vcfpp: a C++ API for scripting rapid variant analysis with the VCF/BCF

Zilong Li

Abstract

Given the complexity and popularity of the VCF/BCF format as well as ever-growing data size, there is always a need for fast and flexible methods to manipulate it in different programming languages. Many bioinformatics tools developed in C++ for high performance, and modern C++ standards strive to provide growing standard library for developing program easily. This work presents *vcfpp*, a C++ API of htslib in a single file, providing an intuitive interface to manipulate VCF/BCF files rapidly and safely, as well as being portable. Additionally, this work introduces the *vcfppR* package to demonstrate the usage of developing high performance R package with *vcfpp* seamlessly and esaily for analyzing and visualizing genomic variations, such as characterizing the variations in the VCF. In the Benchmarking, the *vcfppR* shows the same performance as the C++ program, and is 1.4× faster than *cyvcf2* when streaming variant analysis using little memory. Also, in a two-step setting where the whole VCF content is loaded first, the *vcfppR* shows 110× speed improvement over the *vcfR* package in processing genotypes.

Zilong Li: Section for Computational and RNA Biology, Department of Biology, University of Copenhagen.
Email: zilong.dk@gamil.com.

Introduction

Many efforts have been contributed by computational biologists, aim for making analyses of genomics variants no pain. The variants are typically stored in the VCF format, which becomes the standard for representing genetic variation information with complicated specification (Danecek et al. 2011). Later on, as the ever-growing data size, the BCF format is designed to query and store big data efficiently. The C API of htslib provide the full functionalities to manipulate the VCF and BCF format for both compressed and uncompressed files. As the C API is hard to use for in-proficient programmers, API of other languages are invented. Existing popular libraries include vcfR(Brian et al. 2017) for R, cyvcf2 (Pedersen et al. 2017) for Python, hts-nim (Pedersen et al. 2018) for Nim and vcflib (Garrison et al. 2022) for C++. All are valuable for corresponding community but not perfect. In particular, vcflib is both an API and a large collection of command line tools, with the primary disadvantage being not supporting the BCF format. It is noteworthy that many methods written in C++ designed for large sample size can not even input the BCF file tailed for big data or the compressed VCF file like Syllbable-PBWT (Wang et al. 2023). The motivation of vcfpp is to offer full functionalities as htslib and provide simple and safe API in a single header file that can be easily integrated for writing program quickly in C++ and R.

Methods

vcfpp is implemented as a single header file for being easily integrated and compiled. There are four core class for manipulating the uncompressed and compressed VCF/BCF files as summarized in Table 1.

TABLE 1. vcfpp capabilities and implemented C++ class

Capability	Class
VCF/BCF read	BcfReader
VCF/BCF write	BcfWriter
VCF/BCF variants and operations	BcfRecord
VCF/BCF header and operations	BcfHeader

Results

To demonstrate the simplicity and performance of vcfpp, the following sections illustrate commonly used features of vcfpp and highlight the vcfppR package as an example of vcfpp working with R.

C++ API

In the Listing 1, we count the number of heterozygous sites for each sample across all records. The following code first includes a single *vcfpp.h* file (line 1), opens a compressed BCF file constrained to 3 samples and region "chr21" (line 2), and creates a variant record associated with the header information in the BCF (line 3). Then it defines several types of objects to collect the results we want (line 4-5). Taking advantage of generic templates in C++, the type of field like genotype can be very flexible regarding memory consumption. Then it starts iterating over the BCF file and processes each variant record in the loop (line 6). We ignore variants of other types (INDEL, SV), or with *FILTER* not being "PASS", or with *QUAL* value smaller than 9 (line 8-9), while we can also do more complicated filterings using the API. Finally, we count the number of heterozygous variant for each diploid sample (line 10-11). The core is just 12 lines.

