Cumhuriyet Sci. J., 45(4) (2024) 663-667
DOI: https://doi.org/10.17776/csj.1511642

Cumhuriyet Science Journal

| csj.cumhuriyet.edu.tr | Founded: 2002 ISSN: 2587-2680 e-ISSN: 2587-246X Publisher: Sivas Cumhuriyet University

A Comparative study of DNA Alignment Algorithms and Boosting Performance Using Different Compilation Strategies

Osman Doluca [1,a,*]

[1] *Department of Biomedical Engineering, Izmir University of Economics, Türkiye*
**Corresponding author*

Research Article

History
Received: 06/06/2024
Accepted: 03/12/2024

ABSTRACT

With the development of next generation sequencing technologies, the requirement of higher performance from DNA and Protein sequence alignment algorithms has become even greater. This work is a systematic comparison of different compilation strategies for two common DNA or Protein sequence alignment algorithms, Needleman-Wunsch and Smith-Waterman, using Python programming language. It aims to investigate the performance benefits of already widely used Biopython's pairwise alignment module versus different compilation approaches of an in-house software. It is shown that using Numba just-in-time compiler provide greater performance overall in comparison to PyPy and Cython compilers or the Biopython module. This work may increase the efficiency of software prototyping where large-scale sequence alignment is necessary.

Keywords: DNA/Protein sequence alignment, Python, Just-in-time compilers.

[a] *osman.doluca@ieu.edu.tr* *https://orcid.org/0000-0003-0412-6148*

Introduction

With the development of next generation sequencing (NGS) the amount of available biological data, in terms of genome or exome sequences, has been dramatically increasing. The alignment methods have been becoming more and more relevant as the necessity to process the raw NGS data or to investigate the similarities between sequences for the discovery of their functional properties has been increasing. As the current NGS sequencing technologies are producing relatively short fragments of sequence information, ranging between 50 to 1000 bp, their alignment is quite computation exhausting process since a single stretch of continuous DNA sequence may be up to hundreds of millions bp long. [1] While the NGS technologies improve the count of the fragments read, the reading length tend to get even shorter, relying even more to the computation for their alignment to produce the genome [2] with the exception of nanopore sequencing which is yet to become applicable. [3]

Any solution to this problem has one of two approaches. One being, increasing the computational power, has long been market-driven endeavor. Unfortunately, as the Moore's law is approaching its limits and a stagnation in the increase of the processing power of chips is imminent, any hardware-based solution is shifting towards multi-core chips instead of faster chips. [4] The other solution is implementing alternative algorithms to the alignment problem. The two most commonly used algorithms, Needleman Wunsch (global) [5] and Smith Waterman (local) [6], and their variations [7] are known to provide the best alignment, but also the slowest as their implementations make use of multiple 2D matrices. New heuristic approaches are also developed however these approaches does not match the sensitivity of these two algorithms or their variations. [8-10]

New approaches for processing biological data often requires software prototyping and testing, and the majority of the data scientists rely on high-level programming languages such as Python. The less time spent coding and high readability of such high-level languages is the foremost reason behind this preference. [11] In comparison, while the low-level programming languages such as C provide better the execution speed, the high complexity deters the scientists from using it. However, especially when it comes to NGS data analysis, speed may be more of an issue than many other factors due to sheer size of alignments required. The solution is often sought in accelerating the alignment algorithms. [12–14] Speed improvement is possible at the interpreter level, either through using modules that exploit pre-compiled libraries or compilation of the code on the go, also known as just-in-time compilation. [15] There has been a number of approaches for improving the performance of SW algorithm, mostly through parallelism by performing calculations on FPGAs or GPU. [16-18] However, these are not available as modules for mainstream programming languages.

