Correspondence | Published: 19 July 2021

A Python-based programming language for high-performance computational genomics

Ariya Shajii, Ibrahim Numanagić, Alexander T. Leighton, Haley Greenyer, Saman Amarasinghe & Bonnie Berger

Nature Biotechnology **39**, 1062–1064 (2021)

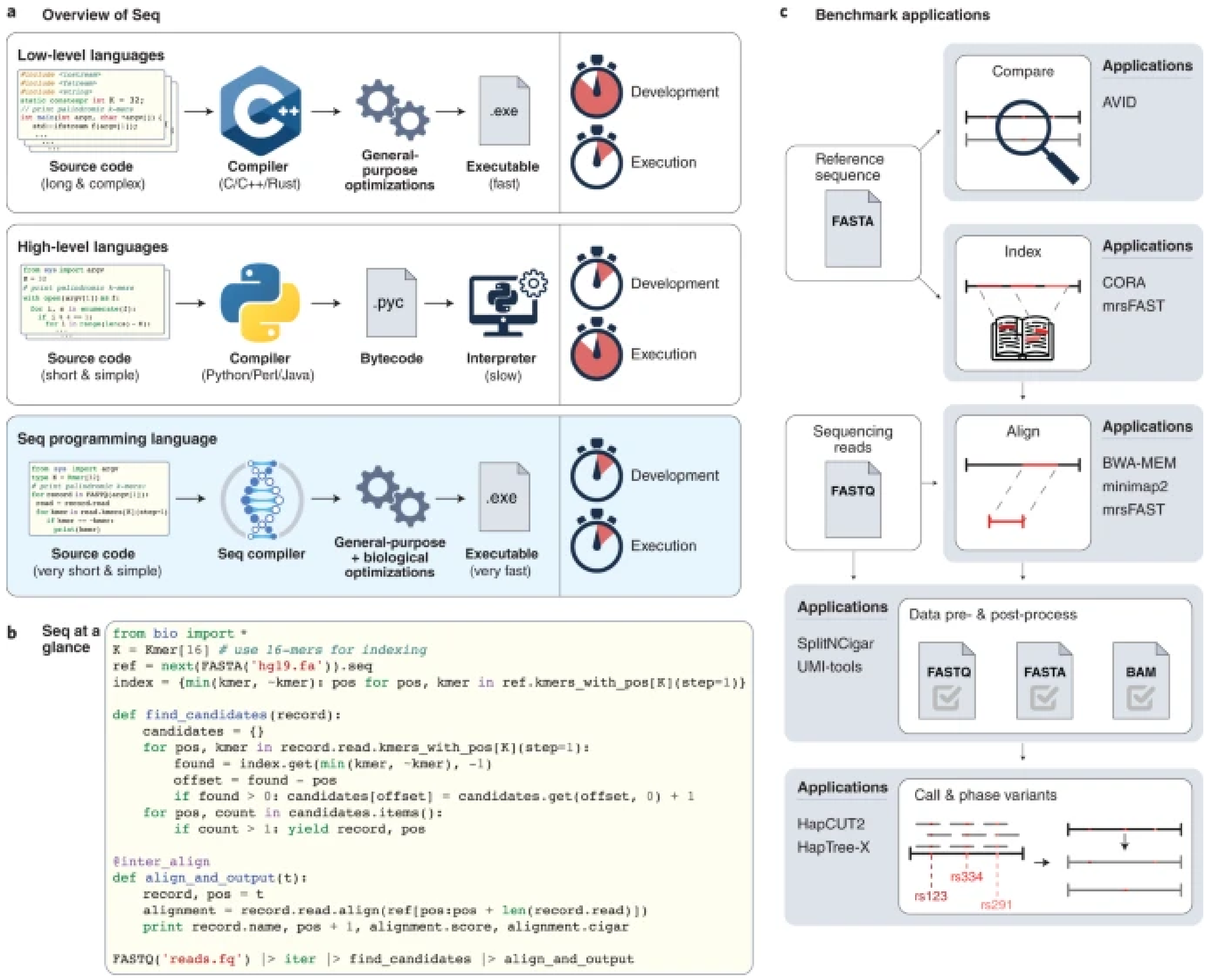
9474 Accesses | **1** Citations | **242** Altmetric | Metrics

To the Editor — The vast growth of next-generation sequencing data has provided us with a new understanding of many biological phenomena. As sequencing technologies evolve, sequencing datatypes (such as standard Illumina short reads, PacBio long reads or 10x Genomics barcoded reads) typically require new implementations of corresponding computational analysis techniques, necessitating software that is not only computationally efficient, but also quick to develop and easy to maintain so as to enable rapid adaptations to new kinds of data.

