SequenceAligner: A High-Performance Tool for Large-Scale All-Versus-All Pairwise Sequence Alignment

Jakov Dragičević[1], Erik Otović[1,2,*] and Goran Mauša[1,2]

[1]University of Rijeka, Faculty of Engineering, Rijeka, Croatia
[2]University of Rijeka, Center for Artificial Intelligence and Cybersecurity, Rijeka, Croatia

Abstract

Sequence alignment is an indispensable technique in bioinformatics. It facilitates the comparative analysis of biological sequences for evolutionary studies, drug discovery, protein function prediction, as well as data filtering based on similarity and results analysis in machine learning-based methodologies. Despite the range of available alignment libraries and tools, researchers still face significant challenges when implementing large-scale all-versus-all pairwise sequence alignment workflows, particularly when a complete solution is required to process datasets that exceed available memory. The SequenceAligner software offers an open-source solution tailored for high-performance all-versus-all sequence alignment, emphasizing convenience, efficiency, scalability, and accessibility. It provides exact dynamic programming solutions for the Needleman-Wunsch, Smith-Waterman, and Gotoh algorithms with configurable substitution matrices and gap penalty models delivered as a complete workflow rather than merely a library. Moreover, the algorithms are implemented for both CPUs and CUDA-compatible GPUs, enabling efficient parallel sequence alignment. Performance benchmarks on three peptide datasets demonstrate that SequenceAligner achieves alignment rates ranging from 3.2 to 22.9 million sequence pairs per second on a consumer-grade AMD Ryzen 7 5700X3D CPU, and from 27.4 to 80.8 million pairs per second with CUDA acceleration on a consumer-grade NVIDIA GeForce RTX 4060 GPU. Notably, the software maintains memory efficiency even when the similarity matrix exceeds the available RAM or VRAM. Finally, the software underscores modularity and extensibility through its simple C99 codebase, rendering it accessible for both research applications and educational purposes. SequenceAligner is publicly available in a GitHub repository at https://github.com/jakovdev/SequenceAligner.

Keywords

Biological sequences, Sequence alignment, Sequence similarity, High-performance computing

1. Introduction

Sequence alignment forms the cornerstone of computational biology, enabling researchers to identify evolutionary relationships, predict protein structure and function, and discover conserved motifs across biological sequences [1, 2]. As biological databases continue to expand exponentially, with repositories like UniProt [3] containing over 200 million protein sequences and GenBank [4] housing billions of nucleotide sequences, the computational demands for comprehensive sequence analysis have grown proportionally. This growth necessitates efficient computational workflows capable of processing millions of sequence comparisons while maintaining accuracy and algorithmic flexibility.

Contemporary bioinformatics research increasingly requires a comprehensive analysis of sequence similarity through pairwise sequence alignment, which involves aligning and comparing two biological sequences to identify regions that may indicate functional, structural, or evolutionary relationships. Proteomics research requires all-versus-all comparisons for protein family classification and evolutionary relationship identification [5]. Drug discovery applications utilize all-versus-all similarity matrices for target identification and side effect prediction [6]. Comparative genomics studies depend on precise alignment scores for ortholog detection across species [7].

SQAMIA 2025: Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications, September 10-12, 2025, Maribor, Slovenia
*Corresponding author.
jdragicevic1@uniri.hr (J. Dragičević); erik.otovic@uniri.hr (E. Otović); goran.mausa@uniri.hr (G. Mauša)
0009-0009-8936-7573 (J. Dragičević); 0000-0001-5713-5879 (E. Otović); 0000-0002-0643-4577 (G. Mauša)

Sequence alignment also plays a crucial role in data preparation for machine learning-based peptide activity prediction. To prevent biased predictions caused by large clusters of highly similar sequences, the dataset is filtered prior to model training by removing redundant peptides, ensuring that no two peptides exceed a customary similarity threshold [8, 9, 10, 11, 12]. Percentage sequence identity (*PID*), shown in Equation 1, is often used to quantify the similarity between two sequences by aligning them and computing the percentage of matching residues ($N_{matching}$) with respect to the length of the shorter sequence, where sequence lengths are indicated by len_1 and len_2. Furthermore, the application of pairwise sequence alignment is not limited only to the data preparation phase, as it can also support the analysis of *in silico* generated sequences. For example, recently it has been used in a pipeline combining machine learning and a genetic algorithm to analyze generated peptide sequences and identify motifs within them [13].

$$PID = \frac{N_{matching}}{min(len_1, len_2)} \tag{1}$$

Although some applications, such as dataset filtration for machine learning, can tolerate heuristic approximations to avoid exhaustive all-versus-all alignments and improve execution time, exact alignments remain essential in other areas of bioinformatics [2]. The fundamental computational challenge in all-versus-all pairwise sequence alignment lies in its inherent complexity. To align sequences of lengths m and n, commonly used dynamic programming algorithms require $\mathcal{O}(mn)$ time and space complexity. When extended to all-versus-all comparisons of N sequences, this complexity scales to $\mathcal{O}(N^2 \cdot \bar{L}^2)$ where $\bar{L}$ represents the average sequence length. For large datasets, such implementations quickly becomes computationally prohibitive when no optimization strategies are implemented.

