PRINSEQ++, a multi-threaded tool for fast and efficient quality control and preprocessing of sequencing datasets

Vito Adrian Cantu[1], Jeffrey Sadural[2], and Robert Edwards[2]

[1]**Computational Science Research Center, San Diego State University, San Diego, 92182, USA.**
[2]**Department of Computer Science, San Diego State University, San Diego, 92182, USA.**

Corresponding author:
Robert Edwards[1]

Email address: redwards@mail.sdsu.edu

ABSTRACT

PRINSEQ++ is a C++ implementation of the very popular software prinseq-lite for quality control and preprocessing of sequencing datasets. PRINSEQ++ can run multi-threaded processes, which makes it more than 10 times faster than the original version. It can read from, and write to, compressed files, drastically reducing the use of hard-drive. PRINSEQ++ can filter, trim and reformat sequences by a variety of options to improve downstream analysis. PRINSEQ++ is freely available on GitHub (https://github.com/Adrian-Cantu/PRINSEQ-plus-plus) and runs on all Unix-like systems.

1 INTRODUCTION

As prices fall, high-throughput sequencing is being used in new ways and areas such as personalized medicine (Wilson et al., 2014) and recreational genomics (Evans, 2008). This brings about novel challenges for the techniques we use to analyze and draw conclusions from sequencing data, in particular speed and scalability.

Quality control is a crucial step in the analysis of sequencing datasets as low-quality sequences, sequence contamination and artifacts can eventually lead to erroneous conclusions. Most applications for quality control and preprocessing are written in high level programming languages such as Perl (prinseq-lite (Schmieder and Edwards, 2011)) or Java (fastQC (Andrews, 2010)) which are slower to execute and provide limited multi-threading support.

Since its publication in in early 2011, prinseq-lite has been cited more than 1500 times and downloaded more than 54000 times. In the same time interval, the number of bases in the Sequence Read Archive has grown 247x (from 74 Tbp to 18412 Tbp). It is clear that a new tool, one that has the usefulness of prinseq-lite while being drastically faster, is needed.

PRINSEQ++ implements all the functionality of the Prinseq-lite tool, adds some new features, but can run 16x times faster as it is written in C++ and can take advantage of multi-threading.

2 NEW FEATURES

2.1 Sequence duplication

Sequence duplication occur at different steps of the sequencing protocol (Gomez-Alvarez et al., 2009). Traditionally, duplicated sequences are hard to detect as the naive approach is to compare every single sequence to each other. This is problematic in a multi-threaded environment were each thread holds in memory a few sequences at most and cross talk between threads needs to be minimum in the interest of speed.

PRINSEQ++ uses a probabilistic data structure, Bloom filter (Bloom, 1970)(Partow, 2010), to identify duplicate sequences. A Bloom filter is bit-array where every sequence is transformed by several fast

(non-cryptographic) hash functions and the corresponding bits turned on. To see if a sequence is already in the filter (if it is duplicated) one only need to check the corresponding bits. Reading and writing from a bloom filter is fast and can be done asynchronously.

2.2 Parallelization

Most parallelization models, like OpenMP, MPI or Cuda, require the user to know the size and shape of the input a priori. Counting the number of sequences in a FASTA or FASTQ file requires reading it completely, which is slow and non trivial (especially so for compressed files). PRINSEQ++ uses POSIX threads (pthreads), an application programming interface (API) designed to allow maximum freedom to developers of multi-threaded applications.

With the exception of sequence duplication, the quality control and preprocesing is independent for each sequence pair. Each thread performs all necessary operations on one sequence pair at the time. This includes: reading from file, uncompressing if necessary, checking for duplicates and other filters, compressing if necessary, and writing to the corresponding output file if the read pair passes all filters. This model drastically reduces run-time, is input size agnostic, and uses little memory.

2.3 Speedup

Threads	speedup over prinseq-lite	speedup over PRINSEQ++
1	1.98 x	1 x
2	3.77 x	1.93 x
4	7.26 x	3.7 x
8	12.96 x	6.62 x
16	16.47 x	8.39 x

Table 1. Speedup of multi-threaded PRINSEQ++ over prinseq-lite and single-threaded PRINSEQ++ on the same computer, for the same dataset, using different numbers of threads.

