fastqsplitter Documentation

Release 1.2.0

Leiden University Medical Center

Nov 18, 2019

Contents

Table of contents

CHAPTER 1

Introduction

A simple application to split FASTQ files.

The algorithm is a reimplementation from biopet-fastqsplitter. Fastqsplitter splits a fastq file over the specified output files evenly. Fastqsplitter will read groups of a 100 fastq files. For example if 3 output files are specified record 1-100 will go to file 1, 101-200 to file 2, 201-300 to file 3 , 301-400 to file 1 again etc. This ensures the output fastq files are of equal size with no positional bias in the output files.

Fastqsplitter is fast because it assumes each record is 4 lines. As a consequence this application does NOT work with multiline fastq sequences. Also input fastq records are NOT checked for being proper fastq records. Since all downstream analysis tools (FastQC, cutadapt, BWA etc.) do check if the input is correct, another input check 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.

Fastqsplitter has cythonized the files splitting algorithm which provides a speedup over the pure python implementation, especially when splitting to and from uncompressed fastq files. A python fallback is always available and fastqsplitter will default to it when the cython extension cannot be build or downloaded during the installation.

CHAPTER 2

Usage

```
usage: fastqsplitter [-h] -i INPUT -o OUTPUT [-c COMPRESSION_LEVEL]
                     [-t THREADS_PER_FILE] [--cython | --python]
```

2.1 Named Arguments

-i, --input The fastq file to be scattered.

-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.

-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. Default=1

Default: 1

--cython Use the cython version of the file splitting algorithm. (default)

Default: True

--python Use the python version of the file splitting algorithm.

Default: True

Note: Fastqsplitter uses a separate process for reading the input file, doing the splitting as well as one seperate process per output file. Fastqsplitter therefore always uses multiple CPU cores.

CHAPTER 3

Example

With an input file `input_fastq.gz` of 2.3 GB. `fastqsplitter -i input_fastq.gz -o split.1.fq.gz -o split.2.fq.gz -o split.3.fq.gz`

Fastqsplitter will read `input_fastq.gz`. The first 100 reads will go to `split.1.fq.gz`, read 101-200 will go to `split.2.fq.gz`, read 201-300 will go to `split.3.fq.gz`, read 301-400 will go to `split.1.fq.gz`, etc.

This way the fastq reads are evenly distributed, with a difference of maximum 100 reads between output files, and no positional bias in each output file.

CHAPTER 4

Performance comparisons

Comparing different modes of fastqsplitter and biopet-fastqsplitter. Biopet-fastqsplitter has only one mode: compression level 5, and an unknown number of threads per file.

Fastqsplitter runs with 1 thread per output file and compression level 1 by default. A comparison between default cython mode and python mode is in the table. For fair comparison with biopet-fastqsplitter, fastqsplitter was run with 4 threads per file (xopen default) and compression level 5. Since fastqsplitter starts several pigz and one gzip process the memory usage of these processes are included in the results.

This test case was run with a 2.3 GB input fastq file zipped. This was split over 5 output files.

The used test machine had 32 GB memory (2x16GB 2133mhz), an Intel core i7-6700 (4 cores, 8 threads) and a Sandisk X400 500gb SSD. Operating system: Debian 10.

The following table shows the average over 10 runs.

- real time = wall clock time
- user time = total cpu seconds spent by the application (useful to see the resource usage of multithreading)
- sys time = total cpu seconds spent by the kernel (for IO and other sys calls)

measurement	fastqsplitter (defaults)	fastqsplitter (python)	fastqsplitter -t 4 -c 5	biopet-fastqsplitter
real time	44.787s	45.702s	77.778s	102.300s
user time	162.272s	169.238s	459.051s	509.594s
sys time	9.825s	9.825s	11.078s	8.411s
max mem	24 MB	24 MB	42 MB	665 MB
max vmem	207 MB	207 MB	2.0 GB	11.0 GB
output files total size	2290 MB	2290 MB	2025 MB	2025 MB

The outcomes for the runs were fairly consistent with a +-3 second real time (wall clock) difference between runs.

4.1 Uncompressed

When splitting a 6.3 Gb uncompressed fastq file into 3 files which are written to /dev/null to eliminate I/O limitations the cython algorithm generates less overhead than the python algorithm.

However in most cases files will be split from compressed to compressed files making the difference much smaller (see table above.)

measurement	fastqsplitter (cython)	fastqsplitter (python)
real time	6.045s	10.345s
user time	4.907s	9.200s
sys time	1.137s	1.145s

CHAPTER 5

Changelog

5.1 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.

5.2 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%).

5.3 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.