Listing 1. Counting the number of heterozygous genotypes for 3 samples on chr21

```
1  #include <vcfpp.h>
2  vcfpp::BcfReader vcf("bcf.gz", "chr21", "id1,id2,id3");
3  vcfpp::BcfRecord var(vcf.header); // create a variant object
4  vector<int> gt; // genotype can be bool, char or int type
5  vector<int> hets(vcf.nsamples, 0); // store the het counts
6  while(vcf.getNextVariant(var)){
7    var.getGenotypes(gt);
8    if(!var.isSNP()||var.QUAL()<9||var.FILTER()!="PASS")continue;
9    assert(var.ploidy()==2); // make sure it is diploidy
10   for(int i=0; i<vcf.nsamples; i++)
11     hets[i] += abs(gt[2*i+0] - gt[2*i+1])==1;
12 }
```

Working with R

While vcfpp is very simple for writing a C++ program, a single C++ header file can be easily integrated into popular script languages like R and Python etc. And R is made for statistical modeling and visualization, which is widely used in data analyses. Therefore, I develop the vcfppR package to demonstrate how vcfpp can work with R seamlessly by using Rcpp (Eddelbuettel et al. 2011). For instance, with the basic knowledge of C++ and Rcpp, we can turn the C++ code in Listing 1 into the Rcpp function to return a list of the number heterzygous sites per sample along with the sample's name (Listing 2), which can be compiled and called in R (Listing 3). As such, we can further proccess and visualize the results efficiently in R, such as run the analyses by region in parallel with the *parallel* package, stratifying the results by populations given external labels of each sample and visualizing them in R.

Listing 2. vcfpp-hets.cpp

```
#include <Rcpp.h>
#include <vcfpp.h>
using namespace Rcpp;
using namespace std;
// [[Rcpp::export]]
List heterozygosity(std::string vcffile,
                    std::string region,
                    std::string samples) {
  vcfpp::BcfReader vcf(vcffile, region, samples);
  // here copy the lines 3-12 in listing 1.
  return List::create(Named("samples")=vcf.header.getSamples(),
                      Named("hets")=hets);
}
```

Listing 3. The R code compiles and calls the vcfpp-hets.cpp

```
library(Rcpp)
sourceCpp("vcfpp-hets.cpp")
heterozygosity("bcf.gz", "chr21","id1,id2,id3");
```

The vcfppR package

The vcfppR package is developed and powered by the vcfpp API. For parsing the VCF in R, the *vcftable* function can read contents of the VCF into the R data type rapidly with fine control of the region, the samples, the variant types, the FORMAT column and the filters. For instance, the code in Listing 4 parses the read depth per variant (DP) in the raw called VCF by the 1000 Genomes Project through the URL link, restricting to 3 samples with variants in "chr21:1-10000000" region of SNP type and passing FILTER, as well as discarding the INFO column in the returned list. Consequently, the visual summarization is done by using *boxplot* in R (see Figure 1).

Listing 4. vcftable example

```
library(vcfppR)
vcffile <- "https://url-path"
vcf <- vcftable(vcffile, region="chr21:1-10000000", samples="
   NA12878,HG00118,HG00119", format="DP", vartype="snps", pass
   = TRUE, info = FALSE)
boxplot(vcf$dp, names=vcf$samples, ylab="Read Depth (DP)")
```

Furthermore, as characterization of the variations is an essential task in understanding the genome, I showcase the *vcfsummary* function in Figure 1 in summarizing the variations in the latest VCF released by the 1000 Genome Project (Byrska-Bishop et al. 2022).

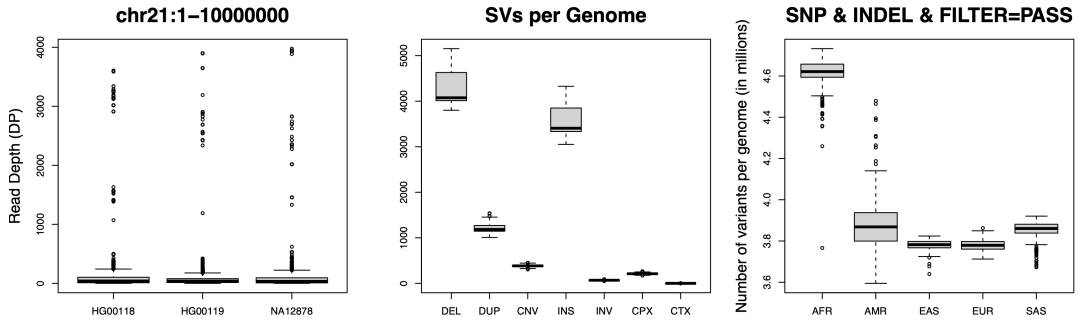

FIGURE 1. Characterizing the variations in the 1000 Genome Project using the vcfppR package (see code in supplementary).

