A High-performance Computing Toolset for Big Data Analysis of Genome-Wide Variants

Xiuwen Zheng

Feb 17, 2013

Contents

1 Overview

In this big data ear, thousands of gigabyte-size data sets are challenging scientists for data management, even on well-equipped hardware. R is one of the most popular statistical programming environments, but it is not typically optimized for high-performance computing necessary for large-scale genome-wide data analyses. Here I introduce a high-performance

C/C++ computing library "CoreArray" [1] for analyses of big-data genome-wide variants. This allows for development of portable and scalable storage technologies, and parallel computing at the multicore and cluster levels. I focus on the application of CoreArray for statisticians working in the R environment. Three R packages gdsfmt, SNPRelate and SeqArray are presented to address or reduce the computational burden associated with the genome-wide association studies.

Gdsfmt provides an R interface for CoreArray that works well generally compared to ncdf (v1.6) and rhdf5 (v2.0). The benchmarks show uniprocessor implementations of PCA and IBD calculation (defined below) in SNPRelate are ∼10 to 45 times faster than the implementations provided in the popular EIGENSTRAT (v3.0) and PLINK (v1.07) programs, respectively, and can be sped up to 70 ∼ 250 fold by utilizing eight cores [1].

SeqArray is designed for data management of sequencing variants, which utilizes the efficient data storage technique and parallel implementation of CoreArray. 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 an atomic data type. The file size can be further reduced to 1.3G by compression algorithms without sacrificing access efficiency, since it has a large proportion of rare variants. The uniprocessor benchmark shows that calculating allele frequencies could be done in 5 minutes with the compressed data.

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

2 Background

Today thousands of gigabyte-size data sets are challenging scientists in the management of big data, diverse types of data, and complex data relationships even on well-equipped hardware. In information technology, "big data" usually refers to a collection of data sets so large and complex that it becomes difficult to process them using existing database management tools or traditional data processing applications [2].

Genome-wide association studies (GWAS) have been widely used to investigate the genetic basis of many complex diseases and traits, but the large volumes of data generated from thousands of study samples and millions of genetic variants pose significant computational challenges. In the last ten years, chip-based genotyping technologies, such as the Illumina

1M BeadChip and the Affymetrix 6.0 chop, allow hundreds of thousands of common variants (SNPs) across the whole genome to be scored simultaneously. The Gene, Environment Association Studies Consortium (GENEVA) has generated genotypic data using chip-based genotyping techniques, with a large number of research participants ($n > 80,000$) from 14 independently designed studies of various phenotypes [3]. Currently, the field of population genetics is moving from chip data to sequencing data. Next-generation sequencing techniques are being adopted to investigate common and rare variants, making the analyses of large-scale genotypic data even more 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 [4]. In the near future, new technologies, like third-generation whole-genome sequencing [5], will be enabling data to be generated at an unprecedented scale [6]. The computational burden associated with GWAS is especially evident with large sample and variant sizes, and it requires efficient numerical implementation and memory management.

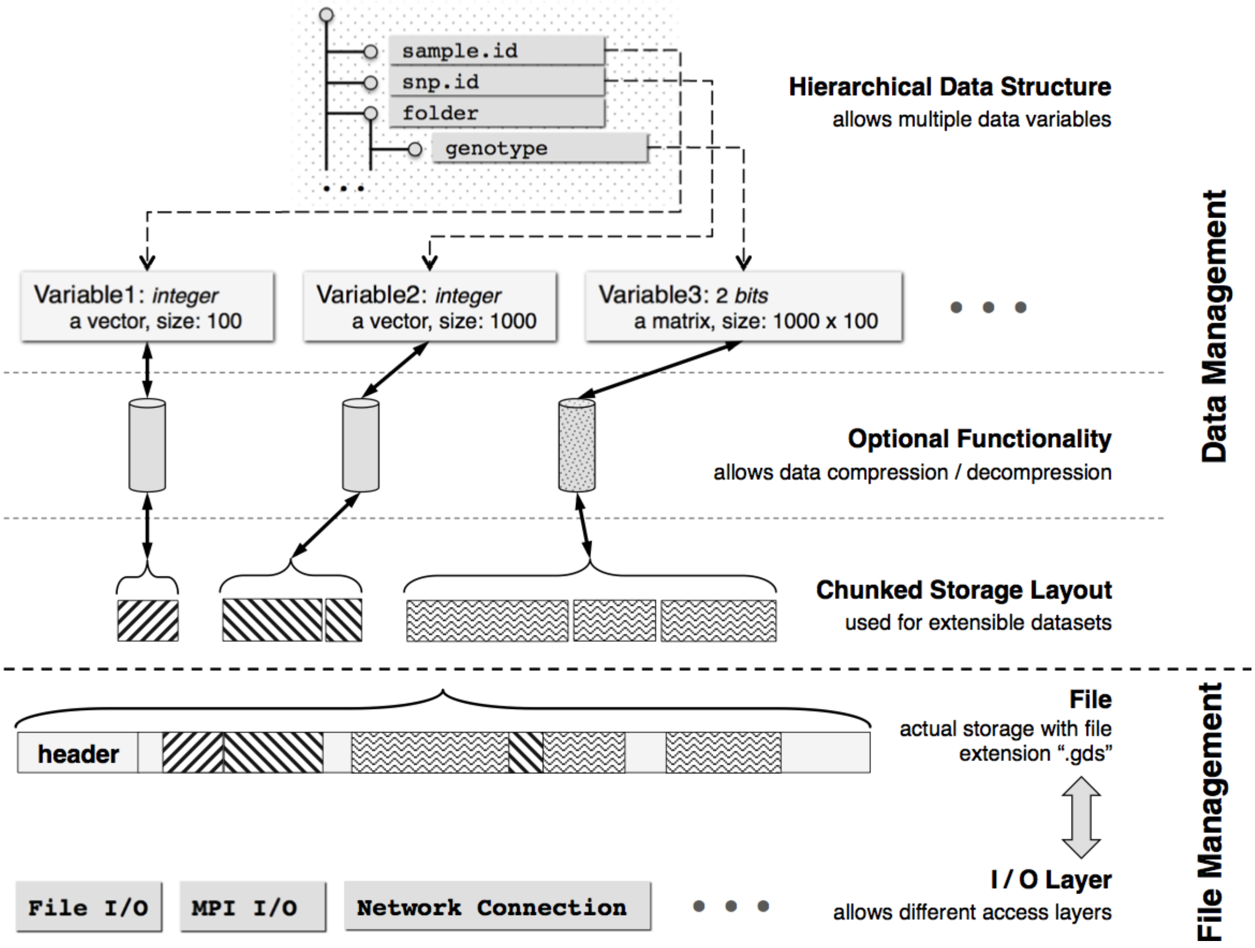
In this study, the development and application of non-commercial solutions to large-scale GWAS in the big-data era are the focus, although companies such like Google, Microsoft and Amazon have long had mastery of petabyte-scale data sets. The Network Common Data Form (netCDF) and Hierarchical Data Format (HDF) are both popular libraries, designed to store and organize large amounts of numerical data, and they are supported by non-profit research groups. Both libraries were originally written in C, but they also provided interfaces to Fortran, C++ and Java. The netCDF project is hosted by the Unidata program at the University Corporation for Atmospheric Research (http://www.unidata.ucar.edu/software/netcdf/), and HDF was originally developed at the National Center for Supercomputing Applications and it is currently supported by the non-profit HDF Group (http://www.hdfgroup.org). The latest versions of netCDF and HDF are netCDF-4 and HDF5. Actually, netCDF-4 is based on the kernel of HDF5 and provides a simple high-level application programming interface (API) for HDF5, and it is much more flexible than netCDF-3. The underlying kernel of netCDF-3 is totally different from netCDF-4. For purposes of comparison, the performance of netCDF-3 is also demonstrated in this study.

