*Bioinformatics*
doi.10.1093/bioinformatics/xxxxxx
Advance Access Publication Date: Day Month Year
Software

OXFORD

Genome Analysis

# hts-nim: scripting high-performance genomic analyses

**Brent S. Pedersen [1]* and Aaron R. Quinlan [1]**

[1] University of Utah, Department of Human Genetics, Department of Biomedical Informatics, and USTAR Center for Genetic Discovery.

*To whom correspondence should be addressed.

Associate Editor: XXXXXXX

Received on XXXXX; revised on XXXXX; accepted on XXXXX

## Abstract

**Motivation:** Processing genomic data requires custom software. In many cases, this is accomplished in scripting languages because of ease of writing and brevity.
**Results:** We present *hts-nim*, a library written in the Nim programming language that enables a scripting-like syntax without sacrificing performance.
**Availability:** *hts-nim* is available at https://github.com/brentp/hts-nim and the example tools are at https://github.com/brentp/hts-nim-tools both under the MIT license.
**Contact:** bpederse@gmail.com
**Supplementary information:** Supplementary data are available at *Bioinformatics* online.

## 1 Introduction

For genomics applications, it can be preferable to release command-line tools over libraries because it limits the *surface-area* exposed to users, provides the opportunity for hidden optimizations that guarantee speed regardless of use, and eases distribution. However, these tools are either limited to their intended use or they will suffer from feature-creep and the resulting complexity. In contrast, genomics libraries place greater burden on the user but offer increased flexibility. In order to fill the gap for fast custom analysis in a simple language, we expose the htslib library in the nim programming language in a library called *hts-nim*. *Nim* compiles to *C*, has very good performance, garbage collection, control over memory re-use, and a simple syntax that is easy to pick up for many programmers familiar with common scripting languages.

We have also previously published *mosdepth* (Pedersen and Quinlan, 2017) which uses *hts-nim* extensively and is now a widely-used tool. Now, we introduce the *hts-nim* library and show how it enables the creation fast tools that are easy to write. We present 3 example tools, each of which is a substantial and useful tool with command-line parsing and error-handling, but each is short enough to be readable in a few minutes. The first example allows filtering a BAM/CRAM file with a simple expression langauge, the second allows counting reads in genomic regions, and the third is a quality-control tool to ensure that regions are not missing from Variant Call Format (VCF) (Danecek *et al.*, 2011) files.

## 2 Approach

*hts-nim* is written in the nim programming language; low-level bindings from nim to htslib are created automatically using a tool called c2nim; then, hand-written and tested code is used for the user-exposed layer. This layer hooks into the garbage collection so that a user of *hts-nim* does not need to explicitly clean up objects or free memory as would be required in *C*. As much as possible, the interface exposed in *hts-nim* allows re-using memory to avoid pressure on the garbage collector. However, the user is also free to write code that results in more allocations for the sake of simplicity. For example, when accessing the base qualities of an alignment from a BAM file, the user passes in a *seq* that is filled by the *base_qualities* method on an *alignment* object; that *seq* is then filled inside the method and returned to the user. This allows the user to control the memory allocations, by either re-using the same container for every alignment, or allocating a new one before each call the the *base_qualities* method. The design of *hts-nim* and the *nim programming language* itself provide many opportunities for optimization like this that allow speed-memory trade-offs.

To show the syntax and scope of the library we cover 3 examples.

## 3 Examples

### 3.1 BAM/CRAM Filtering

It is common to filter alignment files using samtools (Li *et al.*, 2009); here, we introduce a complementary tool built with *hts-nim*. It uses a simple expression language, *kexpr* by Heng Li to parse and evaluate user-specified expressions. The documentation for the tool indicates the fields that are available for filtering, but briefly, flags are prefix with *is_*, and tags are prefixed with *tag_*. This exposes a type of filtering that is not available in *samtools*. An example of usage looks like:

**Listing 1.** bam-filter example
```
hts-nim-tools bam-filter --threads 2 "
    is_proper_pair && insert_size > 200"
    $input_bam > $out
```

