SeqArray – A storage-efficient high-performance data format for WGS variant calls

Xiuwen Zheng [1,*], Stephanie M. Gogarten [1], Michael Lawrence [2], Adrienne Stilp [1], Matthew P. Conomos [1], Bruce S. Weir [1], Cathy Laurie [1], David Levine [1]

[1] Department of Biostatistics, University of Washington, Seattle, WA, USA; [2] Bioinformatics and Computational Biology, Genentech, Inc., South San Francisco, CA, USA.

Supplementary Materials

Table S1: Genotype decompression using SIMD instructions (in minutes) [a].

Input File Format	Compression Algorithm	No SIMD [b]		SSE2 [c]		AVX2 [d]	
		Speed [e]	Time	Speed [e]	Time	Speed [e]	Time
SeqArray	zlib	348.6	9.72 m	759.7	4.46 m	849.2	3.99 m
SeqArray	lzma	319.9	10.6 m	629.1	5.39 m	689.2	4.92 m

[a]: dataset: 1000 Genomes Project Phase 3, 2,504 individuals and 81 million variants
[b]: bit unpacking is not optimized using Single Instruction Multiple Data (SIMD) instruction
[c]: bit unpacking is optimized by Streaming SIMD Extensions 2 (SSE2) instructions
[d]: bit unpacking is optimized by Advanced Vector Extensions 2 (AVX2) instructions
[e]: million genotypes per second

Table S2: Random-access performance. Genotype queries in 250 autosomal regions of one million basepairs each randomly selected. For each region, the samples from 20 random 1000G populations are selected, and the genotypes are exported to an uncompressed VCF file.

Software	Source File Format	Compression Algorithm	Time (minute)
1000G: 2,504 individuals and 81 million variants			
Num. of variants in 250 random regions: 30680.7 ± 4013.0 *(mean ± sd)*			
`bcftools`	VCF.gz	zlib	31.2 m
`bcftools`	BCF	zlib	14.9 m
`BGT`	BGT	pbwt	10.6 m
`SeqArray`	SeqArray	zlib	10.9 m
`SeqArray`	SeqArray	lzma	20.4 m

2-bit array (M_{2x3x7})

M[1, ,]: *the first allele*

	variant 1	variant 2		variant 3	variant 4		
sample 1	0	0	1	1	0	2	0
sample 2	1	3	1	2	0	0	1
sample 3	3	3	3	0	3	3	3

M[2, ,]: *the second allele*

	variant 1	variant 2		variant 3	variant 4		
sample 1	0	0	0	0	0	0	0
sample 2	1	3	0	0	0	1	0
sample 3	2	1	3	3	3	3	1

Genotype array (G_{2x3x4})

G[1, ,]: *the first allele*

	variant 1	variant 2	variant 3	variant 4
sample 1	0	4	1	8
sample 2	1	7	2	16
sample 3	NA	NA	0	NA

G[2, ,]: *the second allele*

	variant 1	variant 2	variant 3	variant 4
sample 1	0	0	0	0
sample 2	1	3	0	4
sample 3	2	13	NA	31

	variant 1	variant 2	variant 3	variant 4
extra vector	1	2	1	3

indicating how many bits used for each variant

```
G[,,1] := (M[,,1] == 3) ? NA : M[,,1]
G[,,2] := (M[,,2] + M[,,3]*4 == 15) ? NA : M[,,2] + M[,,3]*4
G[,,3] := (M[,,4] == 3) ? NA : M[,,4]
G[,,4] := (M[,,5] + M[,,6]*4 + M[,,7]*16 == 63) ? NA : M[,,5] + M[,,6]*4 + M[,,7]*16
```

Figure S1: Diploid genotype decoding from a 2-bit integer array with 3 samples and 4 variants. There are four possible values in a 2-bit integer. The first dimension of M is used for ploidy and the size of the second dimension is the number of samples. The 2-bit array is used to construct the genotype array with an extra vector which indicates the number of bits in each variant. Zero in the genotype array $G_{2\times3\times4}$ denotes for the reference allele, 1 is the first alternative allele, 2 is the second alternative allele and so on. When the number of bits at a site is greater than 2, G is calculated by merging the information in the adjacent matrices, e.g., G[, , 2] is the combination of M[, , 2] and M[, , 3] for the second variant. The missing value at a site is the largest possible value with respect to the bit number of that site. For example, the missing value is 3 for the first variant and 15 for the second variant.