Here we implement the global and local alignment algorithms including the affine gap extension developed by Gotoh et al. [5–7] We compare the performance improvements when rewritten using Numpy module, or compiled using Cython, Numba or PyPy. The performance improvements were also compared with pairwise2 module of Biopython library. The code is made available online at http://github.com/odoluca/Fast-NW-and-SW-Pairwise-alignment-using-numba-JIT/

Method

The affine gap penalty [7] variation of original global [5] and local [6] alignment algorithms was written using python 3.6+. For each local and global algorithms two other variations were written, one of which uses single scores for matches and mismatches or a substitution matrix. Substitution matrix is especially necessary for protein alignments as transition between amino acids do not have equal probabilities. Each of the four methods were rewritten in two forms; one for discovery of only the highest score while the other uses backtracking to reports the best (or one of the best) alignment(s). Together these variations yielded eight different methods. (Table 1)

Each method was interpreted or compiled using different approaches. 1) Pure python approach uses only python 3.6+ syntax and built-in types. No external module was used. 2) Numpy approach incorporates ndarrays from Numpy module (1.14.2) for all matrices. 3) Cython (v0.29.12) was used to compile pure python or Numpy-using code using cythonize method and "build_ext" argument to build all extensions. 4) Just-In-Time compilation using Numba module (v0.44.1) was used with both pure python or with Numpy. Because the Numba does not accept strings in "nopython" mode, all methods were written to accept two lists of integers as sequences to be aligned instead of lists of characters, as in strings, where each character indicates nucleotide or amino acid residues. An additional method is written to convert any protein, DNA or RNA sequence into a list of integers. All numba methods were compiled just-in-time in "nopython" and "cached" modes for optimum performance using "@jit(nopython=True, cached=True)" decoration. 5) finally, a 32-bit pypy compiler (v7.1.1) was used with pure python code to compare. All algorithms were written in two variations, "score only" and "backtrack". The first one is where only scores are calculated, and the latter is where the best alignment is constructed by tracing back the path. Backtracking requires three additional matrices to keep track of the path. Additionally, Biopython's pairwise2 alignment was used for comparison to view the performance improvements. Biopython was also tested in "score only" mode as well as "alignment" mode for equivalent comparison.

Performance was measured using timeit module, aligning sequences with varying percentages of similarity and varying sizes of sequences. For each pairwise alignment, a sequence was generated randomly at first. The other sequence was obtained by introducing a number of mutations until a given percent similarity is obtained. Each mutation was introduced with 80% chance for the substitution, 10% for the insertions and deletions each. The percent similarity was calculated as a ratio of global alignment score of the alignment of the two sequences to the alignment of max possible sequence of any two sequence with the same lengths. The global alignment was performed using +1 for matches, -1 for mismatches and gaps. For each category a thousand sequence pairs were aligned and total processing times were found as summation. All tests were performed using timeit module with garbage collection off to increase accuracy. All module imports or any preprocessing is left out of performance testing and measurements were done only during sequence alignments. Every performance test was repeated five times and the best of five was reported. All tests were performed at AMD 1950X machine equipped with 128 GB ECC RAM with ECC-mode on and locked to the same core. All algorithms were previously run using random sequences and compared with Biopython's pairwise2 module to confirm that the same results were produced before performance testing.

Results and Discussion

With varying features included in the algorithm eight different methods were written and tested in this work. The list of these methods and their features are listed in Table 1. Briefly substitution matrix feature enables different penalties for substitution between different residues. This is preferred especially if the mutations between particular residues is more common or expected than others. Another feature is called "backtracking" which enables production of a final alignment of the two sequences. Alternative, "score only" mode reports only the score of the best alignment which may be used as a measure of sequence similarity. This is often useful for construction of phylogenetic trees. Backtracking requires keeping of three additional matrices with a size of (n x m) with n and m being the lengths of the two sequences.

Table 1. Methods used in this work and their abbreviations.

algorithm	substitution matrix	backtracking	method abbreviation
global	No	No	globalms
local	No	No	localms
global	Yes	No	globalds
local	Yes	No	localds
global	No	Yes	globalms
local	No	Yes	localms
global	Yes	Yes	globalds
local	Yes	Yes	localds

The methods were written and executed with varying modules and compilers. Not all compilers were compatible, as a result, we have tested seven combinations of modules and compilers/interpreters. (Table 2., Figure 1.) As Cython or standard Python interpreter proved to be much slower in all cases, their performance evaluation is omitted, focusing on Biopython, PyPy and Numba.

