Bioinformatics, 32(1), 2016, 140–141
doi: 10.1093/bioinformatics/btv539
Advance Access Publication Date: 10 September 2015
Applications Note

OXFORD

Sequence analysis

BamHash: a checksum program for verifying the integrity of sequence data

Arna Óskarsdóttir[1], Gísli Másson[1] and Páll Melsted[1,2,*]

[1]deCODE Genetics/Amgen, Reykjavík, Iceland and [2]Faculty of Industrial Engineering, Mechanical Engineering and Computer Science, University of Iceland, Reykjavík, Iceland

*To whom correspondence should be addressed.

Associte Editor: Alfonso Valencia

Received on February 26, 2015; revised on July 29, 2015; accepted on September 7, 2015

Abstract

Summary: Large resequencing projects require a significant amount of storage for raw sequences, as well as alignment files. Because the raw sequences are redundant once the alignment has been generated, it is possible to keep only the alignment files. We present BamHash, a checksum based method to ensure that the read pairs in FASTQ files match exactly the read pairs stored in BAM files, regardless of the ordering of reads. BamHash can be used to verify the integrity of the files stored and discover any discrepancies. Thus, BamHash can be used to determine if it is safe to delete the FASTQ files storing raw sequencing read after alignment, without the loss of data.
Availability and implementation: The software is implemented in C++, GPL licensed and available at https://github.com/DecodeGenetics/BamHash
Contact: pmelsted@hi.is

1 Introduction

Resequencing projects, where individuals are sequenced from a species with a known reference genome, generate a significant amount of raw sequences that are then aligned to the reference genome. Data storage becomes an issue as the cost of sequencing decreases and the throughput of current sequencing technologies keeps increasing.

Raw sequencing reads are generally stored in FASTQ file format, usually compressed. After read mapping the resulting alignment is stored in a BAM file (Li *et al.*, 2009). This BAM file is then sorted and processed further, but most importantly it contains all the original information of the FASTQ file. Sorted BAM files yield a better compression, compared with unsorted BAM files, as well as allowing random lookup over genomic regions. For this reason almost all post-alignment analysis, e.g. variant calling, realignment, and local assembly are done on the sorted BAM file, rather than the original FASTQ file.

Because the BAM file contains all the information of the FASTQ file it is justifiable to delete the FASTQ file after alignment. After all, the contents of the FASTQ file can be regenerated from BAM file.

However, before deleting the FASTQ file, we need to be sure that there is no loss of data, i.e. that the sequences in the FASTQ file are exactly the same as the sequences in the BAM file. The two files could differ due to a number of reasons. Any errors in the alignment pipeline could generate inconsistent files. Although the alignment pipelines are based on well-tested tools, they are meant to operate under normal conditions and their behavior can be unpredictable in the presence of hardware failure or running out of disk space. Thus, it is important to be able to independently verify the output of the entire pipeline.

We present BamHash, a tool for verifying the data integrity between a FASTQ and a BAM file. The program computes a 64-bit fingerprint from the sequences and read names for both FASTQ and BAM files. The method is highly sensitive to changes in the input so a change in a single nucleotide will result in different fingerprints; the probability of generating the same fingerprint by chance is astronomically small. The role of this tool is to flag any FASTQ and BAM files that have different fingerprints and mark the FASTQ files as unsafe for deletion.

BamHash plays the same role as the md5sum program, which computes a fingerprint of files. Comparing md5sum fingerprints (Rivest, 1992) of FASTQ and BAM files would not yield a comparable result, since the formatting and ordering are different. Our method is fast and memory efficient; it can compute the fingerprint of a BAM file from 30-fold coverage human sequencing experiment in 30 min.

2 Methods

The information for the sequencing reads, namely the read name, sequence and quality values are stored differently in FASTQ and BAM files, but can easily be parsed and recovered. The internal order of reads is generally not conserved, unless guaranteed by the alignment software. Since BAM files sorted by genomic coordinates are the norm, we cannot expect to maintain the order.

Thus we need to compare the two files as sets, or rather multisets, of items. To do this we use a hash function h for each item and reduce all hash values using a commutative binary operation. The commutative property ensures that the final result is independent of the ordering of the reads.

For the commutative binary operation, the XOR function is a natural candidate for sets. However, XOR has the property that each value is it's own inverse, if an item x is repeated twice in the input, the hash values will cancel each other since $h(x) \oplus h(x) = 0$. Normally, we do not expect to see repeated items since read names tend to be unique; however if the read names have been stripped and quality values are absent we cannot guarantee that this holds. For this reason we chose to work with the sum of hash values as 64-bit integers.