Value	Byte 1								Byte 2								Byte 3								Byte 4								Byte 5							
	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_{16}	b_{15}	b_{14}	b_{13}	b_{12}	b_{11}	b_{10}	b_9	b_{24}	b_{23}	b_{22}	b_{21}	b_{20}	b_{19}	b_{18}	b_{17}	b_{32}	b_{31}	b_{30}	b_{29}	b_{28}	b_{27}	b_{26}	b_{25}	b_{40}	b_{39}	b_{38}	b_{37}	b_{36}	b_{35}	b_{34}	b_{33}
-2^6 .. 2^6-1	0	x	x	x	x	x	x	s																																
	value := (b_1 == 0) ? ($b_7b_6b_5b_4b_3b_2$) : ~($b_7b_6b_5b_4b_3b_2$)																																							
-2^{13} .. -2^6-1	1	x	x	x	x	x	x	s	0	x	x	x	x	x	x	x																									
or 2^6 .. 2^{13}-1	value := (b_1 == 0) ? (b_{15}..b_9b_7..b_2) : ~(b_{15}..b_9b_7..b_2)																																							
-2^{20} .. -2^{13}-1	1	x	x	x	x	x	x	s	1	x	x	x	x	x	x	x	0	x	x	x	x	x	x	x																	
or 2^{13} .. 2^{20}-1	value := (b_1 == 0) ? (b_{23}..$b_{17}b_{15}$..b_9b_7..b_2) : ~(b_{23}..$b_{17}b_{15}$..b_9b_7..b_2)																																							
-2^{27} .. -2^{20}-1	1	x	x	x	x	x	x	s	1	x	x	x	x	x	x	x	1	x	x	x	x	x	x	x	0	x	x	x	x	x	x	x								
or 2^{20} .. 2^{27}-1	value := (b_1 == 0) ? (b_{31}..$b_{25}b_{23}$..$b_{17}b_{15}$..b_9b_7..b_2) : ~(b_{31}..$b_{25}b_{23}$..$b_{17}b_{15}$..b_9b_7..b_2)																																							
-2^{31} .. -2^{27}-1	1	x	x	x	x	x	x	s	1	x	x	x	x	x	x	x	1	x	x	x	x	x	x	x	1	x	x	x	x	x	x	x	0	0	0	0	x	x	x	x
or 2^{27} .. 2^{31}-1	value := (b_1 == 0) ? (b_{36}..$b_{33}b_{31}$..$b_{25}b_{23}$..$b_{17}b_{15}$..b_9b_7..b_2) : ~(b_{36}..$b_{33}b_{31}$..$b_{25}b_{23}$..$b_{17}b_{15}$..b_9b_7..b_2)																																							

Figure S2: Variable-length encoding of 32-bit signed integers. "b_i" denotes the i^{th} bit in the byte vector and $b_i = 0$ or 1. The first bit is the sign bit of signed integers, and the last bit in each byte is used to indicate whether the next byte is needed.

