90 ZIP(28.33%) }
## |--+ variant.id   { Int32 1348 ZIP(35.39%) }
## |--+ position   { Int32 1348 ZIP(86.11%) }
## |--+ chromosome   { VStr8 1348 ZIP(2.14%) }
## |--+ allele   { VStr8 1348 ZIP(16.86%) }
## |--+ genotype   [  ] *
## |  |--+ data   { Bit2 2x90x1348 ZIP(28.36%) }
## |  |--+ extra.index   { Int32 3x0 ZIP } *
## |  |--+ extra   { Int16 0 ZIP }
## |--+ phase   [  ]
## |  |--+ data   { Bit1 90x1348 ZIP(0.24%) }
## |  |--+ extra.index   { Int32 3x0 ZIP } *
## |  |--+ extra   { Bit1 0 ZIP }
## |--+ annotation   [  ]
## |  |--+ id   { VStr8 1348 ZIP(40.89%) }
## |  |--+ qual   { Float32 1348 ZIP(0.57%) }
## |  |--+ filter   { Int32,factor 1348 ZIP(0.56%) } *
## |  |--+ info   [  ]
## |  |  |--+ AA   { VStr8 1348 ZIP(23.55%) } *
## |  |  |--+ AC   { Int32 1348 ZIP(26.89%) } *
## |  |  |--+ AN   { Int32 1348 ZIP(20.29%) } *
## |  |  |--+ DP   { Int32 1348 ZIP(62.24%) } *
## |  |  |--+ HM2   { Bit1 1348 ZIP(106.51%) } *
## |  |  |--+ HM3   { Bit1 1348 ZIP(106.51%) } *
## |  |  |--+ OR   { VStr8 1348 ZIP(12.92%) } *
## |  |  |--+ GP   { VStr8 1348 ZIP(34.24%) } *
## |  |  |--+ BN   { Int32 1348 ZIP(21.31%) } *
## |  |--+ format   [  ]
## |  |  |--+ DP   [  ] *
## |  |  |  |--+ data   { Int32 90x1348 ZIP(33.83%) }
```

For those who would like to know how variable-length genotypic data and annotations are stored in an array-oriented manner, `print(, all=TRUE)` displays all contents including hidden structures:

```
# display all contents of the GDS file
print(genofile, all=TRUE)
```

```
## File: E:\biocbld\bbs-3.1-bioc\tmpdir\RtmpQT6aaN\Rinst21d81b876737\SeqArray\extdata\CEU_
## +    [  ]
## |--+ description   [  ] *
## |  |--+ vcf.header   [ data.frame ] *
## |  |  |--+ id   { VStr8 1 }
## |  |  |--+ value   { VStr8 1 }
## |--+ sample.id   { VStr8 90 ZIP(28.33%) }
## |--+ variant.id   { Int32 1348 ZIP(35.39%) }
## |--+ position   { Int32 1348 ZIP(86.11%) }
## |--+ chromosome   { VStr8 1348 ZIP(2.14%) }
## |--+ allele   { VStr8 1348 ZIP(16.86%) }
## |--+ genotype   [  ] *
## |  |--+ data   { Bit2 2x90x1348 ZIP(28.36%) }
## |  |--+ @data   { Int32 1348 ZIP(0.56%) } *
## |  |--+ extra.index   { Int32 3x0 ZIP } *
## |  |--+ extra   { Int16 0 ZIP }
## |--+ phase   [  ]
## |  |--+ data   { Bit1 90x1348 ZIP(0.24%) }
## |  |--+ extra.index   { Int32 3x0 ZIP } *
## |  |--+ extra   { Bit1 0 ZIP }
## |--+ annotation   [  ]
## |  |--+ id   { VStr8 1348 ZIP(40.89%) }
## |  |--+ qual   { Float32 1348 ZIP(0.57%) }
## |  |--+ filter   { Int32,factor 1348 ZIP(0.56%) } *
## |  |--+ info   [  ]
## |  |  |--+ AA   { VStr8 1348 ZIP(23.55%) } *
## |  |  |--+ @AA   { Int32 1348 ZIP(0.56%) } *
## |  |  |--+ AC   { Int32 1348 ZIP(26.89%) } *
## |  |  |--+ AN   { Int32 1348 ZIP(20.29%) } *
## |  |  |--+ DP   { Int32 1348 ZIP(62.24%) } *
## |  |  |--+ HM2   { Bit1 1348 ZIP(106.51%) } *
## |  |  |--+ HM3   { Bit1 1348 ZIP(106.51%) } *
## |  |  |--+ OR   { VStr8 1348 ZIP(12.92%) } *
## |  |  |--+ GP   { VStr8 1348 ZIP(34.24%) } *
## |  |  |--+ BN   { Int32 1348 ZIP(21.31%) } *
## |  |--+ format   [  ]
## |  |  |--+ DP   [  ] *
## |  |  |  |--+ data   { Int32 90x1348 ZIP(33.83%) }
## |  |  |  |--+ @data   { Int32 1348 ZIP(0.56%) } *
```

```
# close the GDS file
seqClose(genofile)
```

The output lists all variables stored in the GDS file. At the first level, it stores variables `sample.id`, `variant.id`, etc. The additional information are displayed in the square brackets indicating data type, size, compressed or not + compression ratio, where "Bit2" indicates that each byte encodes up to four alleles since one byte consists of eight bits, and "VStr8" indicates variable-length character. The annotations are stored in the directory `annotation`, which includes the fields of ID, QUAL, FILTER, INFO and FORMAT corresponding to the original VCF file(s). All of the functions in *SeqArray* require a minimum set of variables in the annotation data:

- `sample.id`, a unique identifier for each sample.
- `variant.id`, a unique identifier for each variant.
- `position`, integer, the base position of each variant on the chromosome, and 0 or NA for unknown position.
- `chromosome`, character, the chromosome code, e.g., 1-22 for autosomes, X, Y, XY (the pseudoautosomal region), M (the mitochondrial probes), and "" (a blank string) for probes with unknown chromosome.
- `allele`, character, reference and alternative alleles using comma as a separator.
- `genotype`, a folder,
  - `data`, a 3-dimensional array for genotypic data, the first dimension refers to the number of ploidy, the second is sample and the third is variant.
  - `@data`, the index for the variable `data`, and the prefix `@` is used to indicate the index. It should be equal-size as `variant.id`, which is used to specify the data size of each variant.
  - `extra.index`, an index (3-by-$*$) matrix for triploid call (look like "0/0/1" in the VCF file). E.g., for "0/0/1", the first two alleles are stored in `data`, and the last allele is saved in the variable `extra`. For each column of `extra.index`, the first value is the index of sample (starting from 1), the second value is the index of variant (starting from 1), and the last value is how many alleles remain (usually it is 1 since the first two alleles are stored in `data`) that indicates how many alleles stored in `extra` contiguously.
  - `extra`, one-dimensional array, the additional data for triploid call, each allele block corresponds to each column in `extra.index`.

The optional variables include `phase` (phasing information) and `annotation`. The variable `phase` includes:

- `data`, a matrix or 3-dimensional array for phasing information. "0" for unphased status and "1" for phased status. If it is a matrix, the first dimension is sample and the second is variant, corresponding to diploid genotypic data. If it is a 3-dimensional array, the first dimension refers to the number of ploidy minus one. More details about "/" and "—" in a VCF file can be founded: http://www.1000genomes.org/wiki/Analysis/VariantCallFormat/vcf-variant-call-format-version-41.
- `extra.index`, an index (3-by-$*$) matrix for triploid call (look like "0/0/1" in the VCF file). E.g., for "0/0/1", the first separator ("/" here) is stored in `data`, and the last separator is saved in the variable `extra`. For each column of `extra.index`, the first value is the index of sample (starting from 1), the second value is the index of variant (starting from 1), and the last value is how many separators remain (usually it is 1 since the first separator is stored in `data`) that indicates how many separator stored in `extra` contiguously.
- `extra`, one-dimensional array, the additional data of separator indicator for triploid call, each

separator block corresponds to each column in `extra.index`.

The variable `annotation` includes:

- `id`, ID semi-colon separated list of unique identifiers where available. If this is a dbSNP variant it is encouraged to use the rs number(s). No identifier should be present in more than one data record. If there is no identifier available, then a blank string is used.
- `qual`, phred-scaled quality score for the assertion made in ALT.
- `filter`, PASS if this position has passed all filters, i.e. a call is made at this position.
- `info`, additional information for each variant, according to the INFO field in a VCF file,
  - `VARIABLE_NAME`, variable. If it is fixed-length, missing value indicates that there is no entry for that variant in the VCF file.
  - `@VARIABLE_NAME`, optional if `VARIABLE_NAME` is variable-length, one-dimensional array. The prefix `@` is used to indicate the index data. It should be equal-size as `variant.id`, which is used to specify the data size of each variant.
  - `OTHER_VARIABLES, ...`
- `format`, additional information for each variant and sample, according to the FORMAT field in a VCF file,
  - `VARIABLE_NAME`, a folder,
    - `data`, a n_{samp}-by-$*$ matrix.
    - `@data`, one-dimensional array, the index data for the variable `data`, and the prefix `@` is used to indicate the index data. It should be equal-size as `variant.id`, which is used to specify the data size of each variant.
  - `OTHER_VARIABLES, ...`

2.2 Format conversion from VCF files

The *SeqArray* package provides a function `seqVCF2GDS` to reformat a VCF file, and it allows merging multiple VCF files during format conversion. The genotypic and annotation data are stored in a compressed manner.