## 3.2 Counting Alignments in Genomic Regions

We have previously released *mosdepth* (Pedersen and Quinlan, 2017) which calculates the average per-base coverage across the genome and within a given set of regions. However, in some cases, it is preferable to know the count of reads overlapping each region rather than the average per-base coverage. The *count-reads* tool performs this operation by iterating over each read in a BAM or CRAM file and checking first that it meets the required flag and mapping quality constraints and then if it overlaps a region of interest given in a BED file passed on the command-line. The overlap testing is done with a *nim* library we wrote to do very fast interval lookups. Usage of this module looks like:

**Listing 2.** count-reads example
```
hts-nim-tools count-reads --threads 2 --
    mapq 10 exons.bed $input_bam > $out.
    bed
```

For each line in the *exons.bed* file, this will count the number of overlapping reads and output that line plus that count to STDOUT. The user can also filter to specific alignment flags, but the default excludes reads that are duplicates, failed quality-control or secondary reads. We compared this tool to *samtools bedcov* which has a similar functionality except that it sums the total bases of coverage for each region. On an exome BAM file with 82 million reads and a BED file with about 1.2 million regions, this tool–*count-reads*–took 3 minutes and 8 seconds of CPU time while *samtools bedcov* took about 33 minutes. Simply counting the reads with *samtools view -c* takes 1 minute and 36 seconds. Rather than to compare exact times, this is to show the relative speed of *hts-nim* and the highlight ability to create very fast, custom tools in a few lines of code.

## 3.3 Quality Control Variant Call Files

Projects with many samples will often split the genome into regions for parallelization. It is possible that a few regions may result in truncated or no output because of a silent or uncaught error. This missing data can go unnoticed due the the large number of files and the complexity of processing steps. Here, we introduce a tool, *vcf-check* that takes a background VCF, e.g. from ExAC (Lek *et al.*, 2016) gnomAD (http://gnomad.broadinstitute.org/) to establish a base-line of what parts of the genome are expected to have common variation. It then compares chunks of the genome from the background and the query VCF so that the user can find regions that have no representation in the query VCF but are common in the backgrounds. We have found this crude metric to work well in finding regions of the genome that are lost in processing for whatever reason. An example invocation of this tool looks like:

**Listing 3.** vcf-check example
```
vcf-check --maf 0.1 $gnomad_vcf
    $query_vcf > $missed_txt
```

The tab-delimited output contains the count of variants above 0.1 allele frequency for both VCFs in each region. Missing regions from the query will appear as consecutive rows with counts of 0 where the corresponding counts from the background VCF are non-zero.

## 4 Discussion

We have demonstrated the breadth of *hts-nim*'s utility by introducing a set of tools for BAM and VCF processing. These tools are available with documentation at https://github.com/brentp/hts-nim-tools as a complement to the library documentation to aid users in creating their own programs. The speed and simplicity of the language combined with the utility provided by *htslib* (https://htslib.org) will make this a valuable library.

## Acknowledgements

*hts-nim* benefits from several ideas utilized rust-htslib which is itself an excellent genomic software library.

## Funding

This research was supported by awards to ARQ from the US National Human Genome Research Institute (NIH R01HG006693 and NIH R01HG009141), the US National Institute of General Medical Sciences (NIH R01GM124355), and the US National Cancer Institute (NIH U24CA209999).

## References

Danecek, P. *et al.* (2011). The variant call format and vcftools. *Bioinformatics*, **27**(15), 2156–2158.

Lek, M. *et al.* (2016). Analysis of protein-coding genetic variation in 60,706 humans. *Nature*, **536**(7616), 285–291.

Li, H. *et al.* (2009). The sequence alignment/map format and samtools. *Bioinformatics*, **25**(16), 2078–2079.

Pedersen, B. S. and Quinlan, A. R. (2017). mosdepth: quick coverage calculation for genomes and exomes. *Bioinformatics*, **1**, 2.