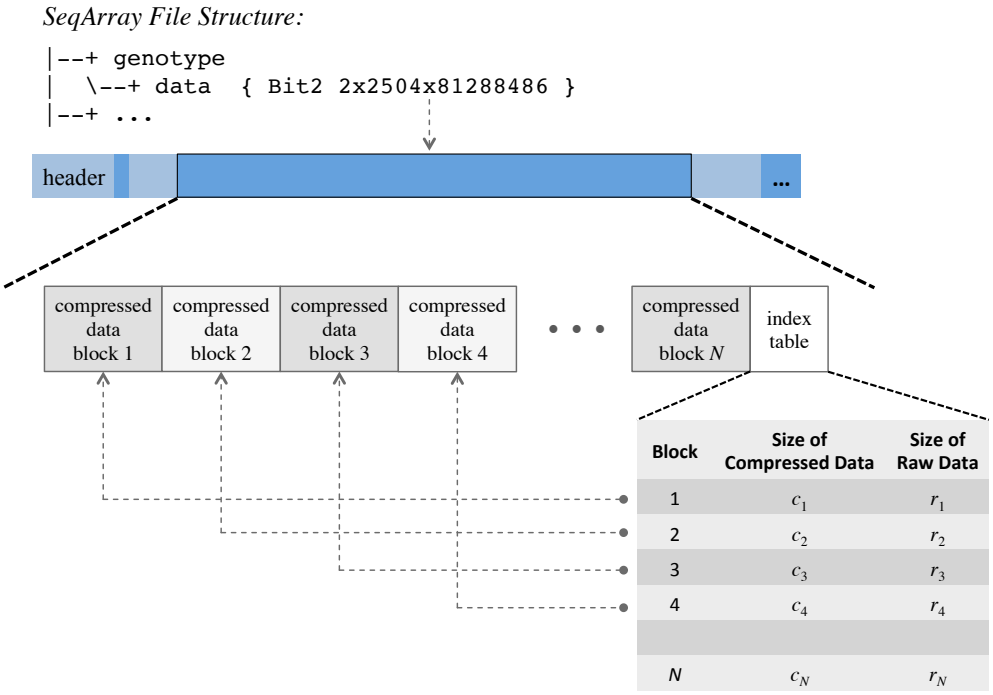

Figure S3: The indexing strategy of compressed data in a SeqArray file. The data field of "genotype" consists of N independent compressed data blocks and two auxiliary vectors, $\{c_1, c_2, ..., c_N\}$ and $\{r_1, r_2, ..., r_N\}$, appended to the end of the last block. The storage algorithm tries to keep the size of each block to be the same (256K by default). For a positional data query, the block of compression is identified by searching the position in the index table, and the data are decompressed and extracted from this block.

```r
# Download 1000 Genomes genotypes
FTP_PATH <- "ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/release/20130502"
for (ch in 1:22)
{
    fn <- file.path(FTP_PATH,
      sprintf("ALL.chr%d.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf.gz", ch))
    system(paste0("wget -q ", fn))
    system(paste0("wget -q ", fn, ".tbi"))
}

# Merge to VCF.gz
fn <- sprintf("ALL.chr%d.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf.gz", 1:22)
cmd <- paste("bcftools concat", "-Oz", "-o", "1kg_phase3_autosome.vcf.gz",
    paste(fn, collapse=" "))
system.time(system(cmd))

# Merge to VCF.gz with 4 threads
cmd <- paste("bcftools concat", "--threads 4", "-Oz", "-o", "1kg_phase3_autosome_mc4.vcf.gz",
    paste(fn, collapse=" "))
system.time(system(cmd))

# Merge to BCF
fn <- sprintf("ALL.chr%d.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf.gz", 1:22)
cmd <- paste("bcftools concat", "-Ob", "-o", "1kg_phase3_autosome.bcf",
    paste(fn, collapse=" "))
system.time(system(cmd))

# Merge to BCF with 4 threads
cmd <- paste("bcftools concat", "--threads 4", "-Ob", "-o", "1kg_phase3_autosome_mc4.bcf",
    paste(fn, collapse=" "))
system.time(system(cmd))

# Convert to BGT
fn <- "1kg_phase3_autosome.vcf.gz"
bgt.fn <- "1kg_phase3_autosome_new.bgt"
cmd <- paste("bgt import", "-S", bgt.fn, fn, collapse=" ")
system.time(system(cmd))

# Convert to SeqArray (zlib/lzma)
library(SeqArray)
fn <- sprintf("ALL.chr%d.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf.gz", 1:22)
system.time(
    seqVCF2GDS(fn, "1kg_p3_autosome_zlib_mc1.gds", storage.option="ZIP_RA", parallel=1L))
system.time(
    seqVCF2GDS(fn, "1kg_p3_autosome_zlib_mc4.gds", storage.option="ZIP_RA", parallel=4L))
system.time(
    seqVCF2GDS(fn, "1kg_p3_autosome_lzma_mc1.gds", storage.option="LZMA_RA", parallel=1L))
system.time(
    seqVCF2GDS(fn, "1kg_p3_autosome_lzma_mc4.gds", storage.option="LZMA_RA", parallel=4L))
```

Figure S4: R scripts for format conversion and compression of 1000 Genomes data.

```c
// FILE: bcf_read.c
// gcc -O2 bcf_read.c -Ihtslib htslib/libhts.a -lz -o bcf_read
#include <stdio.h>
#include <htslib/hts.h>
#include <htslib/vcf.h>

int main(int argc, char **argv)
{
	if (argc == 2)
	{
		char *fname = argv[1];
		htsFile *fp = hts_open(fname, "rb");
		bcf_hdr_t *hdr = bcf_hdr_read(fp);
		bcf1_t *rec = bcf_init1();

		int ngt, *gt_arr = NULL, ngt_arr = 0;
		while (bcf_read1(fp, hdr, rec) >= 0)
		{
			bcf_unpack(rec, BCF_UN_FMT);
			ngt = bcf_get_genotypes(hdr, rec, &gt_arr, &ngt_arr);
		}

		hts_close(fp);
		return 0;
	} else {
		printf("usage: bcf_read filename.bcf\n"); return -1;
	}
}
```