```
# the VCF file, using the example in the SeqArray package
vcf.fn <- seqExampleFileName("vcf")
# or vcf.fn <- "C:/YourFolder/Your_VCF_File.vcf"
vcf.fn
```

```
## [1] "E:/biocbld/bbs-3.1-bioc/tmpdir/RtmpQT6aaN/Rinst21d81b876737/SeqArray/extdata/CEU_Ex
```

```
# parse the header
seqVCF.Header(vcf.fn)
```

```
## $fileformat
## [1] "VCFv4.0"
##
## $info
##    ID Number    Type                                        Description
## 1  AA      .  String                                   Ancestral Allele
```

```
## 2  AC      1 Integer total number of alternate alleles in called genotypes
## 3  AN      1 Integer           total number of alleles in called genotypes
## 4  DP      1 Integer                                             Total Depth
## 5 HM2      0    Flag                                        HapMap2 membership
## 6 HM3      0    Flag                                        HapMap3 membership
## 7  OR      1  String                                        Previous rs number
## 8  GP      1  String                                        GRCh37 position(s)
## 9  BN      1 Integer                                      First dbSNP build #
##
## $filter
## NULL
##
## $format
##   ID Number    Type                Description
## 1 GT      1  String                   Genotype
## 2 DP      . Integer Read Depth from MOSAIK BAM
##
## $alt
## NULL
##
## $contig
## NULL
##
## $assembly
## NULL
##
## $header
##           id                value
## 13 reference human_b36_both.fasta
##
## $num.ploidy
## [1] 2
##
## attr(,"class")
## [1] "SeqVCFHeaderClass"
```

The columns "Number", "Type" and "Description" are defined by the 1000 Genomes Project: http://www.1000genomes.org/wiki/Analysis/VariantCallFormat/vcf-variant-call-format-version-41. Briefly, the Number entry is an Integer that describes the number of values that can be included with the INFO field. For example, if the INFO field contains a single number, then this value should be 1; if the INFO field describes a pair of numbers, then this value should be 2 and so on. If the field has one value per alternate allele then this value should be 'A'; if the field has one value for each possible genotype then this value should be 'G'. If the number of possible values varies, is unknown, or is unbounded, then this value should be '.'. The 'Flag' type indicates that the INFO field does not contain a Value entry,

and hence the Number should be 0 in this case. Possible Types for FORMAT fields are: Integer, Float, Character, and String (this field is otherwise defined precisely as the INFO field).

```
# convert, save in "tmp.gds"
seqVCF2GDS(vcf.fn, "tmp.gds")
```

```
## The Variant Call Format (VCF) header:
##  file format: VCFv4.0
##  the number of sets of chromosomes (ploidy): 2
##  the number of samples: 90
## Parsing "E:/biocbld/bbs-3.1-bioc/tmpdir/RtmpQT6aaN/Rinst21d81b876737/SeqArray/extdata/CE
## + genotype/data   { Bit2 2x90x1348 ZIP }
## Done.
## Optimize the access efficiency ...
## Clean up the fragments of GDS file:
##  open the file "tmp.gds" (size: 216559).
##  # of fragments in total: 93.
##  save it to "tmp.gds.tmp".
##  rename "tmp.gds.tmp" (size: 216247).
##  # of fragments in total: 67.
```

```
seqSummary("tmp.gds")
```

```
## File: E:\biocbld\bbs-3.1-bioc\tmpdir\RtmpQT6aaN\Rbuild21d8603de59\SeqArray\vignettes\tmp
## Sequence Variant Format: v1.0
## The number of samples: 90
## The number of variants: 1348
## The chromosomes:
##    1   10   11   12   13   14   15   16   17   18   19    2   20   21   22    3    4
##  142   70   16   62   11   61   46   84  100   54  111   59   59   23   23   81   48   6
##    6    7    8    9 <NA>
##   99   58   51   29    0
## The number of alleles per site:
##    2    3
## 1346    2
## Annotation, information variables:
##  AA, ., String, Ancestral Allele
##  AC, 1, Integer, total number of alternate alleles in called genotypes
##  AN, 1, Integer, total number of alleles in called genotypes
##  DP, 1, Integer, Total Depth
##  HM2, 0, Flag, HapMap2 membership
##  HM3, 0, Flag, HapMap3 membership
##  OR, 1, String, Previous rs number
##  GP, 1, String, GRCh37 position(s)
##  BN, 1, Integer, First dbSNP build #
## Annotation, format variables:
##  DP, ., Integer, Read Depth from MOSAIK BAM
```

2.3 Export to a VCF File

The *SeqArray* package provides a function `seqGDS2VCF` to export data to a VCF file. The arguments `info.var` and `fmt.var` in `seqGDS2VCF` allow users to specify the variables listed in the INFO and FORMAT fields of VCF format, or remove the INFO and FORMAT information. `seqSetFilter` can be used to define a subset of data for the export.

