btllib: A C++ library with Python interface for efficient sequence processing

Vladimir Nikolić [1] Parham Kazemi [1, 2] Lauren Coombe [1] Johnathan Wong [1] Amirhossein Afshinfard [1, 2] René Warren [1] Inanç Birol [1]

[1]Canada's Michael Smith Genome Sciences Centre at BC Cancer, Vancouver, BC V5Z 4S6, Canada
[2]Bioinformatics Graduate Program, The University of British Columbia, Vancouver, BC V6T 1Z4, Canada

Background

- Bioinformaticians often write one-off computer programs to perform a specific task instead of reusing existing code. This could be because the already existing code is either not easily found, the code is not well written or documented, or it is not efficient or general enough. This practice leads to lower quality and non-reusable code. As bioinformatics analyses increase in complexity and deal with ever more data, high quality code is needed for reliable and producible performance.
- The solution to this are well-designed and documented libraries, such as SeqAn [1] – a C++ library that implements algorithms and data structures commonly used in bioinformatics. Not all programmers are well versed in C++, so for users of widely used and accessible higher-level programming languages such as Python, Biopython [2] is available as a set of Python modules with implementations of commonly used algorithms.
- Here, we present the btllib library as an addition to this ecosystem with the goal of providing highly efficient, scalable, and ergonomic implementations of bioinformatics algorithms and data structures. The library is implemented in C++ with Python bindings available for a high-level interface.

Conclusions

- The goal of btllib is not to compete, but to complement other available libraries with applications in bioinformatics and genomics research.
- What sets it apart from other libraries is its focus on specialized algorithms with efficiency and scalability in mind as its aim is to enable sequence processing for large genomes.
- The library can be obtained using the **Conda** package manager:

```
conda install -c bioconda btllib
```

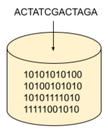
Funding

Genome BritishColumbia
GenomeCanada
National Institutes of Health
NSERC CRSNG

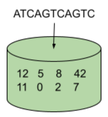
Design & Implementation

The library has an implementation of the following algorithms and data structures:

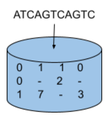
- **ntHash** [3]: A very efficient DNA/RNA rolling hash function, an order of magnitude faster than the best performing alternatives in typical use cases. The implementation includes hashing sequences with spaced seeds as well as feeding arbitrary nucleotides for implicit hash-based graph traversal.



- **Bloom filter**: A generic Bloom filter data structure. Thread safe and allows insertion of an array of hash values per element. Allows saving to disk with the associated metadata.

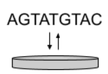


- **Counting Bloom filter**: A Bloom filter data structure that allows counting of the number of times an element has been inserted. Allows multithreaded insertion of elements while minimizing the effect of race conditions and preserving data integrity at a statistical level. This design was motivated by the need to maximize performance, as a fully thread safe counting Bloom filter would be unnecessarily slow. Allows saving to disk with the associated metadata.



- **Multi-index Bloom filter** [4]: A Bloom filter data structure that associates integer indices/IDs with the inserted elements. Like the counting Bloom filter, the race conditions are minimized for multithreaded insertion.

- **Indexlr**: An optimized and versatile minimizer calculator. For a given sequence file, Indexlr produces minimizers given a k-mer size and a window size. Optionally outputs minimizer sequence, sequence length, position, and strand. The library also includes an indexlr executable that produces minimizers from a given sequence file.



- **Sequence I/O**: SeqReader and SeqWriter classes provide efficient and flexible I/O for sequence files. SeqReader is capable of reading sequences in different formats such as FASTA and FASTQ including multiline, and SAM format. SeqReader also supports multiple threads to read in parallel, each thread receiving a copy of the sequence that can be modified as well as ad-hoc compression and decompression of the data in common formats (gzip, bzip2, xz, lrzip, zip, 7zip).

- **Utility functions**: Various functions for common tasks such as reverse complementation, string manipulation, and logging.