(a)

```r
# Genotype decompression, htslib
system.time(system("bcf_read 1kg_phase3_autosome.vcf.gz"))
system.time(system("bcf_read 1kg_phase3_autosome.bcf"))

# Genotype decompression, SeqArray (zlib/lzma)
library(SeqArray)
read_geno <- function(fn)
{
    for (ncpu in c(1, 4, 8, 16))
    {
        cat("Number of Cores: ", ncpu, "\n", sep="")
        print(system.time({
            gdsfile <- seqOpen(fn)
            seqParallel(ncpu, gdsfile, FUN=function(f) {
                seqApply(f, "genotype", function(x) {}, .useraw=NA)
                NULL
            })
            seqClose(gdsfile)
        }))
    }
}
read_geno("1kg_p3_autosome_zlib_mc1.gds")
read_geno("1kg_p3_autosome_lzma_mc1.gds")
```

(b)

Figure S5: C and R scripts for genotype decompression of 1000 Genomes data: a) C program "`bcf_read`" that calls functions in the htslib library; b) genotype decompression with htslib and SeqArray.

```
# Allele frequency calculation
system.time(
    system(paste("vcftools", "--gzvcf", "1kg_phase3_autosome.vcf.gz", "--freq", "--out", "q.txt")))
system.time(
    system(paste("vcftools", "--bcf", "1kg_phase3_autosome.bcf", "--freq", "--out", "q.txt")))
system.time(
    system(paste("plink", "--vcf", "1kg_phase3_autosome.vcf.gz", "--allow-extra-chr", "--freq")))
system.time(
    system(paste("plink", "--bcf", "1kg_phase3_autosome.bcf", "--allow-extra-chr", "--freq")))

# Allele frequency calculation, SeqArray (zlib/lzma)
library(SeqArray)
run_afreq <- function(fn)
{
    for (ncpu in c(1, 4, 8, 16))
    {
        cat("Number of Cores: ", ncpu, "\n", sep="")
        print(system.time({
            gdsfile <- seqOpen(fn)
            af <- seqAlleleFreq(gdsfile, ref.allele=NULL, parallel=ncpu)
            seqClose(gdsfile)
        }))
    }
}
run_afreq("1kg_p3_autosome_zlib_mc1.gds")
run_afreq("1kg_p3_autosome_lzma_mc1.gds")
```

Figure S6: R scripts for allele frequency calculation of 1000 Genomes data.

```
library(Rcpp)

# Define an inline C++ function
cppFunction("
    double CalcFreq(IntegerVector x)
    {
        int len=x.size(), n=0, n0=0;
        for (int i=0; i < len; i++)
        {
            int g = x[i];
            if (g != NA_INTEGER)
            {
                n++;
                if (g == 0) n0++;
            }
        }
        return double(n0) / n;
    }")

# Apply the function to a genotype margin
seqApply(file, "genotype", as.is="double", margin="by.variant", FUN=CalcFreq)
```

Figure S7: The inline C++ codes with the Rcpp package for defining a function in R and passed to seqApply().

```
# Covariance variable with an initial value
s <- 0

seqBlockApply(file, "$dosage", function(x)
   {
       p <- 0.5 * colMeans(x, na.rm=TRUE) # allele frequencies (a vector)
       g <- (t(x) - 2*p) / sqrt(p*(1-p))  # normalized by allele frequency
       g[is.na(g)] <- 0                   # correct missing values
       s <<- s + crossprod(g)             # update the cov matrix s in the parent environment
   }, margin="by.variant", .progress=TRUE)

# Scaled by the number of samples over the trace
s <- s * (nrow(s) / sum(diag(s)))
# Eigen-decomposition
eig <- eigen(s)

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

Figure S8: The implementation of principal component analysis in R with block-wise computations. `x` is a sample-by-variant integer matrix.