```
# the file of GDS
gds.fn <- seqExampleFileName("gds")
# or gds.fn <- "C:/YourFolder/Your_GDS_File.gds"

# open a GDS file
genofile <- seqOpen(gds.fn)

# convert
seqGDS2VCF(genofile, "tmp.vcf.gz")
```

```
## Output: tmp.vcf.gz
## The INFO field: AA, AC, AN, DP, HM2, HM3, OR, GP, BN
## The FORMAT field: DP
## Done.
```

```
# read
z <- readLines("tmp.vcf.gz", n=20)
for (i in z) cat(substr(i, 1, 76), " ...\n", sep="")
```

```
## ##fileformat=VCFv4.0 ...
## ##fileDate=20150416 ...
## ##source=SeqArray_RPackage_v1.0 ...
## ##INFO=<ID=AA,Number=.,Type=String,Description="Ancestral Allele"> ...
## ##INFO=<ID=AC,Number=1,Type=Integer,Description="total number of alternate a ...
## ##INFO=<ID=AN,Number=1,Type=Integer,Description="total number of alleles in  ...
## ##INFO=<ID=DP,Number=1,Type=Integer,Description="Total Depth"> ...
## ##INFO=<ID=HM2,Number=0,Type=Flag,Description="HapMap2 membership"> ...
## ##INFO=<ID=HM3,Number=0,Type=Flag,Description="HapMap3 membership"> ...
## ##INFO=<ID=OR,Number=1,Type=String,Description="Previous rs number"> ...
## ##INFO=<ID=GP,Number=1,Type=String,Description="GRCh37 position(s)"> ...
## ##INFO=<ID=BN,Number=1,Type=Integer,Description="First dbSNP build #"> ...
## ##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype"> ...
## ##FORMAT=<ID=DP,Number=.,Type=Integer,Description="Read Depth from MOSAIK BA ...
## ##reference=human_b36_both.fasta ...
## #CHROM POS ID REF ALT QUAL FILTER INFO FORMAT NA06984 NA06985 NA06986 NA0698 ...
## 1 1105366 rs111751804 T C . PASS AA=T;AC=4;AN=114;DP=3251;GP=1:1115503;BN=13 ...
## 1 1105411 rs114390380 G A . PASS AA=G;AC=1;AN=106;DP=2676;GP=1:1115548;BN=13 ...
```

```
## 1 1110294 rs1320571 G A . PASS AA=A;AC=6;AN=154;DP=7610;HM2;HM3;GP=1:1120431 ...
## 1 3537996 rs2760321 T C . PASS AA=C;AC=128;AN=146;DP=3383;HM2;HM3;GP=1:35481 ...
```

```
# output BN,GP,AA,HM2 in INFO (the variables are in this order), no FORMAT
seqGDS2VCF(genofile, "tmp2.vcf.gz", info.var=c("BN","GP","AA","HM2"),
    fmt.var=character(0))
```

```
## Output: tmp2.vcf.gz
## The INFO field: BN, GP, AA, HM2
## The FORMAT field:
## Done.
```

```
# read
z <- readLines("tmp2.vcf.gz", n=15)
for (i in z) cat(substr(i, 1, 56), " ...\n", sep="")
```

```
## ##fileformat=VCFv4.0 ...
## ##fileDate=20150416 ...
## ##source=SeqArray_RPackage_v1.0 ...
## ##INFO=<ID=BN,Number=1,Type=Integer,Description="First d ...
## ##INFO=<ID=GP,Number=1,Type=String,Description="GRCh37 p ...
## ##INFO=<ID=AA,Number=.,Type=String,Description="Ancestra ...
## ##INFO=<ID=HM2,Number=0,Type=Flag,Description="HapMap2 m ...
## ##FORMAT=<ID=GT,Number=1,Type=String,Description="Genoty ...
## ##reference=human_b36_both.fasta ...
## #CHROM POS ID REF ALT QUAL FILTER INFO FORMAT NA06984 NA ...
## 1 1105366 rs111751804 T C . PASS BN=132;GP=1:1115503;AA= ...
## 1 1105411 rs114390380 G A . PASS BN=132;GP=1:1115548;AA= ...
## 1 1110294 rs1320571 G A . PASS BN=88;GP=1:1120431;AA=A;H ...
## 1 3537996 rs2760321 T C . PASS BN=100;GP=1:3548136;AA=C; ...
## 1 3538692 rs2760320 G C . PASS BN=100;GP=1:3548832;AA=G; ...
```

```
# close the GDS file
seqClose(genofile)
```

Users can use `diff`, a command line tool in Unix-like systems, to compare files line by line, in order to confirm data consistency.

```
# assuming the original VCF file is old.vcf.gz,
# call "seqVCF2GDS" for the import and "seqGDS2VCF" for the export to create a new VCF file
$ diff <(zcat old.vcf.gz) <(zcat new.vcf.gz)
# OR
$ diff <(gunzip -c old.vcf.gz) <(gunzip -c new.vcf.gz)
```

```
1a2,3
> ##fileDate=20130309
> ##source=SeqArray_RPackage_v1.0
```

```
# LOOK GOOD! There are only two lines different, and both are in the header.
```

```
# delete temporary files
unlink(c("tmp.vcf.gz", "tmp1.vcf.gz", "tmp2.vcf.gz"))
```

2.4 Modification

The *SeqArray* package provides a function `seqDelete` to remove data annotations in the INFO and FORMAT fields. It is suggested to use `cleanup.gds` in the *gdsfmt* package after calling `seqDelete` to reduce the file size. For example,

```
# the file of VCF
vcf.fn <- seqExampleFileName("vcf")
# or vcf.fn <- "C:/YourFolder/Your_VCF_File.vcf"

# convert
seqVCF2GDS(vcf.fn, "tmp.gds", verbose=FALSE)

# make sure that open with "readonly=FALSE"
genofile <- seqOpen("tmp.gds", readonly=FALSE)

# display the original structure
genofile
```

```
## Object of class "SeqVarGDSClass"
## File: E:\biocbld\bbs-3.1-bioc\tmpdir\RtmpQT6aaN\Rbuild21d8603de59\SeqArray\vignettes\tmp
## +    [  ] *
## |--+ description    [  ] *
## |--+ sample.id    { VStr8 90 ZIP(28.33%) }
## |--+ variant.id    { Int32 1348 ZIP(35.39%) }
## |--+ position    { Int32 1348 ZIP(86.11%) }
## |--+ chromosome    { VStr8 1348 ZIP(2.14%) }
## |--+ allele    { VStr8 1348 ZIP(16.86%) }
## |--+ genotype    [  ] *
## |  |--+ data    { Bit2 2x90x1348 ZIP(28.36%) }
## |  |--+ extra.index    { Int32 3x0 ZIP } *
## |  |--+ extra    { Int16 0 ZIP }
## |--+ phase    [  ]
## |  |--+ data    { Bit1 90x1348 ZIP(0.24%) }
## |  |--+ extra.index    { Int32 3x0 ZIP } *
## |  |--+ extra    { Bit1 0 ZIP }
## |--+ annotation    [  ]
## |  |--+ id    { VStr8 1348 ZIP(40.89%) }
## |  |--+ qual    { Float32 1348 ZIP(0.57%) }
## |  |--+ filter    { Int32,factor 1348 ZIP(0.56%) } *
```

```
## |   |--+ info   [  ]
## |   |   |--+ AA   { VStr8 1348 ZIP(23.55%) } *
## |   |   |--+ AC   { Int32 1348 ZIP(26.89%) } *
## |   |   |--+ AN   { Int32 1348 ZIP(20.29%) } *
## |   |   |--+ DP   { Int32 1348 ZIP(62.24%) } *
## |   |   |--+ HM2   { Bit1 1348 ZIP(106.51%) } *
## |   |   |--+ HM3   { Bit1 1348 ZIP(106.51%) } *
## |   |   |--+ OR   { VStr8 1348 ZIP(12.92%) } *
## |   |   |--+ GP   { VStr8 1348 ZIP(34.24%) } *
## |   |   |--+ BN   { Int32 1348 ZIP(21.31%) } *
## |   |--+ format   [  ]
## |   |   |--+ DP   [  ] *
## |   |   |   |--+ data   { Int32 90x1348 ZIP(33.83%) }
```

```
# delete "HM2", "HM3", "AA", "OR" and "DP"
seqDelete(genofile, info.varname=c("HM2", "HM3", "AA", "OR"),
    format.varname="DP")
```