R is one of the most popular statistical programming environments, but it is not typically optimized for high performance or parallel computing which would ease the burden of large-scale GWAS calculations. Direct support of parallel computing in R started with release 2.14.0 (Dec, 2011) including a new package "parallel" shipped with the main program. For out-of-memory data, two existing R packages "ff" and "bigmemory" offer file-based access to data sets that are too large to be loaded into memory, along with a number of higher-level functions. However, unlike netCDF and HDF, these two packages do not provide sufficient functions for data management, nor a universal data format to store multiple datasets in a single file.

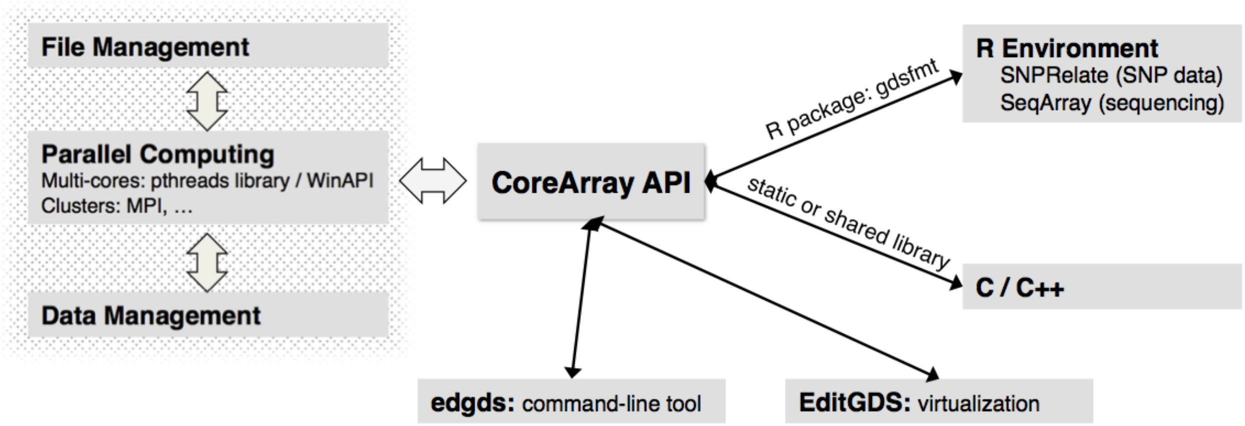
HDF5 is a powerful developer tool for big-data problems and it has been applied to the fields ofAstronomy, Biological and Biomedical Science, Environmental Science and Engineering, etc (http://www.ncsa.illinois.edu/Projects/). It is popular in the C/C++, Fortran and Java communities. Currently, an R package allowing limited HDF5 functions is "rhdf5". SNP data have a special data format, i.e., there are at most 4 cases at a biallelic site which can be stored by 2 bits instead of one byte. There are only 4 single nucleotide A, G, C or T, so it is possible to use less than 8 bits to store that information. In addition, whole-genome sequencing data are not likely to be extremely polymorphic, i.e., there are large proportions of rare variants. Hence the information on variants could be highly compressed, reducing file size and increasing access efficiency of data. HDF5 supports bit-type data via the n-bit filter, but the current version of rhdf5 (v2.0.2) does not provide those functions.

To overcome these limitations and embrace the age of big data, in 2007 I initiated a project named CoreArray (http://corearray.sourceforge.net/, hosted by SourceForge that acts as a centralized location for software developers to control and manage free and open source software development). CoreArray was designed for developing portable and scalable storage technologies for bioinformatics data, allowing parallel computing at the multicore and cluster levels. The CoreArray kernel was written in C/C++, but its application is not limited to the C/C++ language. The CoreArray project 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) and (b). 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. Users can use the CoreArray application programming interface (API) to conduct the functions of file management, data management and parallel computing. The application of CoreArray includes R packages, virtualization and command-line tools.

In this study, I focus on the application of CoreArray for statisticians working in the R environment but with limited C programming experience. Here, I provide an efficient R interface "gdsfmt" for creating and access of array-based data. Compared to other R



(a) Data modules.



(b) Application modules (API – Application Programming Interface).

Figure 1: CoreArray library modules.

interfaces (“ncdf” to netCDF-3 and “rhdf5” to HDF5), gdsfmt works well generally, and even outperforms “ncdf” on the test datasets used in this study.

2.1 SNP Data in Genome-wide Association Studies

PLINK, an open-source C/C++ tool, was developed to address the computational challenges for whole-genome association and population-based linkage analyses. Since a biallelic SNP site has at most two different alleles, constituting three possible genotypes with an additional state to indicate missing data. This information can be packed into two bits instead of using one byte, and PLINK reduces the file size by packing 4 SNP genotypes into one byte. The main limitation of PLINK is memory because PLINK has to load all SNP genotypes to memory. Therefore, the GDS format provides a big-data solution to storing SNP genotypes for GWAS, by alowing access to data as needed without loading all data to memory.

Overall, two R packages have been presented to address some of computational challenges in GWAS: “gdsfmt” to provide efficient, platform-independent memory and file management for genome-wide numerical data, and “SNPRelate” to solve large-scale, numerically-intensive GWAS calculations (i.e., PCA and IBD, see below) on multi-core symmetric multi-processing (SMP) computer architectures [1]. Future development based on the GDS format is allowed, and users could exploit the parallel computing functions in the gdsfmt package with the R package “parallel” to speed up the analyses.

Principal component analysis (PCA) has been proposed to detect and correct for population structure in genetic association studies. The eigen-analysis has been implemented in the software package “EIGENSTRAT” but the computational burden is evident for large scale GWAS SNP data with several thousand study individuals. Parallel computing was formally supported by EIGENSTRAT v4.0, but it still required keeping all data in memory. The PCA functions in the R package SNPRelate allow much larger datasets than does EIGENSTART: the kernel of SNPRelate was written in C and has been highly optimized and it runs faster than EIGENSTART.

