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

Published in final edited form as:

Nat Biotechnol. 2021 September ; 39(9): 1062–1064. doi:10.1038/s41587-021-00985-6.

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

Ariya Shajii[1,4], **Ibrahim Numanagić**[1,2,4], **Alexander T. Leighton**[1,3], **Haley Greenyer**[2], **Saman Amarashinghe**[1,*], **Bonnie Berger**[1,3,*]

[1]Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, MA 02139, USA

[2]Department of Computer Science, University of Victoria, Victoria, BC V8P 5C2, Canada

[3]Department of Mathematics, MIT, Cambridge, MA 02139, USA

[4]Co-first authors

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 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 replication crisis in the fields of computational biology and life sciences in general[2–4]. Ultimately, researchers in the field lack good tools for developing and updating software; one typically has to choose between a software ecosystem that allows rapid development at the expense of performance and scalability (e.g., Python or R) or low-level languages that have higher performance but are hard to develop and maintain (e.g., C, C++ or Rust)[5]. Existing solutions that try to fill the void between these extremes (e.g., MATLAB or Julia) are primarily geared towards numerical computing rather than computational genomics, and the few attempts that have been made at simplifying the development of bioinformatics software (e.g., 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.

[*]*Correspondence:* bab@mit.edu, saman@csail.mit.edu.

Competing interests

The authors declare that they have no competing financial interests.

Additional information

Editorial note: This article has been peer reviewed.

Here, we introduce Seq (Fig. 1a), a high-performance, Python-based, compiled programming language geared towards 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 due to its succinct, clear syntax and widespread adoption; it is one of the most commonly used languages in bioinformatics, and 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 like 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. S5) and in Seq, they often require just a single additional line of code (Fig. 1b shows example Seq code for a simple *k*-mer based read mapper), whereas implementation by hand would require large-scale code changes and substantial testing time to incorporate into existing tools. Many of these optimizations require a global view of the program and are thus out of reach for software libraries, several of which have been developed for several languages[6,10,12]. Thus, Seq enables users to write high-level, Pythonic code without worrying about low-level implementation details or optimizations, which the compiler handles internally (see 'A short, practical guide to Seq' in Supplementary Information). 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.

To demonstrate Seq's versatility, we re-implemented 8 popular genomics tools in Seq, spanning key tasks in the genomics analysis pipeline (Fig. 1c), such as finding 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 pre-processing (UMI-tools[17]), SAM/BAM post-processing (GATK[18]), global sequence alignment (AVID[19]), and haplotype phasing (HapTree20[20,21]). We observe substantial runtime improvements over the original, hand-optimized C/C++ implementations while using less code than, and maintaining identical accuracy to, the original versions. Specifically, Seq achieved up to an order of magnitude runtime improvement in data pre- and post-processing, genome index construction and numerical downstream analysis, a two-fold improvement in FM-index querying, and up to three-fold improvements in Smith-Waterman alignment and Hamming distance-based read mapping, with similar memory usages to the hand-optimized, original versions (Supplementary Fig. S1 and Tables S1–S8). 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 (over an order of magnitude in some cases; Supplementary Note 2).

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 the Supplementary Information (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 is the first programming language tailored for bioinformatics that combines high-performance and scalability with a familiar, high-level and easy-to-maintain programming interface. It is the first step towards 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.

Supplementary Material

Refer to Web version on PubMed Central for supplementary material.

References

1. Yu YW, Daniels NM, Danko DC & Berger B. Cell Systems 1, 130–140 (2015). [PubMed: 26436140]
2. Peng RD Science 334, 1226–1227 (2011). [PubMed: 22144613]
3. Baker M. Nature News 533, 452 (2016).
4. Lee RS &Hanage WP The Lancet Microbe (2020).
5. Perkel JM Nature 588, 185–186 (2020). [PubMed: 33262490]
6. Köster J. Bioinformatics 32, 444–446 (2015). [PubMed: 26446134]
7. Döring A, Weese D, Rausch T. & Reinert K. BMC Bioinformatics 9, 11 (2008). [PubMed: 18184432]
8. Reinert K. et al. J. Biotechnol 261, 157–168 (2017). [PubMed: 28888961]
9. Ward BJ BioJulia. (accessed November 19, 2020).
10. Cock PJ et al. Bioinformatics 25, 1422–1423 (2009). [PubMed: 19304878]
11. Russell RLAA, Pamela H. AND Johnson PLOS ONE 13, 1–19 (2018).
12. Stajich JE et al. Genome Res. 12, 1611–1618 (2002). [PubMed: 12368254]
13. Li H. arXiv preprint arXiv:1303.3997 (2013)
14. Yorukoglu D, Yu YW, Peng J. & Berger B. Nat. Biotechnol 34, 374–376 (2016). [PubMed: 27054987]
15. Hach F. et al. Nucleic Acids Res. 42, W494–W500 (2014). [PubMed: 24810850]
16. Li, HBioinformatics34, 3094–3100 (2018).
17. Smith TS, Heger A. &Sudbery I. Genome Res. 27, 491–499 (2017). [PubMed: 28100584]
18. McKenna A. et al. Genome Res. 20, 1297–1303 (2010). [PubMed: 20644199]
19. Bray N, Dubchak I. &Pachter L. Genome Res. 13, 97–102 (2003). [PubMed: 12529311]
20. Berger E. et al. Nature Comm. 11, 1–9 (2020).
21. Berger E, Yorukoglu D. & Berger B. International conference on research in computational molecular biology 28–29 (Springer, Warsaw, 2015).
22. Abelson H. & Sussman GJ Structure and Interpretation of Computer Programs. (MIT Press, Cambridge, MA, 1996).
23. Shajii A, Numanagić I, Baghdadi R, Berger B. &Amarasinghe S. Proc. ACM Program. Lang 3, 125:1–125:29 (2019).