```
# display
genofile
```

```
## Object of class "SeqVarGDSClass"
## File: E:\biocbld\bbs-3.1-bioc\tmpdir\RtmpQT6aaN\Rbuild21d8603de59\SeqArray\vignettes\tmp
## +    [  ] *
## |--+ description   [  ] *
## |--+ sample.id   { VStr8 90 ZIP(28.33%) }
## |--+ variant.id   { Int32 1348 ZIP(35.39%) }
## |--+ position   { Int32 1348 ZIP(86.11%) }
## |--+ chromosome   { VStr8 1348 ZIP(2.14%) }
## |--+ allele   { VStr8 1348 ZIP(16.86%) }
## |--+ genotype   [  ] *
## |   |--+ data   { Bit2 2x90x1348 ZIP(28.36%) }
## |   |--+ extra.index   { Int32 3x0 ZIP } *
## |   |--+ extra   { Int16 0 ZIP }
## |--+ phase   [  ]
## |   |--+ data   { Bit1 90x1348 ZIP(0.24%) }
## |   |--+ extra.index   { Int32 3x0 ZIP } *
## |   |--+ extra   { Bit1 0 ZIP }
## |--+ annotation   [  ]
## |   |--+ id   { VStr8 1348 ZIP(40.89%) }
## |   |--+ qual   { Float32 1348 ZIP(0.57%) }
## |   |--+ filter   { Int32,factor 1348 ZIP(0.56%) } *
## |   |--+ info   [  ]
## |   |   |--+ AA   { VStr8 1348 ZIP(23.55%) } *
## |   |   |--+ AC   { Int32 1348 ZIP(26.89%) } *
```

```
## |   |   |--+ AN   { Int32 1348 ZIP(20.29%) } *
## |   |   |--+ DP   { Int32 1348 ZIP(62.24%) } *
## |   |   |--+ HM2   { Bit1 1348 ZIP(106.51%) } *
## |   |   |--+ HM3   { Bit1 1348 ZIP(106.51%) } *
## |   |   |--+ OR   { VStr8 1348 ZIP(12.92%) } *
## |   |   |--+ GP   { VStr8 1348 ZIP(34.24%) } *
## |   |   |--+ BN   { Int32 1348 ZIP(21.31%) } *
## |   |--+ format   [  ]
## |   |   |--+ DP   [  ] *
## |   |   |   |--+ data   { Int32 90x1348 ZIP(33.83%) }
```

```
# close the GDS file
seqClose(genofile)

# clean up the fragments, reduce the file size
cleanup.gds("tmp.gds")
```

```
## Clean up the fragments of GDS file:
##   open the file "tmp.gds" (size: 216247).
##   # of fragments in total: 67.
##   save it to "tmp.gds.tmp".
##   rename "tmp.gds.tmp" (size: 216247).
##   # of fragments in total: 67.
```

3 Data Processing

3.1 Get Data

```
# open a GDS file
gds.fn <- seqExampleFileName("gds")
genofile <- seqOpen(gds.fn)
```

It is suggested to use `seqGetData` to take out data from the GDS file since this function can take care of variable-length data and multi-allelic genotypes, although users could also use `read.gdsn` in the *gdsfmt* package to read data.

```
# take out sample id
head(samp.id <- seqGetData(genofile, "sample.id"))
```

```
## [1] "NA06984" "NA06985" "NA06986" "NA06989" "NA06994" "NA07000"
```

```
# take out variant id
head(variant.id <- seqGetData(genofile, "variant.id"))
```

```
## [1] 1 2 3 4 5 6
```

```
# get "chromosome"
table(seqGetData(genofile, "chromosome"))
```

```
##
##   1  10  11  12  13  14  15  16  17  18  19   2  20  21  22   3   4   5   6   7   8   9
## 142  70  16  62  11  61  46  84 100  54 111  59  59  23  23  81  48  61  99  58  51  29
```

```
# get "allele"
head(seqGetData(genofile, "allele"))
```

```
## [1] "T,C" "G,A" "G,A" "T,C" "G,C" "C,T"
```

```
# get "annotation/info/GP"
head(seqGetData(genofile, "annotation/info/GP"))
```

```
## [1] "1:1115503" "1:1115548" "1:1120431" "1:3548136" "1:3548832" "1:3551737"
```

Users can set a filter to sample and/or variant by `seqSetFilter`. For example, we consider a data subset consisting of three samples and four variants:

```
# set sample and variant filters
seqSetFilter(genofile, sample.id=samp.id[c(2,4,6)])
```

```
## # of selected samples: 3
```

```
set.seed(100)
seqSetFilter(genofile, variant.id=sample(variant.id, 4))
```

```
## # of selected variants: 4
```

```
# get "allele"
seqGetData(genofile, "allele")
```

```
## [1] "T,A" "G,A" "G,C" "A,G"
```

Get genotypic data, it is a 3-dimensional array with respect to allele, sample and variant. "0" refers to the reference allele (or the first allele in the variable `allele`), "1" for the second allele, and so on, while NA is missing allele.

```
# get genotypic data
seqGetData(genofile, "genotype")
```

```
## , , 1
##
##       sample
## allele [,1] [,2] [,3]
##   [1,]    0    0    0
##   [2,]    0    0    0
##
## , , 2
##
```

```
##       sample
## allele [,1] [,2] [,3]
##   [1,]    1    0    0
##   [2,]    0    0    0
##
## , , 3
##
##       sample
## allele [,1] [,2] [,3]
##   [1,]    0    0    0
##   [2,]    0    0    0
##
## , , 4
##
##       sample
## allele [,1] [,2] [,3]
##   [1,]    0    0    0
##   [2,]    0    0    0
```

Now let us take a look at a variable-length dataset `annotation/info/AA`, which corresponds to the INFO column in the original VCF file. There are four variants, each variant has data with size ONE (`$length`), and data are saved in `$data` contiguously. `$length` could be ZERO indicating no data for that variant.

```
# get "annotation/info/AA", a variable-length dataset
seqGetData(genofile, "annotation/info/AA")
```

```
## $length
## [1] 1 1 1 1
##
## $data
## [1] "T" "G" "G" "A"
```

Another variable-length dataset is `annotation/format/DP` corresponding to the FORMAT column in the original VCF file. Again, `$length` refers to the size of each variant, and data are saved in `$data` contiguously with respect to the dimension `variant`. `$length` could be ZERO indicating no data for that variant.

```
# get "annotation/format/DP", a variable-length dataset
seqGetData(genofile, "annotation/format/DP")
```

```
## $length
## [1] 1 1 1 1
##
## $data
##       variant
## sample [,1] [,2] [,3] [,4]
```

```
##   [1,]    1   12   18    9
##   [2,]   11   79   34  130
##   [3,]   52   31   56   81
```

3.2 Apply Functions Over Array Margins

SeqArray provides `seqApply` to apply a user-defined function over array margins, which is coded in C/C++. It is suggested to use `seqApply` instead of `apply.gdsn` in the *gdsfmt* package, since this function can take care of variable-length data and multi-allelic genotypes. For example, read two variables `genotype` and `annotation/id` variant by variant:

```
# set sample and variant filters
set.seed(100)
seqSetFilter(genofile, sample.id=samp.id[c(2,4,6)],
    variant.id=sample(variant.id, 4))
```

```
## # of selected samples: 3
## # of selected variants: 4
```

```
# read multiple variables variant by variant
seqApply(genofile, c(geno="genotype", id="annotation/id"),
    FUN=function(x) print(x), margin="by.variant", as.is="none")
```

```
## $geno
##       sample
## allele [,1] [,2] [,3]
##   [1,]    0    0    0
##   [2,]    0    0    0
##
## $id
## [1] "rs114199731"
##
## $geno
##       sample
## allele [,1] [,2] [,3]
##   [1,]    1    0    0
##   [2,]    0    0    0
##
## $id
## [1] "rs12347"
##
## $geno
##       sample
## allele [,1] [,2] [,3]
##   [1,]    0    0    0
```

```
##   [2,]    0    0    0
##
## $id
## [1] "rs41269293"
##
## $geno
##       sample
## allele [,1] [,2] [,3]
##   [1,]    0    0    0
##   [2,]    0    0    0
##
## $id
## [1] "rs35349730"
```

```
# remove the sample and variant filters
seqSetFilter(genofile)
```

```
## # of selected samples: 90
## # of selected variants: 1348
```

```
# get the numbers of alleles per variant
z <- seqApply(genofile, "allele",
    FUN=function(x) length(unlist(strsplit(x,","))), as.is="integer")
table(z)
```

```
## z
##    2    3
## 1346    2
```

Another example is to use the argument `var.index` in the function `seqApply` to include external information in the analysis, where the variable `index` in the user-defined `FUN` is an index of the specified dimension starting from 1 (e.g., variant).