The effect of sequence length. The biggest impact on the performance was observed to be the sequence length. In all the cases there was an exponential increase, close to the order of two as the size of the matrices (n x m) also increases in the order of two. This was independent of the algorithm or modules that were used. Interestingly only with Numba the order of the power was close to 1.5 which indicates that the matrix processing is not a bottleneck for Numba.

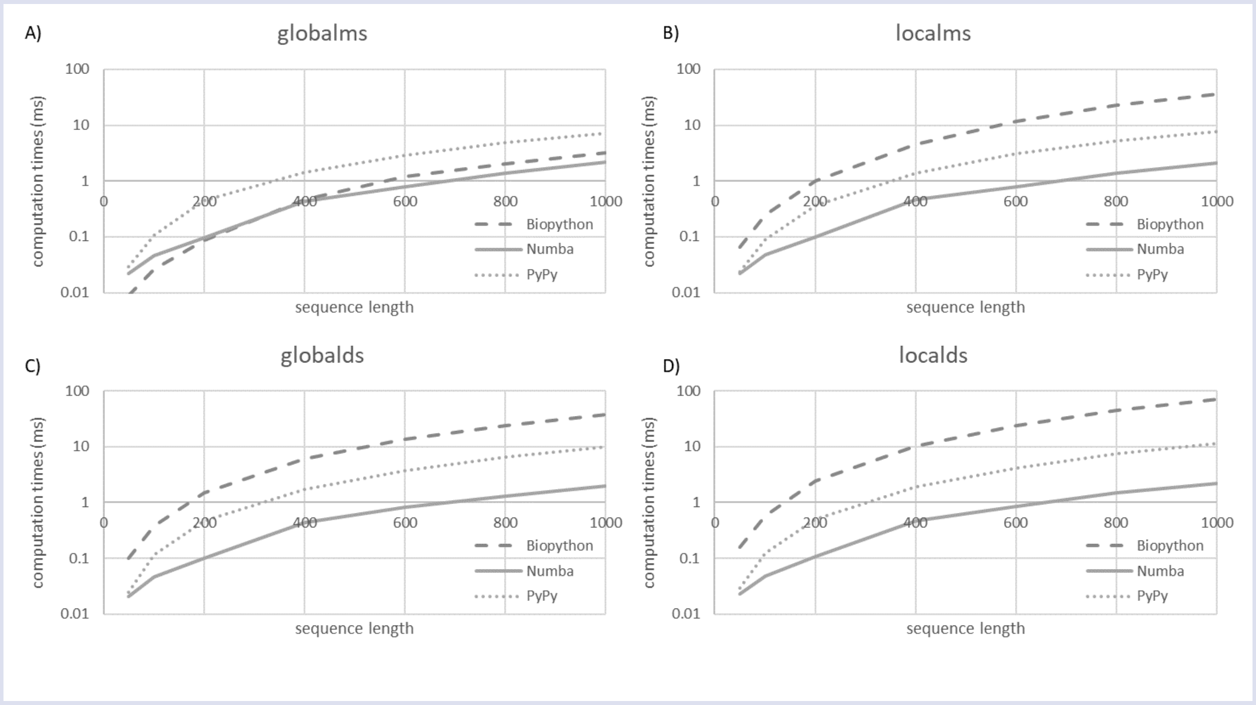

Figure 1. The computation times in milliseconds (ms) versus sequence length for globalms (A), localms (B), globalds (C) and localds (D) alignment methods without backtracking, using various compilation methods, Biopython using standard Python (dashed), In-house method with Numba compiler (light gray) and In-house method with PyPy compiler (dotted).

The effect of sequence similarity. On the other hand sequence similarities ranging from 20% to 80% did not seem to have an effect on the computation time. (Data not shown) In most cases the differences were too small to consider significant. Although some difference was expected between methods using backtrack to produce an alignment since the final alignments would be longer when there is low similarity due to increased amounts of gap.