Identity-by-descent (IBD) methods have also used to correct for population structure and cryptic relatedness by estimating the degree of relatedness between each pair of study samples [7]. Maximum-likelihood estimation (MLE) was first proposed by Thompson (1975) [8] to estimate three IBD coefficients in a non-inbred population by a “hill climbing” technique. An expectation–maximization (EM) algorithm was proposed by Choi et al. (2009) [7] to estimate IBD coefficients but this is very time-consuming and not suitable for large-scale data. An alternative is the method of moments (MoM) approach provided by PLINK based on identity-by-state. Compared to MLE, MoM has a great advantage in computational efficiency. The R packages “CrypticIBDcheck” and “ibdreg”, and many other binary software written in other languages, involved with IBD coefficients, have a limitation in data scale. By contrast, gdsfmt and SNPRelate provide an efficient data storage technique and a parallel implementation to address and reduce the burden of IBD calculation. Since big-data analyses are the focus of this study, requiring computationally efficient methods, the performance of different implementations of MoM, rather than MLE, are compared here.

2.2 Sequencing Variants

The Variant Call Format (VCF) was developed for the 1000 Genomes Project. It is a generic text format for storing DNA polymorphism data such as SNPs, insertions, deletions and structural variants, together with rich annotations (http://vcftools.sourceforge.net) [9]. It is most likely stored in a compressed manner with indices for fast data retrieval of variants. A less-flexible binary format (Binary Call Format, BCF) is designed for efficient storing and parsing of VCF records. PLINK/SEQ, a toolset for working with sequencing data, is designed to be complementary to the existing PLINK package (http://atgu.mgh.harvard.edu/plinkseq/). PLINK/SEQ was written in C/C++, but it also provides an R interface “Rplinkseq” with limited functions. An R/Bioconductor package “VariantAnnotation” was designed for annotating and filtering genetic variants using VCF files, but is not used for data analysis.

Most of the existing software and packages for the analyses of sequencing data are not designed for R users, and it is thought that the implementation of the R language is slow and not likely to be suitable for big-data analyses. To address this limitation I developed an R package “SeqArray” that utilizes the efficient data storage technique and parallel implementation of CoreArray. SeqArray provides an alternative data storage to VCF for exploitig genetic variant data, and the kernel of SeqArray is written in C/C++ to speed up the intensive computation for large-scale sequencing data. The primary functions in SeqArray are related to data management, offering efficient access of genetic variants using the R language. It is a possible solution to make up the gap in data analyses between R users and high-throughput sequencing data. R users can use their own packages to extend the functions and computational efficiency of SeqArray.

Table 1: Data types supported by the CoreArray library.

signed integer	
	8-bit integer, 16-bit integer, 24-bit integer, 32-bit integer, 64-bit integer signed integer with 2 bits, signed integer with 3 bits, ..., signed integer with 15 bits
unsigned integer	
	8-bit integer, 16-bit integer, 24-bit integer, 32-bit integer, 64-bit integer unsigned integer with 1 bit, unsigned integer with 2 bits, ..., unsigned integer with 15 bits
floating-point number	
	single-precision number (32 bits), double-precision number (64 bits)
character	
	UTF-8 string, UTF-16 string, UTF-32 string

3 Features

The following features are described in this section: CoreArray modules, the GDS structure for SNP data, and the GDS structure for sequencing data.

3.1 Features of CoreArray

CoreArray is the project name that includes the C/C++ kernel library and external applications. Multiple data variables can be stored in a single file with the universal data format for genomic data structure (GDS). As shown in Table 1, the data types are not limited to array-oriented data, and CoreArray also supports storing any file, such as a text file describing project information. Data variables are organized in a hierarchical structure allowing folders to contain different variables. The variable dimension can be extended from any direction. The CoreArray library supports a single file with size of at most 128 terabytes (2^{47} bytes).

The algorithm modules are shown in Figure 1a, and the detailed documents for C/C++ source codes are provided at: `http://corearray.sourceforge.net/lib/html/index.html`. Instead of going through the programming details, I provide a higher-level description of CoreArray's functionality here, and an overview of the inheritance diagram for CoreArray object classes on the webpage (`http://corearray.sourceforge.net/lib/html/class_core_array_1_1_cd_object.html`). For data management, an optional data compression / decompression function can be plugged in. The standard deflate and inflate algorithm, zlib (http://www.zlib.net), is used currently by the CoreArray kernel. A chunked storage layout is adopted in the low-level storage management for extensible datasets, and a contiguous data space may be divided into two or more chunks stored in the actual file without necessarily being adjacent. For example, the user adds 1000 integers to a data variable when a GDS file is created, and then he would like to append another 1000 integers to the variable. However, if the original chunk has no enough space for the new integers, then GDS format will automatically create a new chunk to store the additional data. For file management, CoreArray allows different access I/O layers, such as standard file I/O, MPI I/O (Message Passing Interface, MPI, for parallel computing by multiple processes), etc.

In Unix-like systems, the standard pthreads library is adopted for parallel computing, whereas Windows systems do not provide pthreads by default, instead WinAPIs are called to substitute the functions in pthreads. CoreArray offers a universal platform-independent interface of multi-thread functions. The binary program "EditGDS" was designed for virtualization of GDS format. Users can use EditGDS to open a GDS file immediately, and its tree-like structure is automatically displayed in the left panel. Browsing and modifying datasets manually are allowed in EditGDS. EditGDS was written in Free Pascal language (http://www.freepascal.org) using Lazarus (a free development environment). Its source code can be downloaded from http://sourceforge.net/projects/corearray/. A command-line tool "edgds" is shipped with the main C/C++ source codes, allowing multiple operations on a GDS file, such as extracting a subset of the data variables.

3.2 Features of SNPRelate for SNP Data

3.2.1 Data Structure for SNPRelate

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. This is a container for storing annotation data and SNP genotypes. In this format each byte encodes up to four SNP genotypes, thereby reducing file size and access time. The GDS format supports data blocking so that only the subset of data that is being processed needs to reside in memory. GDS formatted data is also designed for efficient random access to large data sets. Although SNPRelate functions operate only on GDS-format data files, functions to reformat data from PLINK [10], sequencing Variant Call Format (VCF) [9], netCDF [11] and other data files, are provided by my packages.

```
> # load the R packages: gdsfmt and SNPRelate
> library(gdsfmt)
> library(SNPRelate)
```