By using a hash function we ensure that the resulting fingerprint is sensitive to any changes in the input. For BamHash, we chose the MD5 hash function. Whereas MD5 was developed for cryptographic purposes, making it hard to forge an MD5 signature, we only rely on the sensitivity of the hash function to catch accidental changes. It should be noted that the proposed method cannot guarantee that it would be too hard for a malicious agent to modify the input to produce any given fingerprint.

Algorithm 1. Checksum for paired sequences

```
function HASH-UPDATE-BAM (r)
  s ← r.name
  if r is first in pair then
    s ← s+ "/1"
  else
    s ← s+ "/2"
  if r is on reverse strand then
    r.seq ← REVERSE-COMPLEMENT (r.seq)
  s ← s + r.seq + r.qual
  return MD5 (s)
function HASH-UPDATE-FASTQ (r)
  s ← r.name + r.seq + r.qual
  return MD5 (s)
function HASH-FILE (f)
  H ← 0
  for all reads r in f do
    if f is a BAM file then
      H ← H +(HASH-UPDATE-BAM (r)) mod 2^64
    else
      H ← H +(HASH-UPDATE-FASTQ (r)) mod 2^64
  return H
```

2.1 Implementation

The pseudo-code for the method is given in Algorithm 1. For FASTQ files, the input for paired reads is given by two FASTQ files. Each read in the FASTQ file is processed and the read names for pairs are modified to end in /1 or /2 (we do not expect the input to follow this convention it is only used in the hashing stage). For BAM files, a single BAM file is given and the flags are used to determine whether each read is the first or second in a read pair. Furthermore, reads that are mapped to the reverse strand have the reverse complement of the sequence stored in the BAM file to aid compression. For these reads we reconstruct the original sequence to match what was stored in the corresponding FASTQ file. In addition to this, BamHash supports working with multiple files, single and paired reads, as well as options of ignoring quality values or read names when computing the fingerprints. The program is written in C++ and uses the SeqAn library (Döring *et al.*, 2008) for parsing FASTQ, gzip compressed FASTQ and BAM files.

3 Results

To assess the performance of BamHash, we compared the running time for processing BAM files to viewing with Samtools. The dataset chosen was a whole genome sequencing experiment, aligned to GRCh38 Human reference using BWA-MEM. All datasets were generated at the laboratory at deCODE Genetics and were processed with the same pipeline (Gudbjartsson *et al.*, 2015). The BAM file consists of 832 million read pairs at 38× coverage. BamHash required 38 min to compute the hash values, whereas Samtools required 49 min to parse the BAM file and count lines. We note that the program is largely I/O bound. It runs on a single core, which is underutilized, as most of the time is spent waiting for data from the disk.

4 Discussion

The role of BamHash is to detect differences between the read sets of raw FASTQ and aligned BAM files. This discrepancy can arise due to mistakes in the pipeline, bugs in alignment code or disk failures. When the data integrity has been verified, the original FASTQ files can be safely discarded, thus freeing up storage space. Additionally, BamHash will be useful when porting alignments to a new reference genome. Such a pipeline would create intermediate FASTQ files, which would then be aligned to the new reference. The old BAM file can be removed only if the BamHash signature agrees with the newly created alignment. BamHash can only detect differences between exact matches of set of reads, not how they differ. In many scenarios, low quality reads are discarded before alignment, or reads that do not map are discarded from the BAM file. In this case the set of reads in the final BAM file is a subset of the original set of reads. Unfortunately, no fingerprinting method can detect if the BAM reads are a subset of the FASTQ reads. This is because fingerprinting is a restricted form of communication between two parties, the BAM hasher and the FASTQ hasher, and lower bounds on the communication complexity of the set-disjointness problem (Razborov, 1992) dictate a lower bound of $\Omega(n)$ bits of communication to simply answer the question of whether two sets are disjoint, namely the set of BAM reads and the complement of the set of FASTQ reads.

Conflict of Interest: none declared.

References

Döring,A. *et al.* (2008) SeqAn an efficient, generic C++ library for sequence analysis. *BMC Bioinformatics*, **9.1**, 11.

Gudbjartsson,D.F. *et al.* (2015) Large-scale whole-genome sequencing of the Icelandic population. *Nat. Genet.*, **47**, 435–444.

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

Razborov,A.A. (1992) On the distributional complexity of disjointness. *Theor. Comput. Sci.* **106**, 385–390.

Rivest,R. (1992) RFC 1321: the MD5 message-digest algorithm. Internet Engineering Task Force. http://dl.acm.org/citation.cfm?id=RFC1321.