However, developing an efficient software tool requires domain expertise in performance engineering, computational modeling, and the ability to translate biological assumptions into algorithm and software optimizations[1]. As a result, most high-performance genomics software is highly application specific, difficult to understand, and difficult to maintain. These factors collectively have sparked a reproducibility crisis in the fields of computational biology and life sciences in general[2,3,4]. Ultimately, researchers in the field lack good tools for developing and updating software; one typically must choose between a software ecosystem that allows rapid development at the expense of performance and scalability (for example, Python or R) or low-level languages that have higher performance but are harder to develop and maintain (for example, C, C++ or Rust)[5]. Existing solutions that try to fill the void between these extremes (for example, Matlab or Julia) are primarily geared toward numerical computing rather than computational genomics, and the few attempts that have been made at simplifying the development of bioinformatics software (for example, Rust-Bio[6], SeqAn[7,8], BioJulia[9] or Biopython[10]) are based on languages that are either unfamiliar[11], complex and hard to program, or too high level to attain good performance.

Here, we introduce Seq (Fig. 1a), a high-performance, Python-based, compiled programming language geared toward biology that combines the ease of use of high-level languages like Python or Matlab with the runtime performance of low-level languages like C or C++. Seq shares the syntax and semantics of the widely used Python language, to which it adds genomics-specific language constructs, types and optimizations invisible to the user. We chose Python as a base language because of its succinct, clear syntax and widespread adoption; it is one of the most commonly used languages in bioinformatics, as well as in programming in general[11]. Moreover, Seq's first-class support for C and Python interoperability makes it easy to integrate into existing pipelines. Unlike compilers for more general-purpose languages, Seq's compiler uses fundamental genomics building blocks such as sequences and *k*-mers, as well as operations on them, such as hashing, indexing and alignment, to increase performance. These domain-specific optimizations and program transformations are performed automatically by the Seq compiler (Supplementary Fig. 5), and in Seq they often require just a single additional line of code (Fig. 1b), whereas implementation by hand would require large-scale code changes and substantial testing time to incorporate into existing tools. Many of these optimizations require analyzing the program as a whole and are thus out of reach for software libraries, several of which have been developed for a number of languages[6,10,12]. Thus, Seq enables users to write high-level, Python-type code without worrying about low-level implementation details or optimizations, which the compiler handles internally (Supplementary Notes 1, 3 and 4). Seq aims to fill the same role in computational biology as Matlab has in numerical computing by becoming an accessible portal to the world of high-performance genomics.

Fig. 1: The Seq programming language.



```
from bio import *
K = Kmer[16] # use 16-mers for indexing
ref = next(FASTA('hg19.fa')).seq
index = {min(kmer, ~kmer): pos for pos, kmer in ref.kmers_with_pos[K](step=1)}

def find_candidates(record):
    candidates = {}
    for pos, kmer in record.read.kmers_with_pos[K](step=1):
        found = index.get(min(kmer, ~kmer), -1)
        offset = found - pos
        if found > 0: candidates[offset] = candidates.get(offset, 0) + 1
    for pos, count in candidates.items():
        if count > 1: yield record, pos

@inter_align
def align_and_output(t):
    record, pos = t
    alignment = record.read.align(ref[pos:pos + len(record.read)])
    print record.name, pos + 1, alignment.score, alignment.cigar

FASTQ('reads.fq') |> iter |> find_candidates |> align_and_output
```

a, Conceptual comparison of Seq, Python and C++. Seq combines the high performance of C++ with the programming ease and clarity of Python, by virtue of domain-specific compiler optimizations that are hidden from the user. **b**, Example Seq code for a simple *k*-mer-based read mapper. **c**, Schematic of standard genomics pipeline and those state-of-the-art tools compared to Seq.

To demonstrate Seq's versatility, we reimplemented eight popular genomics tools in Seq, spanning key tasks in the genomics analysis pipeline (Fig. 1c and Supplementary Note 2), such as the finding of super-maximal exact matches, or SMEMs (BWA-MEM[13]), genome homology table construction (CORA[14]), Hamming distance–based all-mapping (mrsFAST[15]), long-read alignment (minimap2[16]), single-cell data preprocessing (UMI-tools[17]), SAM/BAM postprocessing (GATK[18]), global sequence alignment (AVID[19]) and haplotype phasing (Haptree-X[20,21]). We observe substantial runtime improvements over the original, hand-optimized C or C++ implementations while using less code than these and maintaining identical accuracy to them. Specifically, Seq achieved up to an order of magnitude runtime improvement in data pre- and postprocessing, genome index construction and numerical downstream analysis, a twofold improvement in FM-index querying, and up to threefold improvements in Smith–Waterman alignment and Hamming distance–based read mapping, with similar memory usages to those of the hand-optimized, original versions (Supplementary Fig. 1 and Supplementary Tables 5–8). The Seq implementations had up to an order of magnitude fewer lines of code than their C or C++ counterparts, making them easier to develop and less prone to bugs[22]. We also compared Seq to several high-performance genomics libraries, including SeqAn, Rust-Bio and BioJulia, and observed substantial speedups in every case, with over an order of magnitude in some cases (Supplementary Tables 3 and 5).