```
HM3 <- seqGetData(genofile, "annotation/info/HM3")
```

```
# Now HM3 is a global variable
# print out RS id if the frequency of reference allele is less than 0.5% and it is HM3
seqApply(genofile, c(geno="genotype", id="annotation/id"),
    FUN = function(index, x) {
        p <- mean(x$geno == 0, na.rm=TRUE)  # the frequency of reference allele
        if ((p < 0.005) & HM3[index]) print(x$id) },
    as.is="none", var.index="relative", margin="by.variant")
```

```
## [1] "rs10908722"
## [1] "rs939777"
## [1] "rs698673"
## [1] "rs943542"
## [1] "rs2062163"
```

```
## [1] "rs7142228"
## [1] "rs322965"
## [1] "rs2507733"
```

3.3 Apply Functions in Parallel

Now, let us consider an example of calculating the frequency of reference allele, and this calculation can be done using `seqApply` and `seqParallel`. Let's try the uniprocessor implementation first.

```
# calculate the frequency of reference allele,
afreq <- seqApply(genofile, "genotype", FUN=function(x) mean(x==0, na.rm=TRUE),
    as.is="double", margin="by.variant")
length(afreq)
```

```
## [1] 1348
```

```
summary(afreq)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.8114  0.9719  0.8488  0.9942  1.0000
```

A multi-process implementation:

```
# load the "parallel" package
library(parallel)

# Use option cl.core to choose an appropriate cluster size (or # of cores)
cl <- makeCluster(getOption("cl.cores", 2))
```

```
# run in parallel
afreq <- seqParallel(cl, genofile, FUN = function(gdsfile) {
        seqApply(gdsfile, "genotype", as.is="double",
            FUN=function(x) mean(x==0, na.rm=TRUE))
    }, split = "by.variant")

length(afreq)
```

```
## [1] 1348
```

```
summary(afreq)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.8114  0.9719  0.8488  0.9942  1.0000
```

```
# Since we created the cluster manually, we should stop it:
stopCluster(cl)
```

```
seqClose(genofile)
```

3.4 Integration with Other Packages in Bioconductor

An R/Bioconductor package *SeqVarTools* is available on Bioconductor, which defines S4 classes and methods for other common operations and analyses on SeqArray datasets.

4 Examples

In this section, a GDS file shipped with the package is used as an example:

```
# open a GDS file
gds.fn <- seqExampleFileName("gds")
genofile <- seqOpen(gds.fn)
```

4.1 The performance of seqApply

Let us try three approaches to export unphased genotypes: 1) the for loop in R; 2) vectorize the function in R; 3) the for loop in `seqApply`. The function `seqApply` has been highly optimized by blocking the computations to exploit the high-speed memory instead of disk. The results of running times (listed as follows) indicate that `seqApply` works well and is comparable with vectorization in R.

1) the for loop in R:

```
> system.time({
>     geno <- seqGetData(genofile, "genotype")
>     gc <- matrix("", nrow=dim(geno)[2], ncol=dim(geno)[3])
>     for (i in 1:dim(geno)[3])
>     {
>         for (j in 1:dim(geno)[2])
>         gc[j,i] <- paste(geno[1,j,i], geno[2,j,i], sep="/")
>     }
>     gc[gc == "NA/NA"] <- NA
>     gc
> })
```

```
   user  system elapsed
  2.185   0.019   2.386       <- the function takes 2.4 seconds
```

```
> dim(gc)
```

```
[1]   90 1348
```

```
> table(c(gc))
```

```
  0/0   0/1   1/0   1/1  <NA>
88350  7783  8258  8321  8608
```

2) Vectorize the function in R:

```
> system.time({
>     geno <- seqGetData(genofile, "genotype")
>     gc <- matrix(paste(geno[1,,], geno[2,,], sep="/"),
>         nrow=dim(geno)[2], ncol=dim(geno)[3])
>     gc[gc == "NA/NA"] <- NA
>     gc
> })

   user  system elapsed
  0.134   0.002   0.147       <- the function takes 0.15 seconds
```

3) the for loop in `seqApply`:

```
> system.time({
>     gc <- seqApply(genofile, "genotype",
>         function(x) { paste(x[1,], x[2,], sep="/") },
>         margin="by.variant", as.is="list")
>     gc2 <- matrix(unlist(gc), ncol=length(gc))
>     gc2[gc2 == "NA/NA"] <- NA
>     gc2
> })

   user  system elapsed
  0.157   0.002   0.168       <- the function takes 0.17 seconds
```

4.2 Missing Rates

1) Calculate the missing rate per variant:

```
m.variant <- local({
    # get ploidy, the number of samples and variants
    z <- seqSummary(genofile, "genotype")
    # dm[1] -- # of selected samples, dm[2] -- # of selected variants
    dm <- z$seldim
    # ploidy
    ploidy <- z$dim[1]

    # apply the function marginally
    m <- seqApply(genofile, "genotype", function(x) { sum(is.na(x)) },
        margin="by.variant", as.is="integer")

    # output
    m / (ploidy * dm[1])
})
```

```
length(m.variant)
```

```
## [1] 1348
```

```
summary(m.variant)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.00000 0.01111 0.07095 0.07778 0.96670
```

2) Calculate the missing rate per sample:

```
m.sample <- local({
    # get ploidy, the number of samples and variants
    z <- seqSummary(genofile, "genotype")
    # dm[1] -- # of selected samples, dm[2] -- # of selected variants
    dm <- z$seldim
    # ploidy
    ploidy <- z$dim[1]

    # need a global variable (only available in the bracket of "local")
    n <- integer(dm[1])

    # apply the function marginally
    # use "<<-" operator to find "n" in the parent environment
    seqApply(genofile, "genotype", function(x) { n <<- n + colSums(is.na(x)) },
        margin="by.variant", as.is="none")

    # output
    n / (ploidy * dm[2])
})

length(m.sample)
```

```
## [1] 90
```

```
summary(m.sample)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.001484 0.014090 0.054150 0.070950 0.117000 0.264100
```

4.2.1 Multi-process Implementation

```
# load the "parallel" package
library(parallel)

# Use option cl.core to choose an appropriate cluster size (or # of cores)
cl <- makeCluster(getOption("cl.cores", 2))
```

1) Calculate the missing rate per variant:

```
# run in parallel
m.variant <- local({
    # get ploidy, the number of samples and variants
    z <- seqSummary(genofile, "genotype")
    # dm[1] -- # of selected samples, dm[2] -- # of selected variants
    dm <- z$seldim
    # ploidy
    ploidy <- z$dim[1]