References

[1] Knut Reinert et al. "The SeqAn C++ template library for efficient sequence analysis: A resource for programmers". In: *Journal of Biotechnology* 261 (2017). Bioinformatics Solutions for Big Data Analysis in Life Sciences presented by the German Network for Bioinformatics Infrastructure, pp. 157–168. ISSN: 0168-1656. DOI: https://doi.org/10.1016/j.jbiotec.2017.07.017. URL: https://www.sciencedirect.com/science/article/pii/S0168165617315420.

[2] Peter J. A. Cock et al. "Biopython: freely available Python tools for computational molecular biology and bioinformatics". In: *Bioinformatics* 25.11 (Mar. 2009), pp. 1422–1423. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btp163. eprint: https://academic.oup.com/bioinformatics/article-pdf/25/11/1422/944180/btp163.pdf. URL: https://doi.org/10.1093/bioinformatics/btp163.

[3] Hamid Mohamadi et al. "ntHash: recursive nucleotide hashing". In: *Bioinformatics* 32.22 (July 2016), pp. 3492–3494. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btw397. eprint: https://academic.oup.com/bioinformatics/article-pdf/32/22/3492/19397492/btw397.pdf. URL: https://doi.org/10.1093/bioinformatics/btw397.

[4] Justin Chu et al. "Mismatch-tolerant, alignment-free sequence classification using multiple spaced seeds and multiindex Bloom filters". In: *Proceedings of the National Academy of Sciences* 117.29 (2020), pp. 16961–16968. DOI: 10.1073/pnas.1903436117. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.1903436117. URL: https://www.pnas.org/doi/abs/10.1073/pnas.1903436117.

Performance

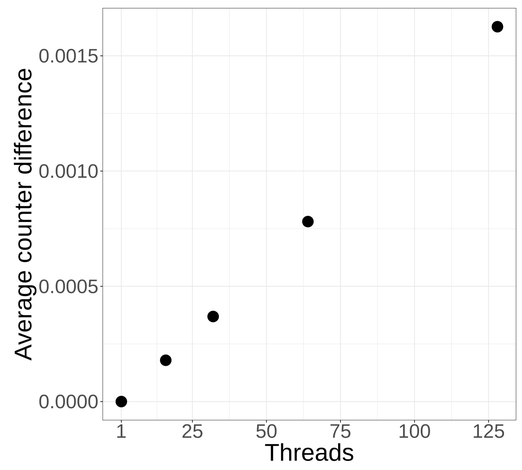


Figure 1. Average counter difference between a 10 Gigabyte counting Bloom filter with a single threaded and a multithreaded insertion of 250,000 long reads. The race condition mitigation mechanism minimizes the differences.

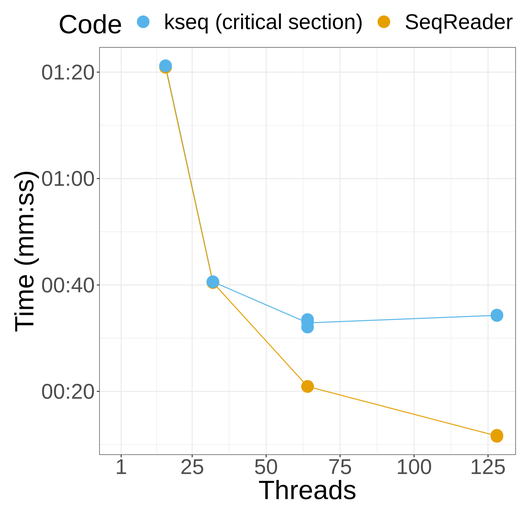


Figure 2. Wall-clock run time needed to load from disk and process 250,000 long reads with a simulated workload of 5ms per read with different number of CPU threads. The orange datapoints use btllib's sequence I/O and the blue data points implement a critical section approach using an efficient sequence reading code, kseq.