The Journal of Open Source Software

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

Vladimir Nikolić [1,2¶], **Parham Kazemi**[1,2], **Lauren Coombe** [1], **Johnathan Wong**[1], **Amirhossein Afshinfard** [1,2], **Justin Chu**[1,2], **René L. Warren** [1], **and 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 ¶ Corresponding author

DOI: 10.21105/joss.04720

Software

- Review
- Repository
- Archive

Editor: Frederick Boehm

Reviewers:

- @pjb7687
- @MaximLippeveld

Submitted: 04 August 2022
Published: 28 October 2022

License
Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

Summary

Bioinformaticians often do not have software engineering training or background, and software quality is not the top priority of research groups due to limited time and funding (Georgeson et al., 2019). Additionally, one-off scripts or code is frequently written to perform a specific task instead of reusing existing code. This could be because the pre-existing computer programming code is either not well written, widely available, documented, efficient nor general enough. This practice leads to lower quality and non-reusable code. As bioinformatics analyses are increasingly complex and deal with ever more data, high quality code is needed to handle the complexities of the analyses reliably and productively. The solution to this is well designed and documented libraries. For example, SeqAn (Reinert et al., 2017) is 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 (Cock et al., 2009) is available as a set of Python modules with implementations of commonly needed 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.

Statement of need

btllib is implemented in C++ with Python bindings available for a high level interface and ease of use. What sets it apart from other libraries is its focus on specialized algorithms with efficiency and scalability in mind as its origins are sequence processing for large genomes. The goal of btllib is not to compete, but to complement other available libraries.

btllib has helped enable bioinformatics scientific publications, including ntJoin (Coombe et al., 2020) and LongStitch (Coombe et al., 2021). Thanks to its focus on scalable algorithms and efficient implementation, btllib is a valuable tool for genomic applications, especially in the context of large genomes.

Design & implementation

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

- **ntHash** 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. Code adapted from the ntHash publication (Mohamadi et al., 2016). The following example produces 4 hashes per 6-mer for the seq sequence:

```
std::string seq = "ACTAGCTATGC";
int hash_num = 4;
int k = 6;
btllib::NtHash nthash(seq, hash_num, k);
while (nthash.roll()) {
    for (int i = 0; i < hash_num; i++) {
        std::cout << nthash.hashes()[i] << '\n';
    }
}
```

- **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 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. Figure 1 A) shows the effect of race condition mitigation. Allows saving to disk with the associated metadata. The following example stores all 6-mers from seq into the counting Bloom filter and then checks for their presence:

```
std::string seq = "ACTAGCTATGC";
int hash_num = 4;
int k = 6;
int bytes = 1024;
btllib::KmerCountingBloomFilter8 kcbf(bytes, hash_num, k);
kcbf.insert(seq);
assert(kcbf.contains(seq) == seq.size() - k + 1);
```

- **Multi-index Bloom filter** 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. Code adapted from the Multi-index Bloom filter publication (Chu et al., 2020).
- **Indexlr** An optimized and versatile minimizer calculator. For a given sequence file, Indexlr produces minimizer hash values 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. Code adapted from the Physlr publication (Afshinfard et al., 2022).
- **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. The format is automatically detected even without file extension or if the data is piped. 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). Figure 1 B) shows the scalability of using multiple threads to load and process sequences. The following example demonstrates the ease of use of SeqReader in a multithreaded environment using OpenMP. The sequences are loaded from my_reads.fq.gz in a mode optimized for long reads:

```
int flags = btllib::SeqReader::Flag::LONG_MODE;
btllib::SeqReader reader("my_reads.fq.gz", flags);
#pragma omp parallel
for (const auto record : reader) {
    std::cout << record.seq << '\n';
```

```
}
```