Here is a typical GDS file:

```
> snpgdsSummary(snpgdsExampleFileName())
```

```
The total number of samples: 279
The total number of SNPs: 9088
SNP genotypes are stored in individual-major mode.
```

snpgdsExampleFileName() returns the file name of a GDS file used as an example in SNPRelate, and it is a subset of data from the HapMap project and the samples were genotyped by the Center for Inherited Disease Research (CIDR) at Johns Hopkins University and the Broad Institute of MIT and Harvard University (Broad). **snpgdsSummary()** summarizes the genotypes stored in the GDS file. "Individual-major mode" indicates listing all SNPs for an individual before listing the SNPs for the next individual, etc. Conversely, "SNP-major mode" indicates listing all individuals for the first SNP before listing all individuals for the second SNP, etc. Sometimes "SNP-major mode" is more computationally efficient than "individual-major model". For example, the calculation of the genetic covariance matrix deals with genotypic data SNP by SNP, and then "SNP-major mode" should be more efficient.

```
> # open a GDS file
> (genofile <- openfn.gds(snpgdsExampleFileName()))
```

```
file name: Users/ZhengX/extdata/hapmap_geno.gds

+         [   ]
|--+ sample.id        [ FStr8 279 ZIP(23.10%) ]
|--+ snp.id        [ Int32 9088 ZIP(34.76%) ]
|--+ snp.rs.id        [ FStr8 9088 ZIP(42.66%) ]
|--+ snp.position        [ Int32 9088 ZIP(51.77%) ]
|--+ snp.chromosome        [ Int32 9088 ZIP(0.33%) ]
|--+ snp.allele        [ FStr8 9088 ZIP(14.45%) ]
|--+ genotype        [ Bit2 9088x279 ] *
```

```
|--+ sample.annot        [   ] *
|  |--+ sample.id        [ FStr8 279 ZIP(23.10%) ]
|  |--+ family.id        [ FStr8 279 ZIP(28.37%) ]
|  |--+ geneva.id        [ Int32 279 ZIP(80.29%) ]
|  |--+ father.id        [ FStr8 279 ZIP(12.98%) ]
|  |--+ mother.id        [ FStr8 279 ZIP(12.86%) ]
|  |--+ plate.id        [ FStr8 279 ZIP(1.29%) ]
|  |--+ sex        [ FStr8 279 ZIP(28.32%) ]
|  |--+ pop.group        [ FStr8 279 ZIP(7.89%) ]
```

The output lists all variables stored in the GDS file. At the first level, it stores variables **sample.id**, **snp.id**, etc. The additional information are displayed in the square brackets indicating data type, size, compressed or not + compression ratio. The second-level variables **sex** and **pop.group** are both stored in the folder of **sample.annot**. All of the functions in SNPRelate require a minimum set of variables in the SNP annotation data. The minimum required variables are

- **sample.id**, a unique identifier for each sample.
- **snp.id**, a unique identifier for each SNP.
- **snp.position**, the base position of each SNP on the chromosome, and 0 for unknown position; it does not allow NA.
- **snp.chromosome**, an integer mapping for each chromosome, with values 1-26, mapped in order from 1-22, 23=X,24=XY (the pseudoautosomal region), 25=Y, 26=M (the mitochondrial probes), and 0 for probes with unknown positions; it does not allow NA.
- **genotype**, a SNP genotypic matrix. SNP-major mode: $n_{sample} \times n_{snp}$, individual-major mode: $n_{snp} \times n_{sample}$.

```
> # Take out snp.id
> head(read.gdsn(index.gdsn(genofile, "snp.id")))
[1] 1 2 3 4 5 6

> # Take out snp.rs.id
> head(read.gdsn(index.gdsn(genofile, "snp.rs.id")))
```

```
[1] "rs1695824"  "rs13328662" "rs4654497"  "rs10915489" "rs12132314"
[6] "rs12042555"
```

There are two additional variables:

- **snp.rs.id**, a character string for reference SNP ID that may not be unique.
- **snp.allele**, it is not necessary for the analysis, but it is necessary when merging genotypes from different platforms. The format of **snp.allele** is "A allele/B allele", like "T/G" where T is A allele and G is B allele.

There are four possible values stored in the variable **genotype**: 0, 1, 2 and 3. For bi-allelic SNP sites, "0" indicates two B alleles, "1" indicates one A allele and one B allele, "2" indicates two A alleles, and "3" is a missing genotype. For multi-allelic sites, it is a count of the reference allele (3 meaning no call). "Bit2" indicates that each byte encodes up to four SNP genotypes since one byte consists of eight bits. "FStr8" indicates a character-type variable.

```
> # Take out genotype data for the first 3 samples and the first 5 SNPs
> (g <- read.gdsn(index.gdsn(genofile, "genotype"), start=c(1,1), count=c(5,3)))

     [,1] [,2] [,3]
[1,]    2    1    2
[2,]    1    1    1
[3,]    0    0    1
[4,]    1    1    2
[5,]    2    2    2

> # read population information
> pop <- read.gdsn(index.gdsn(genofile, c("sample.annot", "pop.group")))
> table(pop)
pop
CEU HCB JPT YRI 
 92  47  47  93 

> # close the GDS file
> closefn.gds(genofile)
```

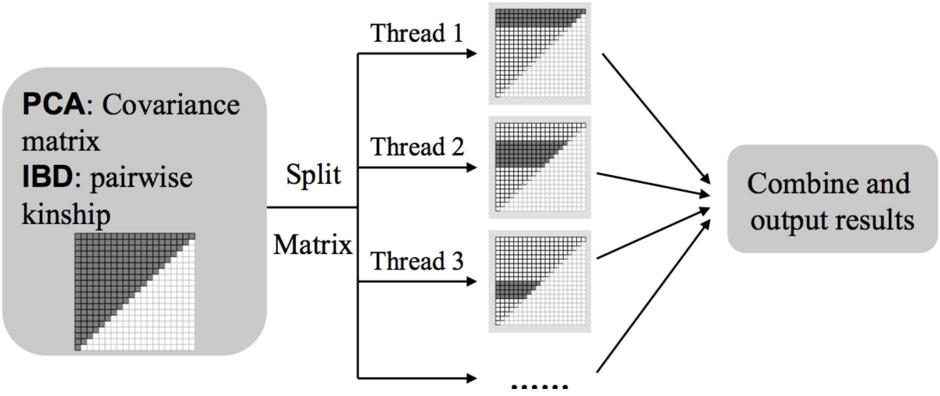


Figure 2: Flowchart of parallel computing for principal component analysis and identity-by-descent analysis.

3.2.2 Functions of SNPRelate

SNPRelate provides computationally efficient functions for PCA and IBD relatedness analysis on GDS genotype files. The calculations of the genetic covariance matrix and pairwise IBD coefficients are split into non-overlapping parts and assigned to multiple cores for performance acceleration, as shown in Figure 2. The functions in SNPRelate for PCA include the basic calculations of sample and SNP eigenvectors, as well as useful accessory functions. The correlation between sample eigenvectors and observed allelic dosage can be used to evaluate the genome-wide distribution of SNP effects on each eigenvector. The SNP eigenvectors can be used to project a new set of samples to the existing axises, which is useful in studies with substantial relatedness [12].

For relatedness analysis, IBD estimation in SNPRelate can be done by either the method of moments (MoM) [10] or maximum likelihood estimation (MLE) [7, 8] through identity by state (IBS). Our experience shows that MLE is significantly more computationally intensive than MoM for large-scale data analysis, although MLE estimates are usually more reliable than MoM. Additionally, the functions for linkage disequilibrium (LD) pruning generate a pruned subset of SNPs that are in approximate linkage equilibrium with each other, to avoid the strong influence of SNP clusters in PCA and IBD analysis. An actual kinship matrix of individuals can be estimated by either method, which could be used in downstream association analyses [13].

Both R packages are written in C/C++, use the POSIX threads library for shared memory parallel computing on Unix-like systems, and have an R interface in which the kernel has been highly optimized by blocking the computations to exploit the high-speed cache memory. The algorithms are optimized to load genotypes block by block with no limit to the number of SNPs. The algorithms are limited only by the size of the main memory, which is accessed by the parallel threads, and holds either the genetic covariance matrix or IBD coefficient matrix.