Seq is open source and freely available (https://seq-lang.org) and can be tried online (https://seq-lang.org/demo) with no downloads required. The implementations mentioned above, as well as the experimental notebook, are available on GitHub (https://github.com/seq-lang/seq-benchmarks). A practical Seq tutorial is also included in Supplementary Note 1. A preliminary conference version that showcased Seq and focused on specifics of the programming language itself, without genomics applications, appeared in OOPSLA[23].

Seq combines high-performance and scalability with a familiar, high-level and easy-to-maintain programming interface. It is the first step toward a rich ecosystem providing software development environments, visualization tools and debugging support, all designed with the domain of bioinformatics in mind. By offering biologists, bioinformaticians and other researchers a scalable way to prototype, experiment and analyze large biological datasets through a familiar, high-performance language, we hope that Seq will act as a catalyst for scientific discovery and innovation.

References

1. Yu, Y. W., Daniels, N. M., Danko, D. C. & Berger, B. *Cell Syst.* **1**, 130–140 (2015).

2. Peng, R. D. *Science* **334**, 1226–1227 (2011).

3. Baker, M. *Nature* **533**, 452–454 (2016).

4. Lee, R. S. & Hanage, W. P. *Lancet Microbe* https://doi.org/10.1016/S2666-5247(20)30028-8 (2020).

5. Perkel, J. M. *Nature* **588**, 185–186 (2020).

6. Köster, J. *Bioinformatics* **32**, 444–446 (2016).

7. Döring, A., Weese, D., Rausch, T. & Reinert, K. *BMC Bioinformatics* **9**, 11 (2008).

8. Reinert, K. et al. *J. Biotechnol.* **261**, 157–168 (2017).

9. Ward, B. J. *BioJulia* https://biojulia.net (accessed 19 November 2020).

10. Cock, P. J. et al. *Bioinformatics* **25**, 1422–1423 (2009).

11. Russell, P. H., Johnson, R. L., Ananthan, S., Harnke, B. & Carlson, N. E. *PLoS One* **13**, e0205898 (2018).

12. Stajich, J. E. et al. *Genome Res.* **12**, 1611–1618 (2002).

13. Li, H. Preprint at https://arxiv.org/abs/1303.3997 (2013).

14. Yorukoglu, D., Yu, Y. W., Peng, J. & Berger, B. *Nat. Biotechnol.* **34**, 374–376 (2016).

15. **15.** Hach, F. et al. *Nucleic Acids Res.* **42**, W494–W500 (2014).

16. Li, H. *Bioinformatics* **34**, 3094–3100 (2018).

17. Smith, T., Heger, A. & Sudbery, I. *Genome Res.* **27**, 491–499 (2017).

18. McKenna, A. et al. *Genome Res.* **20**, 1297–1303 (2010).

19. Bray, N., Dubchak, I. & Pachter, L. *Genome Res.* **13**, 97–102 (2003).

20. Berger, E. et al. *Nat. Commun.* **11**, 4662 (2020).

21. Berger, E., Yorukoglu, D. & Berger, B. *International Conference on Research in Computational Molecular Biology* 28–29 (Springer, 2015).

22. Abelson, H. & Sussman, G. J. *Structure and Interpretation of Computer Programs* (MIT Press, 1996).

23. Shajii, A., Numanagić, I., Baghdadi, R., Berger, B. & Amarasinghe, S. *Proc. ACM Program. Lang.* **3**, 125:1–125:29 (2019).

Author information

1. These authors contributed equally: Ariya Shajii, Ibrahim Numanagić.

Affiliations

1. Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA

Ariya Shajii, Ibrahim Numanagić, Alexander T. Leighton, Saman Amarasinghe & Bonnie Berger

2. Department of Computer Science, University of Victoria, Victoria, British Columbia, Canada

   Ibrahim Numanagić & Haley Greenyer

3. Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA, USA

   Alexander T. Leighton & Bonnie Berger

Corresponding authors

Correspondence to Saman Amarasinghe or Bonnie Berger.

Ethics declarations

Competing interests

The authors declare no competing interests.

Additional information

Peer review information *Nature Biotechnology* thanks Ivan Costa and Judith Zaugg for their contribution to the peer review of this work.

Supplementary information

Supplementary Information

Supplementary Notes 1–4, including Figs. 1–5 and Tables 1–8

Rights and permissions

Reprints and Permissions

About this article

Cite this article

Shajii, A., Numanagić, I., Leighton, A.T. *et al.* A Python-based programming language for high-performance computational genomics. *Nat Biotechnol* **39,** 1062–1064 (2021). https://doi.org/10.1038/s41587-021-00985-6

Published 19 July 2021 **Issue Date** September 2021

DOI https://doi.org/10.1038/s41587-021-00985-6

Share this article

Anyone you share the following link with will be able to read this content:

Get shareable link

Provided by the Springer Nature SharedIt content-sharing initiative

Subjects Data processing • Programming language • Software

Nature Biotechnology (*Nat Biotechnol*) ISSN 1546-1696 (online) ISSN 1087-0156 (print)