“Score only” versus “backtracking” modes. When a comparison between “score only” and “backtracking” modes, there has been small variations in the computation time in spite of the choice of the compiler/interpreter since backtracking requires three additional matrices to be constructed. Between “score_only” modes and “backtracking” modes of Biopython there is a huge impact on the performance. This impact drops down for larger sequences for all methods with Biopython. On the other hand, with Numba and PyPy the impact is significantly less and ranging only between 1.5 and 2. The difference is mostly related to the way the Biopython’s pairwise2 module is executed rather than the choice of the compiler/interpreter.Local vs global. No significant difference was observed between local and global algorithms. The only exception was Biopython methods where the difference varies between 1.3 to 10 times. In case of Numpy or PyPy no significant difference was apparent.

Substitution matrix choice. Oddly, Numpy exploiting methods when compiled with Cython proved to be even slower when using methods that include a substitution matrix. Considering that these methods are generally less efficient than Cython compiled pure Python code, an overall Cython Numpy uncomplimentary was apparent.

Between the methods a dramatic difference was observed when Biopython module was used. Only for Biopython, while globalms method was fastest, localds and localds with backtracking were significantly slower than their counterparts.

Using standard Python interpreter, Biopython showed greater overall performance in comparison to Numpy module or pure Python. However, with the introduction of an alternative compiler performance dramatically improves. When overall performance is considered, Numba assisted just-in-time compilation proves to be optimal in almost all cases. On average, Numba provided 15 times faster computation time in “score only” mode and 18 times faster computation time in “backtracking” mode than Biopython. Biopython showed better performance only for globalms method when aligning sequences shorter than ~200 base pairs. At the same time, PyPy achieved greater performance only for localms method and for sequences of a length of ~50 bp. Comparison of PyPy and BioPython showed that PyPy was around 3.5 times faster on average in “score only” mode while comparable in “backtracking” mode.

Table 2. Computation times in milliseconds of various sequence alignment methods, globalms, localms, globalds and localds, with and without backtracking, using different compilation strategies. All methods were tested with varying sequence lengths and 20% sequence similarity. Best performances of each series of sequence lengths are reported in bold.

	method:			**globalms without backtracking**				
	sequence length:	**50**	**100**	**200**	**400**	**600**	**800**	**1000**
module	**interpreter/compiler**							
Biopython	standard Python 3.6+	0.009	0.026	0.087	0.470	1.212	2.048	3.225
numpy	Numba	0.022	0.046	0.096	0.433	0.795	1.363	2.159
-	PyPy	0.029	0.108	0.449	1.433	2.918	4.887	7.194
numpy	Cython	0.139	0.553	2.236	10.120			
-	Cython	0.631	2.475	9.878	40.824			
numpy	standard Python 3.6+	1.001	3.916	15.898				
-	standard Python 3.6+	0.475	1.980	7.548				

	method:			**localms without backtracking**				
module	**interpreter/compiler**	**50**	**100**	**200**	**400**	**600**	**800**	**1000**
Biopython	standard Python 3.6+	0.065	0.248	1.019	4.583	11.708	22.448	36.291
numpy	Numba	0.022	0.047	0.098	0.465	0.793	1.383	2.125
-	PyPy	0.024	0.090	0.367	1.396	3.102	5.203	7.766
numpy	Cython	0.145	0.569	2.258	10.025			
-	Cython	0.748	2.883	11.559	47.241			
numpy	standard Python 3.6+	1.149	4.614	17.805				
-	standard Python 3.6+	0.494	1.958	7.694				

	method:			**globalds without backtracking**				
module	**interpreter/compiler**	**50**	**100**	**200**	**400**	**600**	**800**	**1000**
Biopython	standard Python 3.6+	0.100	0.381	1.501	6.082	13.618	24.038	37.705
numpy	Numba	0.021	0.047	0.100	0.445	0.827	1.318	1.949
-	PyPy	0.025	0.114	0.466	1.715	3.757	6.488	9.768
numpy	Cython	0.459	1.825	7.354	30.445			
-	Cython	0.602	2.362	9.399	38.945			
numpy	standard Python 3.6+	0.998	3.967	15.507				
-	standard Python 3.6+	0.879	3.314	13.300				