GDS is also used by an R/Bioconductor package GWASTools as one of its data storage formats [14]. GWASTools provides many functions for quality control and analysis of GWAS, including statistics by SNP or scan, batch quality, chromosome anomalies, association tests, etc.

3.3 Features of SeqArray for Sequencing Data

A GDS format for storing DNA polymorphism data such as SNPs, insertions, deletions and structural variants, together with rich annotations, including haplotypic phase states, was provided by the R package "SeqArray". A typical GDS for sequencing data with minimal variables is as follows:

```
+
|--+ description      [ ]           <-- indicates sequencing format
|--+ sample.id        [ VStr8, n ]  <-- unique IDs
|--+ variant.id       [ Int32, m ]  <-- unique IDs
|--+ position         [ Int32, m ]  <-- position of the start of the variant
|--+ chromosome       [ VStr8, m ]  <-- chromosome code
|--+ allele           [ VStr8, m ]  <-- reference and alternative alleles
|--+ genotype
|  |--+ length        [ Int32, m ]  <-- # of bits needed for each variant over 2
|  |--+ data     [ Bit2, (2, n, m1) ]  <-- stores multiple genetic variants
|  |--+ ~data    [ Bit2, (2, m1, n) ]  <-- the transposed "data", optional
|--+ phase
|  |--+ data     [ Bit1, (n, m) ]      <-- 1/0, whether phased or unphased
|  |--+ ~data    [ Bit1, (m, n) ]      <-- the transposed "data", optional
```

where n is the number of samples, m is the total number of variants for DNA polymorphism, (2, n, m1) is a 3-dimensional array, and (n, m) is a matrix, where $\mathtt{m1} \geq \mathtt{m}$. "VStr8" represents variable-length string, whereas "Int32" for 32-bit integer, "Bit2" for 2-bit integer and "Bit1" for 1-bit integer (0/1). The variables **sample.id**, **variant.id**, **position**, **chromosome** and **allele** are not necessarily of data type shown here (VStr8, character; Int32, 32-bit integer). The chromosome code supports "X", "XY", etc, or "Z" for other species. The variable **data** stores multiple genotypic variants in a single 3-dimensional dataset. The size of the first dimension is two, since human genomes consist of pairs of chromosomes. For other polyploid species, the size of the first dimension could be greater than two to reflect actual number of copies of chromosomes. The type of **genotype/data** is "Bit2" allowing at most four possible values, and it is sufficient to represent most of genetic variants since SNPs are the most common polymorphism (two alleles plus a missing flag, three possible values in total). If a site has more than three possible polymorphisms (like multiple alleles, insertion or deletion), contiguous space will be automatically used to store additional polymorphic information. For example, a site with seven polymorphisms, an allele with eight possible values (seven alleles plus a missing flag) cannot be stored in two bits, then SeqArray will utilize contiguous two bits in the next sub data space to represent this value, i.e., a total of four bits can represent at most 16 classes. Therefore, the third dimension of **genotype/data** `m1` could be greater than `m`. The variable **genotype/length** provides information for how many bits needed for each variant. Finally, **phase** indicates the phasing states have been determined or not by sequencing methods. The chip-based genotyping techniques cannot determine phases or haplotypes, the next-generation sequencing partially offers phasing information but that information is limited in a small DNA fragment. The prefix ∼ indicates that it is a transposed version of corresponding variable, which helps for optimizing the access efficiency.

Here I describe only the key features of SeqArray. Additional annotation information, such as quality score, are also able to be stored in the GDS file. Any future extension of SeqArray will depend on real problems, and some coding optimization needs to be made by C programming. Comprehensive R analyses using SeqArray will be provided by other packages from my future research.

4 Performances

A benchmark scheme was adopted to make a comparison among three R packages "gdsfmt", "ncdf" and "rhdf5". The array 32-bit integer variables used in the performance comparison are shown in Table 2 with dimensions ranging from one to six, which has been used by the HDF Group for a netCDF-4 performance report [1]. The small sets are approximately 1MB,

[1] "NetCDF-4 Performance Report – The HDF Group" at http://www.hdfgroup.org/pubs/papers/2008-06_netcdf4_perf_report.pdf

Table 2: Array variables of 32-bit integer for performance comparisons in R.

Variable Name	Number of Dimensions	Dimension	
		Small Tests	Large Tests
Variable 1	1	[262144]	[10000000]
Variable 2	2	[512][512]	[3162][3162]
Variable 3	3	[64][64][64]	[215][215][215]
Variable 4	4	[22][22][22][22]	[56][56][56][56]
Variable 5	5	[12][12][12][12][12]	[25][25][25][25][25]
Variable 6	6	[8][8][8][8][8][8]	[14][14][14][14][14][14]
Data Size		$\sim$ 1MB	$\sim$ 30 to 40 MB

while the sizes of large tests range from 30MB to 40MB. Each benchmark run measured the time to read or write a single variable, and the execution sequence was:

```
Repeat 100 times:
    Test 1: Open File1; write Variable 1; close File1
        [ call “drop caches” if cache is disabled ]
    Test 2: Open File1; read Variable 1; close File1
        [ call “drop caches” if cache is disabled ]
    Test 3: Open File2; write Variable 2; close File2
        [ call “drop caches” if cache is disabled ]
    Test 4: Open File2; read Variable 2; close File2
        [ call “drop caches” if cache is disabled ]
    ...
    Test 11: Open File6; write Variable 6; close File6
        [ call “drop caches” if cache is disabled ]
    Test 12: Open File6; read Variable 6; close File6
        [ call “drop caches” if cache is disabled ]
```

The entire benchmark consists of 100 runs, and the read and write speeds are estimated by averaging 100 speeds of each single run. The benchmark uses the command **system.time** in R to bracket the read and write functions. The read and write speeds are reported in Figure 3, calculated by the size of dataset in megabytes over the elapsed wall clock time for the corresponding calls. The latest versions of R packages, gdsfmt (v0.9.10), ncdf (v1.6) and rhdf5 (v2.0.2), were used in the tests. The benchmarks were run on a Linux system with two quad-core Intel processors (2.27GHz) and 32 GB RAM. The system kernel caches programs and data in memory as long as possible, so sometimes read and write rates actually reflect the time to access memory rather than disk. In practice the system caching is almost always enabled, but the file size may be out of the range of cache memory. Therefore, the rates with

