fastqsplitter Documentation

Release 2.0.0.dev0

Leiden University Medical Center

Aug 13, 2020

Contents

Table of contents

CHAPTER 1

Introduction

A simple application to split FASTQ files.

Fastqsplitter splits a fastq file over the specified output files evenly. It is similar to the GNU Coreutils split program, except that it is aware of the FASTQ four lines per record format. (Split works with one line per record.) It has support for compressed FASTQ files and can compress splitted FASTQ files on the fly.

Fastqsplitter uses a round-robin method to distribute the FASTQ records evenly across the output files. Alternatively it can distribute files sequentially, which is useful for reading from STDIN and the input size is unknown. Fastqsplitter can split such input in N files with a given maximum size.

This application does not work with multiline fastq sequences.

Fastqsplitter is fast because it only checks if the last record written to a file is a valid FASTQ record before starting to write to a new file. It assumes all records before that were valid. Since all downstream analysis tools (FastQC, cutadapt, BWA etc.) do check if the input is correct, extensive input checking in fastqsplitter was deemed redundant.

fastqsplitter uses the excellent xopen library by @marcelm. This determines by extension whether the file is compressed and allows for very fast compression and decompression of gzip files.

CHAPTER 2

Usage

```
usage: fastqsplitter [-h] [-p PREFIX] [-s SUFFIX]
                     (-n NUMBER | -o OUTPUT | -m MAX_SIZE) [-S]
                     [-c COMPRESSION_LEVEL] [-t THREADS_PER_FILE] [-P]
                     [input]
```

2.1 Positional Arguments

input The fastq file to be scattered.

Default: "/dev/stdin"

2.2 Named Arguments

-p, --prefix The prefix for the output files.

-s, --suffix The default suffix for the output files. The extension determines which compression is used. '.gz' for gzip, '.bz2' for bzip2, '.xz' for xz. Other extensions will use no compression.

Default: ".fastq.gz"

-n, --number Specify the number of output files which to split over. Fastq records will be distributed using a round-robin method.

-o, --output Scatter over these output files. Multiple -o flags can be used. The extensions determine which compression algorithm will be used. '.gz' for gzip, '.bz2' for bzip2, '.xz' for xz. Other extensions will use no compression. Fastq records will be distributed using a round-robin method.

-m, --max-size In round robin mode, determines the number of output files by dividing the input size by max size and distribute the fastq records in a round robin fashion over

these files. WARNING: if compression differs between input and output files, this will not work properly. In sequential mode this is the maximum number of bytes written to each output file. NOTE: This is the size *before* compression (if applied). As a rule of thumb multiply by 0.38 to get the actual filesize when using gzip compression.

-S, --sequential Do not use round-robin but create output files sequentially instead. Default when using stdin. Max size should be set.

Default: True

-c, --compression-level Only applicable when output files have a '.gz' extension. Default=1

Default: 1

-t, --threads-per-file Set the number of compression threads per output file. NOTE: more threads are only useful when using a compression level > 1. To use fastqsplitter in single-threaded mode choose 0. Default=1.

Default: 1

-P, --print Print output files to stdout for easier usage in scripts.

Default: False

Note: Fastqsplitter uses a separate process for reading the input file if it is compressed, doing the splitting as well as one seperate process per compressed output file. Fastqsplitter therefore always uses multiple CPU cores when working with compressed files.

CHAPTER 3

Example

3.1 Round-robin

With an input file `input_fastq.gz` of 2.3 GB. `fastqsplitter -i input_fastq.gz -n 3 -p split. -o .fq.qz` This will create `split.0.fq.gz`, `split.1.fq.gz` and `split.2.fq.gz`.

Fastqsplitter will read `input_fastq.gz`. The first block of records will go to `split.0.fq.gz`, the next block will go to `split.1.fq.gz`, etc.

This way the fastq reads are evenly distributed, with no positional bias in each output file.

3.2 Sequential

```
my_fastq_generating_program | fastqsplitter --max-size 10G -p my_fastq. -s
.fastq.gz
```

This will read from STDIN and write files that contain maximum 10GiB bytes. Note that a `.gz` suffix is used. The 10GiB bytes will be compressed and the output sizes will be smaller than 10 GiB. An unknown number of files will be generated.

Sequential mode can be forced with `-S` or `--sequential` flags.

CHAPTER 4

Performance comparisons

Following benchmarks were performed with a 5 million record FASTQ file (1.6 GiB) on a system with a Ryzen 5 3600 (6 core 12 threads) cpu with 32GB of ddr4-3200 ram.

The files were stored and written on a ramdisk created with `mount -t tmpfs -o size=12G myramdisk ramdisk`. This way IO was bottlenecked by memory bus speed instead of disk speed.

Benchmarks were performed using hyperfine.

4.1 Uncompressed

While uncompressed files are not used often in BioInformatics, they give a good impression of the speed of an algorithm by eliminating all the compression overhead. All benchmarks below split the 1.6 GB file in 3 files.

Fastqsplitter round-robin mode

Fastqsplitter sequential mode

GNU Coreutils split can also do sequential mode and give correct FASTQ records when a line number is chosen that is divisable by 4. The line number 7512140 gives also a 600M result file. So results are comparable.

Note that system times are within 10ms of each other. This signifies the time needed to write the files to the tmps and to read the input. User time is probably closer to the time spent in the algorithm.

The score is as follows: + Fastqsplitter round-robin: 174.9 ms user time. + Fastqsplitter sequential: 57.7 ms user time. + Gnu Coreutils split: 116.9 ms user time.

4.2 Compressed

Usually FASTQ files are compressed. Fastqsplitter uses xopen to call external programs which do the compression and decompression.

TODO: When igzip is patched and xopen supports igzip.

CHAPTER 5

Changelog

5.1 2.0.0-dev

- Redesigned CLI to make it much easier to use with streaming data.
- Added an algorithm that can handle streaming data with no known input size.
- Improved speed of the python algorithm. It is now 5 times faster than the old python algorithm. It is also 3 times faster than the cython algorithm from v1.2.0.
- The cython parts of the code have been deprecated for easier installation and better platform compatibility.

5.2 1.2.0

- Enable pure python fallback so package can be installed on all systems.
- Updated the documentation to reflect changes in speed because of the upstream improvements and the cythonizing of the algorithm in 1.1.0.
- Upstream contributions to xopen have made the reading of gzipped fastq files significantly faster. Newer versions of xopen are now added as a requirement.

5.3 1.1.0

- Enable the building of wheels for the project now that Cython extensions are used. Thanks to @marcelm for providing a working build script on https://github.com/marcelm/dnaio.
- Cythonize the splitting algorithm. This reduces the overhead of the application up to 50% over the fastest native python implementation. Overhead is all the allocated cpu time that is not system time.

  This means splitting of uncompressed fastqs will be noticably faster (30% faster was achieved during testing). When splitting compressed fastq files into compressed split fastq files this change will not be much faster since

all the gzip process will be run in a separate thread. Still when splitting a 2.3 gb gzipped fastq file into 3 gzipped split fastq files the speedup from the fastest python implementation was 14% in total cpu seconds. (Due to the multithreaded nature of the application wall clock time was reduced by only 3%).

5.4 1.0.0

- Added documentation for fastqsplitter and set up readthedocs page.
- Added tests for fastqsplitter.
- Upstream contributions to xopen have improved fastqsplitter speed.
- Initiated fastqsplitter.