```
# The datasets are automatically split into four non-overlapping parts
genmat <- seqParallel(4, file, FUN = function(f)
   {
       s <- 0 # covariance variable with an initial value
       seqBlockApply(f, "$dosage", function(x)
           {
               p <- 0.5 * colMeans(x, na.rm=TRUE) # allele frequencies (a vector)
               g <- (t(x) - 2*p) / sqrt(p*(1-p))  # normalized by allele frequency
               g[is.na(g)] <- 0                   # correct missing values
               s <<- s + crossprod(g)             # update the cov matrix s in the parent
                   environment
           }, margin="by.variant")
       s  # output
   }, .combine = "+",   # sum "s" of different processes together
   split = "by.variant")

# Scaled by the number of samples over the trace
genmat <- genmat * (nrow(genmat) / sum(diag(genmat)))

# Eigen-decomposition
eig <- eigen(genmat)

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

Figure S9: The parallel implementation of principal component analysis in R.

```r
#### Download VCF files ####

# create a folder for downloading VCF files
dir.create("Download")
setwd("Download")

# genotypes (GT) + per-sample read depth (DP)
for (ch in 1:22)
{
    fn <- sprintf("ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/release/20130502/supporting/vcf_with_sample_level_annotation/
        ALL.chr%d.phase3_shapeit2_mvncall_integrated_v5_extra_anno.20130502.genotypes.vcf.gz", ch)
    system(paste0("wget -q ", fn))
    system(paste0("wget -q ", fn, ".tbi"))
}

# log-scaled genotype likelihoods (GL)
for (ch in 1:22)
{
    fn <- sprintf("ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/release/20130502/supporting/genotype_likelihoods/shapeit2/ALL.chr%
        d.phase3_bc_union.20130502.biallelic_svm_snps_indelsAF0.005_svs.gl.vcf.gz", ch)
    system(paste0("wget -q ", fn))
    system(paste0("wget -q ", fn, ".tbi"))
}

setwd("..")
```

(a) VCF file download.

```r
#### Find the common variants between different VCF files ####

skip_vcf_header <- function(f)
{
    while (TRUE)
    {
        s <- suppressWarnings(readLines(f, n=1L))
        if (length(s) <= 0L) break
        if (substr(s, 1L, 6L) == "#CHROM") break
    }
    s
}

get_variant_pos <- function(fn)
{
    f <- gzfile(fn, "rt"); on.exit(close(f))
    skip_vcf_header(f)
    pos <- NULL
    while (length(s <- suppressWarnings(readLines(f, n=10000L))))
    {
        pos <- c(pos, sapply(strsplit(s, "\t", fixed=TRUE), `[`, i=2L))
        cat(date(), "\t", prettyNum(length(pos), ","), "\n", sep="")
    }
    pos
}

for (ch in 1:22)
{
    pos_dp <- get_variant_pos(sprintf(
        "Download/ALL.chr%d.phase3_shapeit2_mvncall_integrated_v5_extra_anno.20130502.genotypes.vcf.gz", ch))
    pos_pl <- get_variant_pos(sprintf(
        "Download/ALL.chr%d.phase3_bc_union.20130502.biallelic_svm_snps_indelsAF0.005_svs.gl.vcf.gz", ch))
    z <- list(pos_dp=pos_dp, pos_pl=pos_pl)
    save(z, file=sprintf("1kg_p3_gt_dp_pl_pos_chr%d.rdata", ch))
}
```

(b) Common variants between different VCF files.