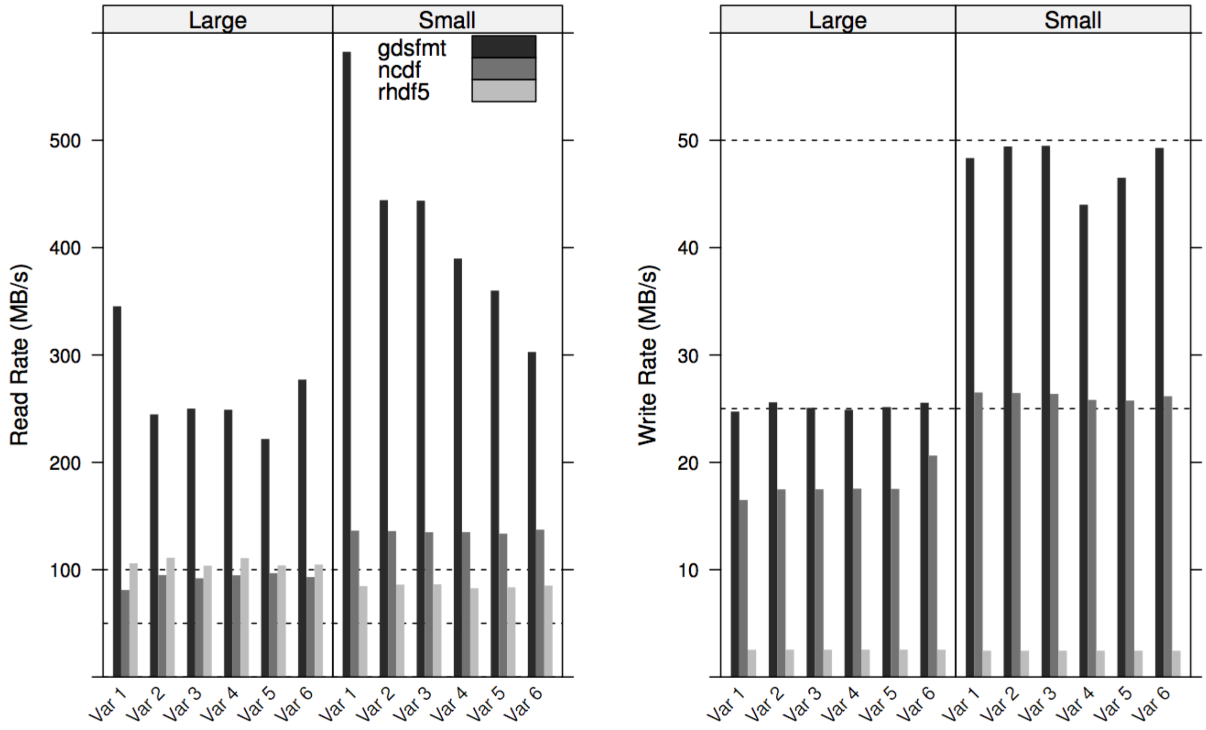


Figure 3: The benchmarks for reading and writing data using gdsfmt, ncdf and rhdf5 when the system cache is enabled. Gdsfmt outperforms the other two packages.

and without memory cache are both investigated in this study.

Figure 3 shows that gdsfmt outperforms the other packages on reading and writing data for every single run when the system cache is enabled. On average, the read rate of gdsfmt is about 3 times of those of ncdf and rhdf5, and the write rate is ∼ 1.6 times compared to ncdf, and ∼ 14 times faster than rhdf5. Reading and writing on small datasets are slower than the same operations on large datasets. The read rates of gdsfmt tend to decline as the number of dimensions, whereas the read rates of ncdf and rhdf5 and write rates do not have such trend. When the system cache is cleared (Figure 4), the read speed is about twice of the write rate on large datasets for gdsfmt and ncdf packages. Note that, on average, the read rate using system cache is about 5 times of that without system cache. Overall, gdsfmt performs well compared to ncdf and rhdf5.

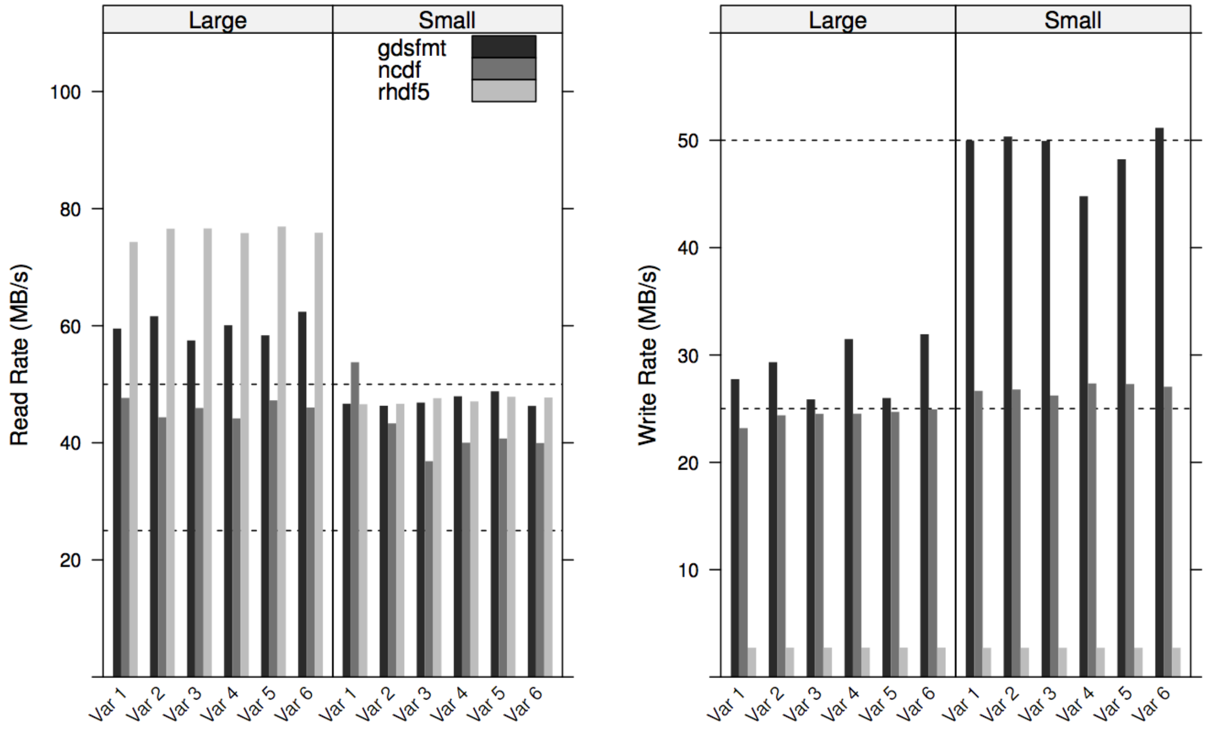


Figure 4: The benchmarks for reading and writing data using gdsfmt, ncdf and rhdf5 when the system cache is cleared. Gdsfmt is more efficient than the other two packages on writing data, whereas rhdf5 is the most efficient on reading data.

4.1 Comparison with PLINK and EIGENSTRAT

We illustrate the performance of SNPRelate using small, medium and large test data sets. The small and medium sets were constructed from simulated data and contain 500 and 5,000 samples with 100K SNP markers, respectively. The large set consists of 55,324 subjects selected from 16 projects of the "Gene-Environment Association Studies" (GENEVA) consortium [3]. We compared the run times of SNPRelate with EIGENSTRAT (v3.0) and PLINK (v1.07) for PCA and IBD estimation respectively. The implementations were benchmarked on a system with two quad-core Intel processors running at 2.27GHz and 32 GB RAM and running Linux Fedora 10.