    # run in parallel
    m <- seqParallel(cl, genofile, FUN = function(gdsfile) {
        # apply the function marginally
        seqApply(gdsfile, "genotype", function(x) { sum(is.na(x)) },
            margin="by.variant", as.is="integer")
    }, split = "by.variant")

    # output
    m / (ploidy * dm[1])
})

summary(m.variant)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.00000 0.01111 0.07095 0.07778 0.96670
```

2) Calculate the missing rate per sample:

```
m.sample <- local({
    # run in parallel
    m <- seqParallel(cl, genofile, FUN = function(gdsfile)
        {
            # dm[1] -- # of selected samples, dm[2] -- # of selected variants
            dm <- seqSummary(gdsfile, "genotype")$seldim

            # need a global variable (only available in the bracket of "local")
            n <- integer(dm[1])

            # apply the function marginally
            # use "<<-" operator to find "n" in the parent environment
            seqApply(gdsfile, "genotype", function(x) { n <<- n + colSums(is.na(x)) },
                margin="by.variant", as.is="none")

            # output
            n
        }, .combine = "+",      # sum all variables "n" of different processes
```

```
        split = "by.variant")

    # get ploidy, the number of samples and variants
    z <- seqSummary(genofile, "genotype")
    # dm[1] -- # of selected samples, dm[2] -- # of selected variants
    dm <- z$seldim
    # ploidy
    ploidy <- z$dim[1]

    # output
    m / (ploidy * dm[2])
})

summary(m.sample)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.001484 0.014090 0.054150 0.070950 0.117000 0.264100
```

```
# Since we created the cluster manually, we should stop it:
stopCluster(cl)
```

4.3 Allele Frequency

Calculate the frequency of reference allele:

```
# apply the function variant by variant
afreq <- seqApply(genofile, "genotype", function(x) { mean(x==0, na.rm=TRUE) },
    as.is="double", margin="by.variant")

length(afreq)
```

```
## [1] 1348
```

```
summary(afreq)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.8114  0.9719  0.8488  0.9942  1.0000
```

4.3.1 Multi-process Implementation

```
# load the "parallel" package
library(parallel)

# Use option cl.core to choose an appropriate cluster size (or # of cores)
cl <- makeCluster(getOption("cl.cores", 2))
```

```
# run in parallel
afreq <- seqParallel(cl, genofile, FUN = function(gdsfile) {
        seqApply(gdsfile, "genotype", as.is="double",
            FUN=function(x) mean(x==0, na.rm=TRUE))
    }, split = "by.variant")

length(afreq)
```

```
## [1] 1348
```

```
summary(afreq)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.8114  0.9719  0.8488  0.9942  1.0000
```

```
# Since we created the cluster manually, we should stop it:
stopCluster(cl)
```

4.4 Principal Component Analysis

In the principal component analysis, we employ the dosage of reference alleles to avoid confusion of multiple alleles. The genetic correlation matrix is defined as $M = [m_{j,j'}]$:

$$m_{j,j'} = \frac{1}{L}\sum_{l=1}^{L}\frac{(g_{j,l} - 2p_l)(g_{j',l} - 2p_l)}{p_l(1-p_l)}$$

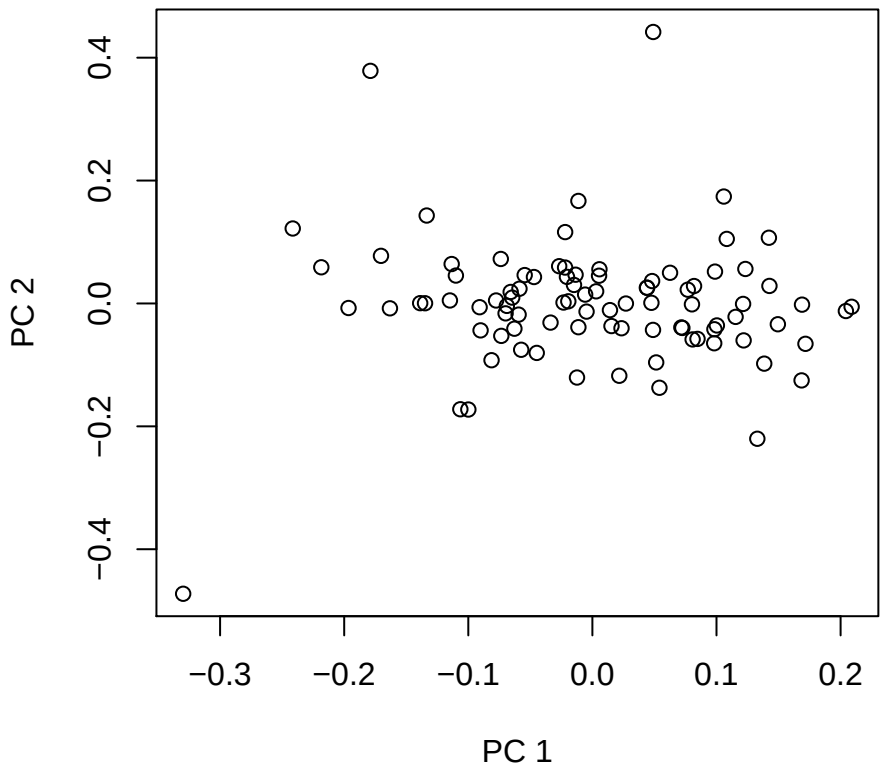
where $g_{j,l}$ is a genotype of individual j at locus l ($\in \{0, 1, 2\}$, # of reference allele), p_l is the frequency of reference allele and there are L loci in total.

```
# genetic correlation matrix
genmat <- local({
    # get the number of samples and variants
    # dm[1] -- # of selected samples, dm[2] -- # of selected variants
    dm <- seqSummary(genofile, "genotype")$seldim

    # need a global variable (only available in the bracket of "local")
    s <- matrix(0.0, nrow=dm[1], ncol=dm[1])

    # apply the function variant by variant
    # use "<<-" operator to find "s" in the parent environment
    seqApply(genofile, "genotype", function(x) {
        g <- (x==0)                    # indicator of reference allele
        p <- mean(g, na.rm=TRUE)       # allele frequency
        g2 <- colSums(g) - 2*p         # genotypes adjusted by allele frequency
        m <- (g2 %o% g2) / (p*(1-p))   # genetic correlation matrix
        m[!is.finite(m)] <- 0          # correct missing values
```

```
        s <<- s + m                    # output to the global variable "s"
    }, margin="by.variant", as.is="none")

    # output, scaled by the trace of matrix "s" over the number of samples
    s / (sum(diag(s)) / dm[1])
})

# eigen-decomposition
eig <- eigen(genmat)

# eigenvalues
head(eig$value)
```

```
## [1] 1.984696 1.867190 1.821147 1.759780 1.729839 1.678466
```

```
# eigenvectors
plot(eig$vectors[,1], eig$vectors[,2], xlab="PC 1", ylab="PC 2")
```



4.4.1 Multi-process Implementation

```
# load the "parallel" package
library(parallel)

# Use option cl.core to choose an appropriate cluster size (or # of cores)
cl <- makeCluster(getOption("cl.cores", 2))
```

```
genmat <- seqParallel(cl, genofile, FUN = function(gdsfile)
    {
        # get the number of samples and variants
        # dm[1] -- # of selected samples, dm[2] -- # of selected variants
        dm <- seqSummary(gdsfile, "genotype")$seldim