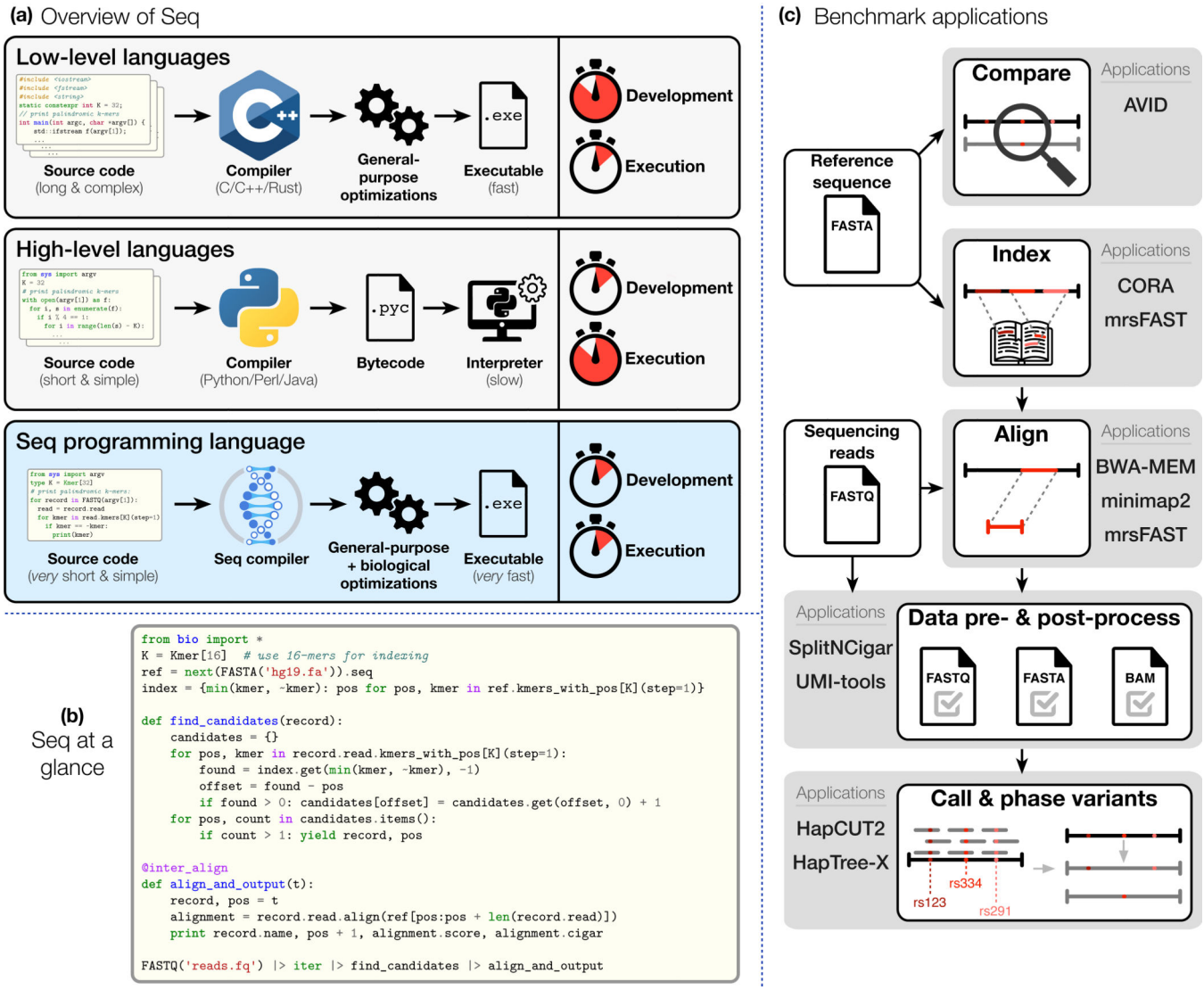

Figure 1.

(**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 implemented in Seq.