As shown in Table 3, the uniprocessor implementations of PCA and IBD in SNPRelate are approximately eight to 50 times faster than the implementations provided in EIGENSTRAT and PLINK respectively. When the SNPRelate algorithms were run using eight cores, the

Table 3: Comparison of run-times (seconds and minutes) for SNPRelate, EIGENSTRAT and PLINK on a Linux system with two quad-core Intel processors (2.27GHz) and 32 GB RAM.

Method /	Small Set[1]			Medium Set[1]		
# of cores	1	4	8	1	4	8
Principal Component Analysis (PCA)						
SNPRelate {	11s+ 1s[2]	5s+ 1s[2]	3s+ 1s[2]	20m+ 12m[2]	8m+ 12m[2]	5m+ 12m[2]
EIGENSTRAT	90s[3]	—	—	710m[3]	—	—
Method of Moment for Identity-by-Descent Analysis (MoM)						
SNPRelate	19s	6s	4s	30m	8m	5m
PLINK	980s	—	—	1630m	—	—

[1]: simulated 500 (small set) and 5000 (medium set) samples with 500K SNPs;
[2]: calls the uniprocessor version of LAPACK in R to compute the eigenvalues and eigenvectors, taking 1s and 12m for the small and medium set respectively;
[3]: includes the computation time of calculating the eigenvalues and eigenvectors.

performance improvement ranged from ∼30 to ∼300. The SNPRelate PCA was conducted on the large data set ($n = 55,324$ subjects with ∼310K selected SNP markers). It took ∼64 hours to compute the genetic covariance matrix (55K-by-55K) when eight cores were used, and ∼9 days to calculate eigenvalues and eigenvectors using the uniprocessor version of the linear algebra package (LAPACK) in R. The analyses on the small- and medium- size data sets required less than 1GB of memory, and PCA on ∼55K subjects required ∼32GB since the genetic covariance matrix is stored in the main memory shared by threads. An improvement on running time for PCA is to employ a multi-threaded version of BLAS to perform the calculation of eigenvalues and eigenvectors instead of the default uniprocessor one. Although SNPRelate is much faster than EIGENSTRAT for PCA or PLINK for IBD estimation using MoM, the results are numerically the same (i.e. identical accuracy).

4.2 Performance for Sequencing Variant Data

Currently, the primary application of CoreArray is in the field of bioinformatics. There are only four single nucleotide A, G, C or T, and at most three possible SNP genotypes at a biallelic locus, therefore we can use less than eight bits to represent SNP or sequencing data. In this section, I compare the performance of three different storage schemes to represent genotypic data from the 1000 Genomes Project [4] using GDS format: 1) one byte represents one allele; 2) four SNP alleles are packed in one byte; 3) four SNP alleles are packed in one byte + data compression. Phasing information have be incorporated in the GDS files, since two alleles at a site are stored separately.

The test 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/`. The function "seqVCF2GDS" in the R package SeqArray was used to convert and merge all VCF files, and all ∼39M variants are extracted from the VCF files. I prepared six GDS files ahead which store genotypic data using the above schemes: one byte, two bits and two bits plus compression for sample-by-variant and variant-by-sample storages respectively. "Sample-by-variant" indicates listing all individuals for the first variant before listing all individuals for the second variant etc, whereas "variant-by-sample" indicates listing all variants first.

A benchmark scheme was adopted to evaluate the performance of CoreArray API by R programming, stratified by scan orders: SNP by SNP, or sample by sample. Since genotypes were stored in the GDS files SNP by SNP, the higher speeds of scanning SNP by SNP should be expected. The function "apply.gdsn" in the R package gdsfmt was used to perform scanning, which allows two ways by SNP or by sample. The function "clusterApply.gdsn" was called in the parallel tests for the multi-core system. The functions "apply.gdsn" and "clusterApply.gdsn" are actually coded in low-level C. Scanning means that passing genotypes to a special function but that function did nothing. Each test was replicated 5 times, and the average running time was reported. The benchmarks were run on a Linux system with two quad-core Intel processors (2.27GHz) and 32 GB RAM, and gdsfmt v0.9.12 and SeqArray v0.9.0 were used in the tests.

As shown in Table 4, the compression ratio for the sample-by-variant storage scheme is 5.6%, reflecting the fact that whole human genomes are unlikely to be very polymorphic, since the dataset consists of large proportions of variants. According to the variant-by-sample scheme, the compression ratio is 19.2%, which indicates it is less efficient to compress genetic data for a single individual.

The read rates are presented as the running times for scanning the whole dataset variant by variant, or sample by sample. When the reading order agrees with the storage order (i.e., reading by variant according to storing by variant), it is significantly efficient than the case of disagreement. E.g., it only took 4.2 minutes to read the whole data variant by variant, when genotypic data are stored by variant. The storage scheme of "two bits" is the optimal way to store genotypic data. For scanning by variant, the read speed for "two bits" with and without

Table 4: The computing times of CoreArray when reading genotypic GDS files, which consist of 39,706,715 variants and 1,092 study individuals from the 1000 Genomes Project [1].

		Storage Scheme		
Running time (minute / hour)		one byte (∼ 86G)	2 bits (∼ 26G)	2 bits + data compression[1]
Store variant by variant:				∼ 1.3G (5.2%)
read by variant	1 core	20.4m	5.4m	4.2m
	4 cores	12.3m	1.4m	1.1m
read by sample[2]	1 core	47.8h	56.7m	234.2m
	4 cores	39.6h	14.4m	58.6m
Store sample by sample:				∼ 4.8G (19.2%)
read by variant[2]	1 core	98.9m	44.4m	72.8m
	4 cores	36.8m	13.8m	18.9m
read by sample	1 core	28.8m	6.8m	12.6m
	4 cores	16.6m	1.7m	3.2m

[1]: standard "zlib" was used with default settings, compression ratios are 5.2% and 19.2%.
[2]: using the default buffer size 1G. Use of large buffer size can reduce the times of scanning whole dataset, but it should not be out of memory limit.

compression are much faster than "one byte" (∼ four times). It can be explained as the size of two-bit GDS file (∼ 26G and 1.3G) can be cached in system memory (32G), but the GDS file for "one byte" (∼ 86G) had exceeded the memory limit. CoreArray kernel has to refresh file caches by loading more data from the hard disk (please compare the performance of gdsfmt with and without caches in Figure 3 and 4). The read bottleneck also influenced the performance of parallel computing, and the ratio for "one byte" (20.4m/12.3m) is far away from the factor four, compared to the ratios for "two bits" (5.4m/1.4m) and "two bits plus compression" (4.2m/1.1m). When reading data sample by sample according to the sample-by-variant storage scheme, CoreArray kernel automatically adopts a buffer strategy since the variants of a specified sample are not stored contiguously. The corresponding running times are significantly slower than the times of reading by variant in the same column. It is interesting to see how the memory factor influence the running times: I reran all tests on a workstation with 96G memory, the most extreme reading time reduced from 47.8h to 1.2h. System memory cache does have significant effects on the access efficiency.