        # need a global variable (only available in the bracket of "local")
        s <- matrix(0.0, nrow=dm[1], ncol=dm[1])

        # apply the function variant by variant
        # use "<<-" operator to find "s" in the parent environment
        seqApply(gdsfile, "genotype", function(x) {
            g <- (x==0)                     # indicator of reference allele
            p <- mean(g, na.rm=TRUE)        # allele frequency
            g2 <- colSums(g) - 2*p          # genotypes adjusted by allele frequency
            m <- (g2 %o% g2) / (p*(1-p))    # genetic correlation matrix
            m[!is.finite(m)] <- 0           # correct missing values
            s <<- s + m                     # output to the global variable "s"
        }, margin="by.variant", as.is="none")

        # output
        s
    }, .combine = "+",      # sum all variables "s" of different processes
    split = "by.variant")

# finally, scaled by the trace of matrix "s" over the number of samples
dm <- seqSummary(genofile, "genotype")$seldim
genmat <- genmat / (sum(diag(genmat)) / dm[1])

# eigen-decomposition
eig <- eigen(genmat)
# eigenvalues
head(eig$value)
```

```
## [1] 1.984696 1.867190 1.821147 1.759780 1.729839 1.678466
```

```
# Since we created the cluster manually, we should stop it:
stopCluster(cl)
```

4.5 Individual Inbreeding Coefficient

To calculate an individual inbreeding coefficient using SNP genotype data, I demonstrate how to use `seqApply` to calculate Visscher's estimator described in Yang *et al.* (2010) [5]. The estimator of individual inbreeding coefficient is defined as

$$\hat{\theta} = \frac{1}{L}\sum_{l=1}^{L} \frac{g_l^2 - g_l(1+2p_l) + 2p_l^2}{2p_l(1-p_l)}$$

where g_l is a SNP genotype at locus l ($\in \{0, 1, 2\}$, # of reference allele), p_l is the frequency of reference allele and there are L loci in total.

```
coeff <- local({
    # get the number of samples and variants
    # dm[1] -- # of selected samples, dm[2] -- # of selected variants
    dm <- seqSummary(genofile, "genotype")$seldim

    # need global variables (only available in the bracket of "local")
    n <- integer(dm[1])
    s <- double(dm[1])

    # apply the function variant by variant
    # use "<<-" operator to find "n" and "s" in the parent environment
    seqApply(genofile, "genotype", function(x) {
        p <- mean(x==0, na.rm=TRUE)       # allele frequency
        g <- colSums(x==0)                # genotypes: # of reference allele
        d <- (g*g - g*(1 + 2*p) + 2*p*p) / (2*p*(1-p))
        n <<- n + is.finite(d)            # output to the global variable "n"
        d[!is.finite(d)] <- 0
        s <<- s + d                       # output to the global variable "s"
    }, margin="by.variant", as.is="none")

    # output
    s / n
})

length(coeff)
```

```
## [1] 90
```

```
summary(coeff)
```

```
##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
## -1.004e-01 -3.662e-02 -1.981e-02 -2.002e-02  5.300e-07  4.181e-02
```

4.5.1 Multi-process Implementation

```
# load the "parallel" package
library(parallel)

# Use option cl.core to choose an appropriate cluster size (or # of cores)
cl <- makeCluster(getOption("cl.cores", 2))
```

```
coeff <- seqParallel(cl, genofile, FUN = function(gdsfile)
    {
        # get the number of samples and variants
        # dm[1] -- # of selected samples, dm[2] -- # of selected variants
        dm <- seqSummary(gdsfile, "genotype")$seldim

        # need global variables (only available in the bracket)
        n <- integer(dm[1])
        s <- double(dm[1])

        # apply the function variant by variant
        # use "<<-" operator to find "n" and "s" in the parent environment
        seqApply(gdsfile, "genotype", function(x) {
            p <- mean(x==0, na.rm=TRUE)       # allele frequency
            g <- colSums(x==0)                # genotypes: # of reference allele
            d <- (g*g - g*(1 + 2*p) + 2*p*p) / (2*p*(1-p))
            n <<- n + is.finite(d)            # output to the global variable "n"
            d[!is.finite(d)] <- 0
            s <<- s + d                       # output to the global variable "s"
        }, margin="by.variant", as.is="none")

        # output
        list(s=s, n=n)
    }, # sum all variables "s" and "n" of different processes
    .combine = function(x1,x2) { list(s = x1$s + x2$s, n = x1$n + x2$n) },
    split = "by.variant")

# finally, average!
coeff <- coeff$s / coeff$n

summary(coeff)
```

```
##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
## -1.004e-01 -3.662e-02 -1.981e-02 -2.002e-02  5.300e-07  4.181e-02
```

```
# Since we created the cluster manually, we should stop it:
stopCluster(cl)
```

4.6 Seamless R and C++ Integration

The Rcpp package provides R functions as well as a C++ library which facilitate the integration of R and C++.

```
library(Rcpp)
```

The user can dynamically define an inline C/C++ function in R.

```
cppFunction('double RefAlleleFreq(IntegerMatrix x)
{
    int nrow = x.nrow(), ncol = x.ncol();
    int cnt=0, zero_cnt=0, g;
    for (int i = 0; i < nrow; i++)
    {
        for (int j = 0; j < ncol; j++)
        {
            if ((g = x(i, j)) != NA_INTEGER)
            {
                cnt ++;
                if (g == 0) zero_cnt ++;
            }
        }
    }
    return double(zero_cnt) / cnt;
}')
```

Now, "RefAlleleFreq" is a function in R.

```
RefAlleleFreq

afreq <- seqApply(genofile, "genotype", RefAlleleFreq,
    as.is="double", margin="by.variant")

summary(afreq)
```

```
# close the GDS file
seqClose(genofile)
```

5 The 1000 Genomes Project

The 1000 Genomes Project datasets consist of 39,706,715 variants and 1,092 study samples. The original VCF data files for the 1000 Genomes Project were downloaded from http://ftp.1000genomes.ebi.ac.uk/vol1/ftp/phase1/analysis_results/integrated_call_sets/, and all compressed VCF files are ∼ 180G in total. The function "seqVCF2GDS" in the R package SeqArray was used to

convert and merge all VCF files, and all ∼39M variants and annotations were extracted from the VCF files without losing any information. A single GDS file of ∼140G was created, and all data variables were stored in a compressed manner.

The detailed information are listed here:

```
> seqSummary("G1000-Release.gds")

Sequence Variant Format: v1.0
The number of samples: 1092
The number of variants: 39706715
The chromosomes:
      1      10      11      12      13      14      15      16      17      18
3007196 1882663 1894908 1828006 1373000 1258254 1130554 1210619 1046733 1088820
     19       2      20      21      22       3       4       5       6       7
 816115 3307592  855166  518965  494328 2763454 2736765 2530217 2424425 2215231
      8       9       X    <NA>
2183839 1652388 1487477       0
The number of alleles per site:
       2
39706715
Annotation, information variables:
    LDAF, 1, Float, MLE Allele Frequency Accounting for LD
    AVGPOST, 1, Float, Average posterior probability from MaCH/Thunder
    RSQ, 1, Float, Genotype imputation quality from MaCH/Thunder
    ERATE, 1, Float, Per-marker Mutation rate from MaCH/Thunder
    THETA, 1, Float, Per-marker Transition rate from MaCH/Thunder
    CIEND, 2, Integer, Confidence interval around END for imprecise variants
    CIPOS, 2, Integer, Confidence interval around POS for imprecise variants
    END, 1, Integer, End position of the variant described in this record
    HOMLEN, ., Integer, Length of base pair identical micro-homology at event breakpoints
    HOMSEQ, ., String, Sequence of base pair identical micro-homology at event breakpoints
    SVLEN, 1, Integer, Difference in length between REF and ALT alleles
    SVTYPE, 1, String, Type of structural variant
    AC, ., Integer, Alternate Allele Count
    AN, 1, Integer, Total Allele Count
    AA, 1, String, Ancestral Allele, ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/pilot_data/te
    AF, 1, Float, Global Allele Frequency based on AC/AN
    AMR_AF, 1, Float, Allele Frequency for samples from AMR based on AC/AN
    ASN_AF, 1, Float, Allele Frequency for samples from ASN based on AC/AN
    AFR_AF, 1, Float, Allele Frequency for samples from AFR based on AC/AN
    EUR_AF, 1, Float, Allele Frequency for samples from EUR based on AC/AN
    VT, 1, String, indicates what type of variant the line represents
    SNPSOURCE, ., String, indicates if a snp was called when analysing the low coverage or
Annotation, format variables:
    DS, 1, Float, Genotype dosage from MaCH/Thunder
```