Benchmarking

In addition to simplicity and portability, I showcase the performance of vcfpp and vcfppR. In the benchmarking, I performed the same task as the cyvcf2 paper, that is counting heterozygous genotypes per sample using code in Listing 1. As shown in Table 2, the vcfppR has the same performance as the vcfpp C++ API, while the cyvcf2 script is 1.4× slower than the vcfppR script. In the streaming setting, all three scripts use little RAM for only loading one variant into memory. However, R packages like vcfR and data.table usually load all tabular data into memory first and perform analysis later. Additionaly, I develop *vcftable* function in vcfppR to load whole VCF content in R for such two-step comparison. Notably, the vcfppR is only 1.8× slower compared to 15.5×, 97.5× slower for vcfR and data.table respectively. This is because genotype values returned by both vcfR and data.table::fread are characters, which are very slow to parse. In contrast, with vcfppR, integer matrix of genotypes can be returned to R directly for fast computation. Importantly, vcfpp and vcfppR offer the users the full functionalities of htslib, such as supporting BCF, selecting samples, regions and variant types. A rapid query of VCF content in vcfppR can be achieved by passing a region parameter to *vcftable*.

TABLE 2. Performance of counting heterozygous genotypes per sample in the 1000 Genome Project for chromosome 21. (*) used by loading data in two-step strategy.

API/Package	Time (s)	Ratio	RAM (Gb)	Strategy
vcfpp	118	1.0	0.006	streaming
vcfppR	118	1.0	0.076	streaming
cyvcf2	165	1.4	0.038	streaming
vcfppR	205*+10	1.8	64.7	two-step
vcfR	678*+1147	15.5	97.5	two-step
data.table	263*+11243	97.5	77.3	two-step

Discussion

I have developed vcfpp, a fast and flexible C++ API for scripting high-performance genetic variant analyses with the VCF/BCF file. Its simplicity and portability can be very useful for both developing packages and writing daily used scripts. Many packages written in C++ using customized VCF parser can be replaced with vcfpp to offer more

functionalities. For instance, imputation software STITCH (Davies, Flint, et al. 2016) and QUILT (Davies, Kucka, et al. 2021) are using vcfpp to parse large reference panel in the VCF/BCF format. Also, there are some useful command line tools available, such as a modified Syllable-PBWT with the compressed VCF/BCF as input.

Software and Code

The latest release of vcfpp.h and documentation can be found at https://github.com/Zilong-Li/vcfpp. Scripts for Benchmarking can be found at https://github.com/Zilong-Li/vcfpp/tree/main/scripts. The vcfppR package is available at https://github.com/Zilong-Li/vcfppR.

Acknowledgments

I would like to thank Anders Albrechtsen at Copenhagen University and Robert W Davies at Oxford University for helpful comments. They are Statisticians as well as R enthusiast working on genetics, whom I work with and learn a lot from.

Funding

This work is supported by the Novo Nordisk 462 Foundation (NNF20OC0061343).

References

Brian, JK and JG Niklaus (2017). "vcfr: a package to manipulate and visualize variant call format data in R". In: *Molecular Ecology Resources* 17.1, pp. 44–53. DOI: 10.1111/1755-0998.12549.

Byrska-Bishop, M et al. (2022). "High-coverage whole-genome sequencing of the expanded 1000 Genomes Project cohort including 602 trios". In: *Cell* 185.18, 3426–3440.e19. DOI: 10.1016/j.cell.2022.08.004.