To assess the effect of increasing the number of threads on speed and the speedup of PRINSEQ++ over prinseq-lite, we measured run-time of prinseq-lite and PRINSEQ++ on several FASTQ pair files of different sizes. A pair of FASTQ files from a metagenomic sample were downloaded from the sequence read archive (Run:SRR7091319). The FASTQ files were cut into files of 1, 2, 3, 4, 5, 10, 15, 20, 25, 30 millions read pairs. PRINSEQ++ and prinseq-lite were run on those files with equivalent filtering options (”min_len 100 -min_gc 40 -max_gc 60 -lc_method entropy -lc_threshold 90” for prinseq-lite and ”-min_len 100 -min_gc 40 -max_gc 60 -lc_entropy=0.9” for PRINSEQ++).

Run-time was measured using GNU time 1.7 on a 24 cores Intel Xeon CPU X5650 running at 2.67GHz with 189Gb of RAM. Each measurement was done three times and the mean time and 0.95 confidence interval were plotted on figure 1. Table 1 shows the speedup of multi-threaded PRINSEQ++ over prinseq-lite and over single-threaded PRINSEQ++.

There are two main reasons why the speedup don’t scales linearly with the number of threads for PRINSEQ++. There is an small overhead in creating a thread and, more importantly, input and output files need to be accessed synchronously. A thread cannot write or read if another thread is doing so, and must wait for it to finish. As the number of threads increases this happens more often and more time is spent waiting for access to files. Additionally, there is little advantage in using more threads than the number of cores in the CPU, as this will cause multiple threads to run on the same core and share execution cycles.

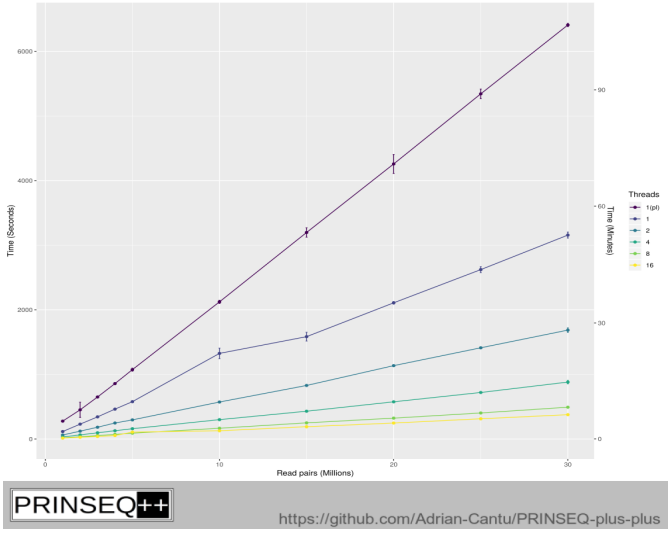

Figure 1. Runtime comparison, Run-time of prinseq-lite and PRINSEQ++ was measured on several FASTQ pair files of different sizes with equivalent options. PRINSEQ++ was run with different number of threads, prinseq-lite single-threaded. Mean speedup of PRINSEQ++ over prinseq-lite(pl) is 1.98x for single core and 16.47x for 16 cores. Error bars use a 0.95 confidence interval.

3 CONCLUSION

PRINSEQ++ is a fast and efficient and can significantly reduce the run-time of sequencing datasets analysis. This is critical in applications that are time-sensitive or where the amount of data is so large that slower method are not feasible. PRINSEQ++ has the capacity or reading from, and writing to compressed files without ever uncompressing the whole file, this drastically reduces use hard-drive use. PRINSEQ++ emulates prinseq-lite syntax, thous it can be easily added to any pipeline currently using prinseq-lite.

ACKNOWLEDGMENTS

We thank the PRINSEQ++ users for comments and suggestions and Dr. James Otto for computational support.

REFERENCES

Andrews, S. (2010). A quality control tool for high throughput sequence data. http://www.bioinformatics.babraham.ac.uk/projects/fastqc/.

Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426.

Evans, J. P. (2008). Recreational genomics; what's in it for you? *Genetics in medicine : official journal of the American College of Medical Genetics*, 10(10):709–710.

Gomez-Alvarez, V., Teal, T. K., and Schmidt, T. M. (2009). Systematic artifacts in metagenomes from complex microbial communities. *The ISME Journal*, 3(11):1314–1317.

Partow, A. (2010). General purpose hash function algorithms. http://www.partow.net/programming/hashfunctions/index.html.

Schmieder, R. and Edwards, R. (2011). Quality control and preprocessing of metagenomic datasets. *Bioinformatics*, 27(6):863–864.

Wilson, M. R., Naccache, S. N., Samayoa, E., Biagtan, M., Bashir, H., Yu, G., Salamat, S. M., Somasekar, S., Federman, S., Miller, S., and et al. (2014). Actionable diagnosis of neuroleptospirosis by next-generation sequencing. *New England Journal of Medicine*, 370(25):2408–2417.