```
    GL, ., Float, Genotype Likelihoods
+   [   ]
|--+ description    [   ] *
|--+ sample.id  [ VStr8 1092 ZIP(26.02%) ]
|--+ variant.id [ Int32 39706715 ZIP(34.58%) ]
|--+ position   [ Int32 39706715 ZIP(45.97%) ]
|--+ chromosome [ VStr8 39706715 ZIP(0.10%) ]
|--+ allele [ VStr8 39706715 ZIP(20.59%) ]
|--+ genotype   [   ] *
|  |--+ data    [ Bit2 2x1092x39706715 ZIP(5.61%) ]
|  |--+ extra.index [ Int32 3x0 ZIP ] *
|  |--+ extra   [ Int16 0 ZIP ]
|--+ phase  [   ]
|  |--+ data    [ Bit1 1092x39706715 ZIP(0.11%) ]
|  |--+ extra.index [ Int32 3x0 ZIP ] *
|  |--+ extra   [ Bit1 0 ZIP ]
|--+ annotation [   ]
|  |--+ id  [ VStr8 39706715 ZIP(38.26%) ]
|  |--+ qual    [ Float32 39706715 ZIP(3.44%) ]
|  |--+ filter  [ Int32,factor 39706715 ZIP(0.10%) ] *
|  |--+ info    [   ]
|  |  |--+ LDAF [ Float32 39706715 ZIP(47.36%) ] *
|  |  |--+ AVGPOST  [ Float32 39706715 ZIP(28.40%) ] *
|  |  |--+ RSQ  [ Float32 39706715 ZIP(58.30%) ] *
|  |  |--+ ERATE    [ Float32 39706715 ZIP(13.38%) ] *
|  |  |--+ THETA    [ Float32 39706715 ZIP(20.83%) ] *
|  |  |--+ CIEND    [ Int32 2x39706715 ZIP(14.41%) ] *
|  |  |--+ CIPOS    [ Int32 2x39706715 ZIP(14.41%) ] *
|  |  |--+ END  [ Int32 39706715 ZIP(26.38%) ] *
|  |  |--+ HOMLEN   [ Int32 8856 ZIP(20.11%) ] *
|  |  |--+ HOMSEQ   [ VStr8 7050 ZIP(27.39%) ] *
|  |  |--+ SVLEN    [ Int32 39706715 ZIP(26.38%) ] *
|  |  |--+ SVTYPE   [ VStr8 39706715 ZIP(1.70%) ] *
|  |  |--+ AC   [ Int32 39706715 ZIP(29.30%) ] *
|  |  |--+ AN   [ Int32 39706715 ZIP(0.10%) ] *
|  |  |--+ AA   [ VStr8 39706715 ZIP(20.51%) ] *
|  |  |--+ AF   [ Float32 39706715 ZIP(22.77%) ] *
|  |  |--+ AMR_AF   [ Float32 39706715 ZIP(21.79%) ] *
|  |  |--+ ASN_AF   [ Float32 39706715 ZIP(22.93%) ] *
|  |  |--+ AFR_AF   [ Float32 39706715 ZIP(23.61%) ] *
|  |  |--+ EUR_AF   [ Float32 39706715 ZIP(23.07%) ] *
|  |  |--+ VT   [ VStr8 39706715 ZIP(1.31%) ] *
|  |  |--+ SNPSOURCE    [ VStr8 38671749 ZIP(0.35%) ] *
|  |--+ format  [   ]
```

```
|   |   |--+ DS    [   ] *
|   |   |  |--+ data  [ Float32 1092x39706715 ZIP(2.88%) ]
|   |   |--+ GL    [   ] *
|   |   |  |--+ data  [ Float32 1092x119120145 ZIP(27.61%) ]
```

6 Resources

1. CoreArray project: http://corearray.sourceforge.net/
2. *gdsfmt* R package: https://github.com/zhengxwen/gdsfmt
3. *SeqArray* an R/Bioconductor package: https://github.com/zhengxwen/SeqArray

7 Session Info

```
toLatex(sessionInfo())
```

- R version 3.2.0 RC (2015-04-08 r68161), x86_64-w64-mingw32
- Locale: LC_COLLATE=C, LC_CTYPE=English_United States.1252, LC_MONETARY=English_United States.1252, LC_NUMERIC=C, LC_TIME=English_United States.1252
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, utils
- Other packages: SeqArray 1.8.0, gdsfmt 1.4.0
- Loaded via a namespace (and not attached): AnnotationDbi 1.30.0, BSgenome 1.36.0, Biobase 2.28.0, BiocGenerics 0.14.0, BiocParallel 1.2.0, BiocStyle 1.6.0, Biostrings 2.36.0, DBI 0.3.1, GenomeInfoDb 1.4.0, GenomicAlignments 1.4.0, GenomicFeatures 1.20.0, GenomicRanges 1.20.0, IRanges 2.2.0, RCurl 1.95-4.5, RSQLite 1.0.0, Rsamtools 1.20.0, S4Vectors 0.6.0, VariantAnnotation 1.14.0, XML 3.98-1.1, XVector 0.8.0, biomaRt 2.24.0, bitops 1.0-6, evaluate 0.6, formatR 1.1, futile.logger 1.4, futile.options 1.0.0, highr 0.4.1, knitr 1.9, lambda.r 1.1.7, rtracklayer 1.28.0, stats4 3.2.0, stringr 0.6.2, tools 3.2.0, zlibbioc 1.14.0

References

1 1000 Genomes Project Consortium, Abecasis, G. R., Auton, A., Brooks, L. D., DePristo, M. A., Durbin, R. M., Handsaker, R. E., Kang, H. M., Marth, G. T., and McVean, G. A. An integrated map of genetic variation from 1,092 human genomes. *Nature* 491, 7422 (Nov 2012), 56–65.

2 Schadt, E. E., Linderman, M. D., Sorenson, J., Lee, L., and Nolan, G. P. Computational solutions to large-scale data management and analysis. *Nat Rev Genet* 11, 9 (Sep 2010), 647–657.

3 Danecek, P., Auton, A., Abecasis, G., Albers, C. A., Banks, E., DePristo, M. A., Handsaker, R. E., Lunter, G., Marth, G. T., Sherry, S. T., McVean, G., Durbin, R., and 1000 Genomes Project Analysis Group. The variant call format and vcftools. *Bioinformatics* 27, 15 (Aug 2011), 2156–2158.

4 Zheng X, Levine D, Shen J, Gogarten SM, Laurie C, Weir BS. A High-performance Computing Toolset for Relatedness and Principal Component Analysis of SNP Data. *Bioinformatics*. 2012 Oct 11.
5 Yang J, Benyamin B, McEvoy BP, Gordon S, Henders AK, Nyholt DR, Madden PA, Heath AC, Martin NG, Montgomery GW, Goddard ME, Visscher PM. 2010. Common SNPs explain a large proportion of the heritability for human height. Nat Genet. 42(7):565-9. Epub 2010 Jun 20.