Danecek, P et al. (2011). "The variant call format and VCFtools". In: *Bioinformatics (Oxford, England)* 27.15, pp. 2156–2158. DOI: 10.1093/bioinformatics/btr330. pmid: 21653522.

Davies, RW, J Flint, S Myers, and R Mott (2016). "Rapid genotype imputation from sequence without reference panels". In: *Nature Genetics* 48.8, pp. 965–969. DOI: 10.1038/ng.3594. pmid: 27376236.

Davies, RW, M Kucka, D Su, S Shi, M Flanagan, CM Cunniff, YF Chan, and S Myers (2021). "Rapid genotype imputation from sequence with reference panels". In: *Nature Genetics* 53.7 (7), pp. 1104–1111. DOI: 10.1038/s41588-021-00877-0.

Eddelbuettel, D and R Francois (2011). "Rcpp: Seamless R and C++ Integration". In: *Journal of Statistical Software* 40, pp. 1–18. DOI: 10.18637/jss.v040.i08.

Garrison, E, ZN Kronenberg, ET Dawson, BS Pedersen, and P Prins (2022). "A spectrum of free software tools for processing the VCF variant call format: vcflib, bio-vcf, cyvcf2, hts-nim and slivar". In: *PLOS Computational Biology* 18.5, e1009123. DOI: 10.1371/journal.pcbi.1009123.

Pedersen, BS and AR Quinlan (2017). "cyvcf2: fast, flexible variant analysis with Python". In: *Bioinformatics* 33.12, pp. 1867–1869. DOI: 10.1093/bioinformatics/btx057.

Pedersen, BS and AR Quinlan (2018). "hts-nim: scripting high-performance genomic analyses". In: *Bioinformatics* 34.19, pp. 3387–3389. DOI: 10.1093/bioinformatics/bty358.

Wang, V, A Naseri, S Zhang, and D Zhi (2023). "Syllable-PBWT for space-efficient haplotype long-match query". In: *Bioinformatics* 39.1, btac734. DOI: 10.1093/bioinformatics/btac734.

Supplementary

Listing 5. R code produces Figure 1. vcffiles (not shown) are url links.

```
library(vcfppR)
par(mfrow=c(1,3)) ## layout in plots
vcf <- vcftable(vcffile, region="chr21:1-10000000", samples="
   NA12878,HG00118,HG00119", format="DP", vartype="snps", pass
   = TRUE, info = FALSE)
boxplot(vcf$dp, names=vcf$samples, ylab="Read␣Depth␣(DP)")
svfile <- "https://ftp.1000genomes.ebi.ac.uk/vol1/ftp/data_
   collections/1000G_2504_high_coverage/working/20210124.SV_
   Illumina_Integration/1KGP_3202.gatksv_svtools_novelins.
   freeze_V3.wAF.vcf.gz"
sv <- vcfsummary(svfile, svtype = TRUE)
boxplot(sv[c("DEL","DUP", "CNV", "INS","INV","CPX","CTX")],
        main = "SVs␣per␣Genome␣stratified␣by␣SV␣types")
ped <- read.table("https://ftp.1000genomes.ebi.ac.uk/vol1/ftp/
   data_collections/1000G_2504_high_coverage/20130606_g1k_3202_
   samples_ped_population.txt", h=T)
ped <- ped[order(ped$Superpopulation),]
supers <- unique(ped$Superpopulation)
all <- parallel::mclapply(vcffiles, vcfsummary, pass = TRUE, mc
   .cores=23)
samples <- all[[1]]$samples
snps <- Reduce("+", lapply(all, "[[", "SNP"))
indels <- Reduce("+", lapply(all, "[[", "INDEL"))
o <- sapply(supers, function(pop) {
  id <- subset(ped, Superpopulation == pop)[,"SampleID"]
  ord <- match(id, samples)
  (snps[ord] + indels[ord]) / 1e6
})
boxplot(o, main = "SNP␣&␣INDEL␣with␣FILTER=PASS", ylab = "
   Number␣of␣variants␣per␣genome␣(in␣millions)")
```