When the genotypic data are stored sample by sample, reading by sample is more efficient than reading by variant, e.g., 6.8m vs 44.4m, and 12.6m vs 72.8m. Its compression ratio (19.2%) is higher than that (5.2%) compared to the sample-by-variant storage scheme. Larger file size (4.8G vs 1.3G) indicates more running times (12.6m vs. 4.2m). If the file size is of greater interest, the storage scheme “two bits plus compression” plus “sample-by-variant” appears to be appropriate for normal-equipped hardware, especially for laptops with at most 8G memory. In the case of laptop, $\sim$ 26G of genotypic data are not able to be cached in memory, then scanning such dataset will require more file read operation interacting with hard disk and will be slowed down. The results in Table 4 also indicate that the functions reading the “sample-by-variant” dataset sample by sample require an optimized programming skill. For example, a typical function reading by sample is to calculate the missing rate per sample. This function can be revised as a function of reading by variant, then it could be sped up without a parallel scheme. The applications include calculating allele frequencies, and the uniprocessor benchmark shows that calculating allele frequencies could be done in 5 minutes with the compressed data.

5 Conclusion

In this study, I introduced a high-performance computing library CoreArray for big-data analyses of genome-wide variants. The CoreArray project was initiated in 2007 with an aim to develop portable and scalable storage technologies for bioinformatics data allowing parallel computing at the multicore and cluster levels. I focus on the application of CoreArray for statisticians working in the R environment but with limited C programming experience. Three R packages gdsfmt, SNPRelate and SeqArray are presented to address or reduce the computational burden associated with the genome-wide association studies.

Gdsfmt provides a general R interface of CoreArray API, and it works well generally, and it even outperforms ncdf and rhdf5 on the most of the test datasets in this study. The benchmarks show the uniprocessor implementations of PCA and IBD in SNPRelate are $\sim$10 to 45 times faster than the implementations provided in the popular EIGENSTRAT (v3.0) and PLINK (v1.07) programs respectively, and can be sped up to 70 $\sim$ 250 fold by utilizing eight cores. SeqArray offers a possible solution to make up the gap in data analyses between R users and high-throughput sequencing data utilizing parallel computing.

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

6 Resources

1. CoreArray project: http://corearray.sourceforge.net/

2. gdsfmt R package: http://cran.r-project.org/web/packages/gdsfmt/index.html

3. SNPRelate R package: http://cran.r-project.org/web/packages/SNPRelate/index.html

4. SeqArray, an R/Bioconductor package.

References

[1] X Zheng, D Levine, J Shen, S M Gogarten, C Laurie, and B S Weir. A high-performance computing toolset for relatedness and principal component analysis of SNP data. *Bioinformatics*, Oct 2012.

[2] O'Reilly Media. *Hadoop: The Definitive Guide*. ISBN 978-1-4493-3877-0. White, Tom, 2012.

[3] M C Cornelis, A Agrawal, J W Cole, N N Hansel, K C Barnes, T H Beaty, S N Bennett, L J Bierut, E Boerwinkle, K F Doheny, B Feenstra, E Feingold, M Fornage, C A Haiman, E L Harris, M G Hayes, J A Heit, F B Hu, J H Kang, C C Laurie, H Ling, T A Manolio, M L Marazita, R A Mathias, D B Mirel, J Paschall, L R Pasquale, E W Pugh, J P Rice, J Udren, R M van Dam, X Wang, J L Wiggs, K Williams, K Yu, and GENEVA Consortium. The Gene, Environment Association Studies consortium (GENEVA): maximizing the knowledge obtained from GWAS by collaboration across studies of multiple conditions. *Genet Epidemiol*, 34(4):364–372, May 2010.

[4] 1000 Genomes Project Consortium, G R Abecasis, A Auton, L D Brooks, M A DePristo, R M Durbin, R E Handsaker, H M Kang, G T Marth, and G A McVean. An integrated map of genetic variation from 1,092 human genomes. *Nature*, 491(7422):56–65, Nov 2012.

[5] J Eid, A Fehr, J Gray, K Luong, J Lyle, G Otto, P Peluso, D Rank, P Baybayan, B Bettman, A Bibillo, K Bjornson, B Chaudhuri, F Christians, R Cicero, S Clark, R Dalal, A Dewinter, J Dixon, M Foquet, A Gaertner, P Hardenbol, C Heiner, K Hester, D Holden, G Kearns, X Kong, R Kuse, Y Lacroix, S Lin, P Lundquist, C Ma, P Marks, M Maxham, D Murphy, I Park, T Pham, M Phillips, J Roy, R Sebra, G Shen, J Sorenson, A Tomaney, K Travers, M Trulson, J Vieceli, J Wegener, D Wu, A Yang, D Zaccarin, P Zhao, F Zhong, J Korlach, and S Turner. Real-time dna sequencing from single polymerase molecules. *Science*, 323(5910):133–138, Jan 2009.

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

[7] Y Choi, E M Wijsman, and B S Weir. Case-control association testing in the presence of unknown relationships. *Genet Epidemiol*, 33(8):668–678, Dec 2009.

[8] E A Thompson. The estimation of pairwise relationships. *Ann Hum Genet*, 39(2):173–188, Oct 1975.

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

[10] S Purcell, B Neale, K Todd-Brown, L Thomas, M A Ferreira, D Bender, J Maller, P Sklar, P I de Bakker, M J Daly, and P C Sham. PLINK: a tool set for whole-genome association and population-based linkage analyses. *Am J Hum Genet*, 81(3):559–575, Sep 2007.

[11] C C Laurie, K F Doheny, D B Mirel, E W Pugh, L J Bierut, T Bhangale, F Boehm, N E Caporaso, M C Cornelis, H J Edenberg, S B Gabriel, E L Harris, F B Hu, K B Jacobs, P Kraft, M T Landi, T Lumley, T A Manolio, C McHugh, I Painter, J Paschall, J P Rice, K M Rice, X Zheng, B S Weir, and GENEVA Investigators. Quality control and quality assurance in genotypic data for genome-wide association studies. *Genet Epidemiol*, 34(6):591–602, Sep 2010.

[12] X Zhu, S Li, R S Cooper, and R C Elston. A unified association analysis approach for family and unrelated samples correcting for stratification. *Am J Hum Genet*, 82(2):352–365, Feb 2008.

[13] A L Price, N A Zaitlen, D Reich, and N Patterson. New approaches to population stratification in genome-wide association studies. *Nat Rev Genet*, 11(7):459–463, Jul 2010.

[14] S M Gogarten, T Bhangale, M P Conomos, C A Laurie, C P McHugh, I Painter, X Zheng, D R Crosslin, D Levine, T Lumley, S C Nelson, K Rice, J Shen, R Swarnkar, B S Weir, and C C Laurie. GWASTools: an R/Bioconductor package for quality control and analysis of Genome-Wide Association Studies. *Bioinformatics*, Oct 2012.