	method:			**localds without backtracking**				
module	**interpreter/compiler**	**50**	**100**	**200**	**400**	**600**	**800**	**1000**
Biopython	standard Python 3.6+	0.157	0.588	2.481	10.386	24.193	44.655	70.487
numpy	Numba	0.023	0.048	0.107	0.466	0.865	1.475	2.205
-	PyPy	0.029	0.124	0.511	1.928	4.086	7.518	11.400
numpy	Cython	0.520	2.089	8.506	35.492			
-	Cython	0.735	2.853	11.417	46.367			
numpy	standard Python 3.6+	1.172	4.542	18.131				
-	standard Python 3.6+	0.962	3.784	14.622				

	method:			**globalms with backtracking**				
module	**interpreter/compiler**	**50**	**100**	**200**	**400**	**600**	**800**	**1000**
Biopython	standard Python 3.6+	1.761	2.190	4.022	7.949	9.164	18.490	20.475
numpy	Numba	0.023	0.050	0.177	0.681	1.573	2.667	4.226
-	PyPy	0.058	0.214	0.840	3.495	7.218	12.612	18.929
numpy	Cython	0.317	1.253	4.951	22.217			
-	Cython	1.622	6.361	25.222	103.363			
numpy	standard Python 3.6+	2.308	9.126	36.224				
-	standard Python 3.6+	0.969	3.581	14.521				

	method:			**localms with backtracking**				
module	**interpreter/compiler**	**50**	**100**	**200**	**400**	**600**	**800**	**1000**
Biopython	standard Python 3.6+	0.169	0.616	2.496	10.177	24.115	43.812	69.278
numpy	Numba	0.023	0.049	0.126	0.684	1.612	2.799	4.327
-	PyPy	0.050	0.190	0.728	3.042	6.380	11.616	17.473
numpy	Cython	0.293	1.148	4.554	20.728			
-	Cython	1.653	6.310	25.439	102.680			
numpy	standard Python 3.6+	2.366	9.056	36.411				
-	standard Python 3.6+	0.903	3.558	13.804				

	method:			**globalds with backtracking**				
module	**interpreter/compiler**	**50**	**100**	**200**	**400**	**600**	**800**	**1000**
Biopython	standard Python 3.6+	1.399	2.377	3.429	8.613	16.305	27.459	41.009
numpy	Numba	0.026	0.058	0.158	0.867	1.850	3.252	4.909
-	PyPy							
numpy	Cython	0.962	3.837	15.267	63.191			
-	Cython	1.522	5.965	23.866	96.943			
numpy	standard Python 3.6+	2.329	8.859	35.338				
-	standard Python 3.6+	1.640	6.794	26.157				

	method:			**localds with backtracking**				
module	**interpreter/compiler**	**50**	**100**	**200**	**400**	**600**	**800**	**1000**
Biopython	standard Python 3.6+	1.444	3.545	6.627	18.518	38.599	68.369	105.336
numpy	Numba	0.027	0.064	0.180	0.945	2.028	3.541	5.458
-	PyPy							
numpy	Cython	1.018	3.999	16.154	68.273			
-	Cython	1.679	6.457	25.711				
numpy	standard Python 3.6+	2.392	9.485	37.360				
-	standard Python 3.6+	1.670	6.669	27.091				

Conclusion

While Python provide a versatile development environment for prototyping new ideas for data scientists, it remains to be one of slower executed programming languages. With the development of next generation sequencing technology and the increase in the amount data to be processed, massive alignment problems such as de novo genome construction or multiple sequence alignments (MSA) arise and bioinformaticians find themselves needing faster solutions for testing their ideas that require sequence alignments. Here, in order to boost the performance of these alignment algorithms, the two most commonly used alignment algorithms were written using Python language and tested for various compilation strategies. Our findings show that the choice of compiler may have significant impact on the speed of execution. The overall optimal approach was found to be using just-in-time Numba compiler while followed by PyPy just-in-time compiler. Biopython also proved to be a decent option, considering certain methods at certain sequence lengths it may give equivalent performance. In conclusion, it is important to consider compilers, no matter the choice of the compiler is, as they would give up to 200 times higher performance than standard Python interpreter.