Figure S10: R scripts for re-formatting VCF files with genotypes (GT), per-sample read depth (DP) and Phred-scaled genotype likelihoods (PL) of 1000 Genomes data. a) VCF file download; b) common variants between different VCF files; c) creating VCF files with GT, DP and PL; d) re-formatting VCF files to other formats.

```r
#### Create VCF files with GT, DP and PL ####

for (ch in 1:22)
{
    fn1 <- sprintf("Download/ALL.chr%d.phase3_shapeit2_mvncall_integrated_v5_extra_anno.20130502.genotypes.vcf.gz", ch)
    fn2 <- sprintf("Download/ALL.chr%d.phase3_bc_union.20130502.biallelic_svm_snps_indelsAF0.005_svs.gl.vcf.gz", ch)
    outfn <- sprintf("ALL.chr%d.phase3.vcf.gz", ch)

    d <- get(load(sprintf("1kg_p3_gt_dp_pl_pos_chr%d.rdata", ch)))
    idlist <- intersect(d$pos_pl, d$pos_dp)
    i1 <- match(idlist, d$pos_dp)
    i2 <- match(idlist, d$pos_pl)
    if (is.unsorted(i1) | is.unsorted(i2))
        stop("chromosome position is unsorted.")

    f1 <- gzfile(fn1, "rt")
    f2 <- gzfile(fn2, "rt")
    outf <- pipe(paste("bgzip >", outfn), "wt") # bgzip (blocked gzip) compression

    s <- c("##fileformat=VCFv4.0",
        "##reference=GRCh37",
        "##FILTER=<ID=PASS,Description=\"All filters passed\">",
        "##FORMAT=<ID=GT,Number=1,Type=String,Description=\"Genotype\">",
        "##FORMAT=<ID=DP,Number=1,Type=Integer,Description=\"Per sample read depth\">",
        "##FORMAT=<ID=PL,Number=G,Type=Integer,Description=\"three -10 * (log10-scaled likelihoods) for RR,RA,AA genotypes\">",
        sprintf("##contig=<ID=%d,assembly=b37>", ch))
    writeLines(s, outf)

    s1 <- unlist(strsplit(skip_vcf_header(f1), "\t", fixed=TRUE))
    s2 <- unlist(strsplit(skip_vcf_header(f2), "\t", fixed=TRUE))
    s <- intersect(s1, s2)
    i1 <- match(s, s1)
    i2 <- match(s, s2)
    writeLines(paste(s1[i1], collapse="\t"), outf) # sample.id

    ii <- -(1:9)
    for (id in idlist)
    {
        while (TRUE)
        {
            s1 <- suppressWarnings(readLines(f1, n=1L))
            s1 <- unlist(strsplit(s1, "\t", fixed=TRUE))
            if (s1[2L] == id) break
        }
        s1 <- s1[i1]

        while (TRUE)
        {
            s2 <- suppressWarnings(readLines(f2, n=1L))
            s2 <- unlist(strsplit(s2, "\t", fixed=TRUE))
            if (s2[2L] == id) break
        }
        s2 <- s2[i2]

        s1[8L] <- "."; s1[9L] <- "GT:DP:PL"

        s <- s2[ii]
        s[s == "."] <- ".,.,."
        s <- suppressWarnings(round(
            -10 * sapply(strsplit(s, ",", fixed=TRUE), as.double)))
        s <- apply(s, 2L, paste, collapse=",")
        s[s == "NA,NA,NA"] <- "."
        s1[ii] <- paste(s1[ii], s, sep=":")

        writeLines(paste(s1, collapse="\t"), outf)
    }

    close(f1); close(f2); close(outf)
}
```

(c) Creating VCF files with GT, DP and PL.

Figure S10: R scripts for re-formatting VCF files with genotypes (GT), per-sample read depth (DP) and Phred-scaled genotype likelihoods (PL) of 1000 Genomes data. a) VCF file download; b) common variants between different VCF files; c) creating VCF files with GT, DP and PL; d) re-formatting VCF files to other formats.

```
#### Convert VCF to BCF ####

for (ch in 1:22)
{
    vcf <- sprintf("ALL.chr%d.phase3.vcf.gz", ch)
    bcf <- sprintf("ALL.chr%d.phase3.GT_DP_PL.bcf", ch)
    cmd <- paste("bcftools convert -Ob -o", bcf, vcf)
    system(cmd)
}

#### Convert VCF to GDS (zlib) ####

library(SeqArray)

for (ch in 1:22)
{
    vcf <- sprintf("ALL.chr%d.phase3.vcf.gz", ch)
    gds <- sprintf("ALL.chr%d.phase3.GT_DP_PL.gds", ch)
    seqVCF2GDS(vcf, gds, storage.option="ZIP_RA", parallel=4L)
}

#### Convert VCF to GDS (lzma) ####

for (ch in 1:22)
{
    vcf <- sprintf("ALL.chr%d.phase3.vcf.gz", ch)
    gds <- sprintf("ALL.chr%d.phase3.GT_DP_PL_lzma.gds", ch)
    seqVCF2GDS(vcf, gds, storage.option="LZMA_RA", parallel=4L)
}
```

(d) Re-formatting VCF files to other formats.

Figure S10: R scripts for re-formatting VCF files with genotypes (GT), per-sample read depth (DP) and Phred-scaled genotype likelihoods (PL) of 1000 Genomes data. a) VCF file download; b) common variants between different VCF files; c) creating VCF files with GT, DP and PL; d) re-formatting VCF files to other formats.