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

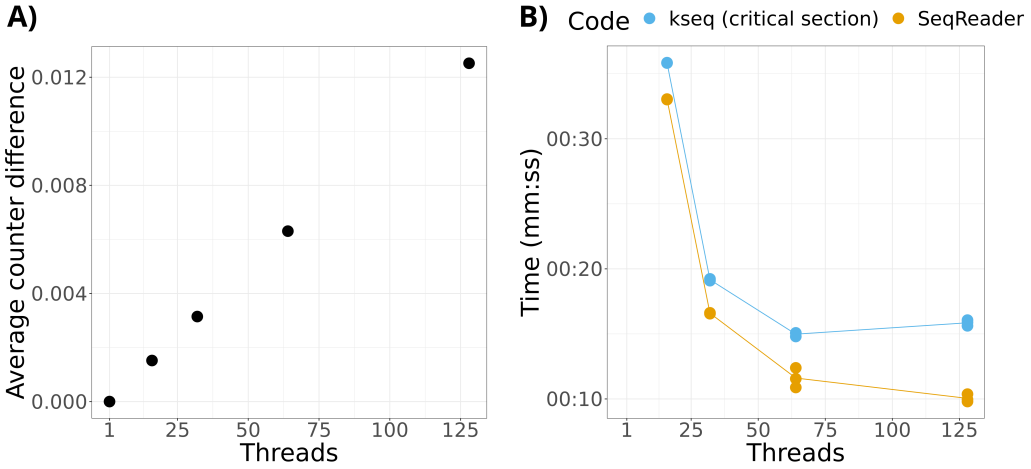

Figure 1: A) The average counter difference between 3GiB Counting Bloom Filters with single threaded and multithreaded insertion of k-mers of 50,000 long reads. The single threaded version does not suffer from race conditions and thus has the ground truth. The more threads we add the more visible is the effect of race conditions, but thanks to the mitigation mechanism, the differences are small. **B)** Wall-clock time it takes to read in and run a simulated workload of 2ms per read on 250,000 long reads. The blue data points use efficient sequence reading code, kseq (Li, 2016), surrounded by a naive implementation of a critical section, while the orange data points use SeqReader. Unlike the critical section implementation which plateaus after a number of threads, SeqReader scales well and keeps benefitting even from higher number of threads. SeqReader performs particularly well if used by multiple threads when loading long reads, as their length increases the time spent in I/O and thus in the critical section as well.

Acknowledgements

This work was supported by Genome BC and Genome Canada [281ANV]; the National Institutes of Health [2R01HG007182-04A1]; and the Natural Sciences and Engineering Research Council of Canada. The content of this paper is solely the responsibility of the authors, and does not necessarily represent the official views of the funding organizations.

References

Afshinfard, A., Jackman, S. D., Wong, J., Coombe, L., Chu, J., Nikolic, V., Dilek, G., Malkoç, Y., Warren, R. L., & Birol, I. (2022). Physlr: Next-generation physical maps. *DNA*, *2*(2), 116–130. https://doi.org/10.3390/dna2020009

Chu, J., Mohamadi, H., Erhan, E., Tse, J., Chiu, R., Yeo, S., & Birol, I. (2020). Mismatch-tolerant, alignment-free sequence classification using multiple spaced seeds and multiindex bloom filters. *Proceedings of the National Academy of Sciences*, *117*(29), 16961–16968. https://doi.org/10.1073/pnas.1903436117

Cock, P. J. A., Antao, T., Chang, J. T., Chapman, B. A., Cox, C. J., Dalke, A., Friedberg, I., Hamelryck, T., Kauff, F., Wilczynski, B., & Hoon, M. J. L. de. (2009). Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics*, *25*(11), 1422–1423. https://doi.org/10.1093/bioinformatics/btp163

Coombe, L., Li, J. X., Lo, T., Wong, J., Nikolic, V., Warren, R. L., & Birol, I. (2021). LongStitch: High-quality genome assembly correction and scaffolding using long reads. *BMC Bioinformatics*, *22*(1). https://doi.org/10.1186/s12859-021-04451-7

Coombe, L., Nikolić, V., Chu, J., Birol, I., & Warren, R. L. (2020). ntJoin: Fast and lightweight assembly-guided scaffolding using minimizer graphs. *Bioinformatics*, *36*(12), 3885–3887. https://doi.org/10.1093/bioinformatics/btaa253

Georgeson, P., Syme, A., Sloggett, C., Chung, J., Dashnow, H., Milton, M., Lonsdale, A., Powell, D., Seemann, T., & Pope, B. (2019). Bionitio: demonstrating and facilitating best practices for bioinformatics command-line software. *GigaScience*, *8*(9). https://doi.org/10.1093/gigascience/giz109

Li, H. (2016). *Seqtk*. https://github.com/lh3/seqtk.

Mohamadi, H., Chu, J., Vandervalk, B. P., & Birol, I. (2016). ntHash: recursive nucleotide hashing. *Bioinformatics*, *32*(22), 3492–3494. https://doi.org/10.1093/bioinformatics/btw397

Reinert, K., Dadi, T. H., Ehrhardt, M., Hauswedell, H., Mehringer, S., Rahn, R., Kim, J., Pockrandt, C., Winkler, J., Siragusa, E., Urgese, G., & Weese, D. (2017). The SeqAn c++ template library for efficient sequence analysis: A resource for programmers. *Journal of Biotechnology*, *261*, 157–168. https://doi.org/10.1016/j.jbiotec.2017.07.017