On the other hand, further improvements may still be possible if compilation strategy is combined with parallelism. However, the overhead cost of setting up parallelism needs to be considered depending on the number and length of the query sequences and the available hardware, such as GPUs. [16-18] For that reason, parallelism might not be a feasible solution to improve the performance in all situations. None the less, the compilation strategy ensures better performance in any Python environment given that necessary modules are installed and independent of the hardware setting.

Conflicts of interest

There are no conflicts of interest in this work.

Acknowledgment

This work was funded by TÜBİTAK project no 118Z226.

References

[1] Zhang J., Chiodini R., Badr A., Zhang G., The impact of next-generation sequencing on genomics, *Journal of Genetics and Genomics*, 38 (2011) 95-109.

[2] McPherson J.D., Next-generation gap, *Nature Methods*, 6 (2019) S2-S5.

[3] Branton D., Deamer D. The Development of Nanopore Sequencing, Nanopore Sequencing, (2019) 1-16

[4] Theis T.N., Wong P.H.S., The End of Moore's Law: A New Beginning for Information Technology, *Computing in Science & Engineering*, 19 (2017) 41-50.

[5] Needleman S.B., Wunsch C.D., A general method applicable to the search for similarities in the amino acid sequence of two proteins, *Journal of Molecular Biology*, 48 (1970) 443-453.

[6] Smith T.F., Waterman M.S., Identification of common molecular subsequences, *Journal of Molecular Biology*, 147 (1981) 195-197.

[7] Gotoh O., An improved algorithm for matching biological sequences, *Journal of Molecular Biology*, 162 (1982) 705-708.

[8] Marco-Sola, S., Moure, J. C., Moreto, M., Espinosa, A., Fast Gap-Affine Pairwise Alignment Using the Wavefront Algorithm, Bioinformatics, 37 (2020) 456–463.

[9] Song Y.-J., Ji D. J., Seo H., Han G.-B., Cho D.-H., Pairwise Heuristic Sequence Alignment Algorithm Based on Deep Reinforcement Learning, *IEEE Open Journal of Engineering in Medicine and Biology*, 2 (2021) 36–43.

[10] Rashed A. E. E.-D., Amer H. M., El-Seddek M., Moustafa H. E.-D., Sequence Alignment Using Machine Learning-Based Needleman–Wunsch Algorithm, *IEEE Access*, 9 (2021) 109522–109535.

[11] Nagpal A., Gabrani G., Python for Data Analytics, Scientific and Technical Applications, 2019 Amity International Conference on Artificial Intelligence (AICAI), Dubai, (2019).

[12] Mondal S., Khatua S., Accelerating Pairwise Sequence Alignment Algorithm by MapReduce Technique for Next-Generation Sequencing (NGS) Data Analysis, Advances in Intelligent Systems and Computing, (2019) 213-220.

[13] Marçais G., Delcher A.L., Phillippy A.M., Coston R., Salzberg S.L., Zimin A., MUMmer4: A fast and versatile genome alignment system, *PLoS Computational Biology*, 14 (2018) e1005944.

[14] Tarasov A., Vilella A.J., Cuppen E., Nijman I.J., Prins P., Sambamba: fast processing of NGS alignment formats, Bioinformatics, 31 (2015) 2032-2034.

[15] Marowka A., Python accelerators for high-performance computing, *The Journal of Supercomputing*, 74 (2018) 1449-1460.

[16] Haghi A., Marco-Sola S., Alvarez L., Diamantopoulos D., Hagleitner C., Moreto M., An FPGA Accelerator of the Wavefront Algorithm for Genomics Pairwise Alignment, 31st International Conference on Field-Programmable Logic and Applications (FPL), Dresden, (2021).

[17] Rognes, T., Faster Smith-Waterman database searches with inter-sequence SIMD parallelization, *BMC Bioinformatics*, 12 (2011) 1.

[18] Liu Y., Maskell D. L., Schmidt, B., CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units. *BMC Research Notes*, 2, 1 (2009) 73.