While modern CPU and GPU architectures offer substantial performance potential through vectorization, parallelization, and advanced memory hierarchies, taking advantage of these features requires low-level optimization and detailed knowledge of both hardware and algorithm design [14, 15]. Consequently, effectively utilizing these hardware features requires specialized knowledge and careful implementation that many researchers lack the time or expertise to develop independently. Therefore, the barrier to entry for implementing high-performance all-versus-all alignment workflows remains substantial. Researchers must navigate complex decisions regarding algorithm selection, optimization strategies, memory management, and result storage while ensuring correctness and reproducibility. This complexity often forces compromises between computational feasibility and analysis completeness, limiting the scope of possible research investigations.

2. Related Works

The landscape of sequence alignment tools presents researchers with a fragmented ecosystem in which no single solution addresses the complete workflow requirements for large-scale all-versus-all analyses. This fragmentation highlights the critical importance of open-source software development in computational biology, where transparency, reproducibility, and community-driven improvements are essential for scientific progress.

Available alignment libraries offer robust foundational components, but integrating them into complete and efficient workflows often requires substantial additional programming effort from users. Parasail [16], for example, is a high-quality implementation of vectorized pairwise alignment algorithms, featuring numerous substitution matrices and multi-language APIs. While Parasail delivers high performance for individual alignments, it focuses solely on core alignment routines.

BioPython [17] and Rust-Bio [18] provide alignment functionality as part of broader bioinformatics frameworks, offering seamless integration with existing computational pipelines. The Python library scikit-bio [19] follows a similar approach, providing data structures, algorithms, and educational resources as part of a general-purpose bioinformatics toolkit. While these libraries offer convenient interfaces and broad functionality, they prioritize versatility over the specialized performance required for large-scale, all-versus-all analyses.

The challenge extends beyond algorithmic implementation to encompass the complete computational workflow. Researchers implementing all-versus-all alignment workflows must address sequence file parsing and validation, memory management for sequences, multithreading the all-versus-all workflow, progress monitoring for long-running computations and similarity matrix storage that may exceed available RAM. Building these components from *scratch* using existing libraries requires substantial software development expertise and time that could be better spent on biological research. Existing libraries also lack comprehensive support for heterogeneous computing environments. While some libraries provide CPU optimizations, GPU acceleration is limited only to pairwise alignments and as such it does not scale well with large all-versus-all operations. This limitation prevents researchers from fully utilizing available computational resources without implementing complex device management logic.

The algorithmic flexibility requirements in research often necessitate modifications to scoring schemes, gap penalty models, or alignment strategies. While high-performance libraries may provide extensive configuration options, they typically cannot accommodate novel algorithmic variants without significant modification to library internals. This limitation creates a trade-off between performance and flexibility that constrains research possibilities.

Memory management represents another critical challenge for large-scale analyses. Libraries optimized for individual alignments may not provide strategies for handling similarity matrices that exceed available system memory. Researchers must implement custom solutions for memory-mapped storage, result streaming, or distributed computation approaches to handle datasets that produce gigabytes or terabytes of results.

The fragmented nature of current tools creates additional barriers through inconsistent interfaces, conflicting dependencies, and complex build processes. Researchers must navigate multiple APIs, manage dependency compatibility issues, and often require specialized build environments to compile and deploy their workflows. These technical barriers particularly affect researchers in resource-constrained environments or those without dedicated computational support.

3. SequenceAligner Tool

SequenceAligner addresses the limitations of the aforementioned tools through a comprehensive approach that prioritizes four core design principles: convenience through complete workflow integration, efficiency through specialized all-versus-all optimizations, scalability through adaptive memory management, and adaptability through simple codebase design. The implementation provides a ready-to-use solution capable of utilizing modern CPU and CUDA-compatible GPU architectures that eliminates the need for researchers to assemble complex toolchains while maintaining the flexibility necessary for scientific research. Even though the developed tool is primarily intended for sequence similarity–based dataset filtration in machine learning pipelines for peptide activity prediction, where peptide sequences are typically short and consist of up to 50 amino acid residues, it imposes no limitation on sequence length and can also be employed in other applications. SequenceAligner and scripts necessary to replicate the results from this study are publicly available in a GitHub repository at https://github.com/jakovdev/SequenceAligner.

3.1. Convenience: Complete Workflow Integration

Unlike software libraries that provide only algorithmic components, SequenceAligner delivers a complete end-to-end workflow optimized for all-versus-all sequence alignment. The software handles sequence file parsing with automatic validation, memory-efficient sequence storage using custom pool allocators, comprehensive result management with HDF5 [20] output format, progress monitoring with detailed performance metrics, and cross-platform deployment with automated build scripts.

The software provides comprehensive substitution matrix support with 65 amino acid matrices and 2 nucleotide matrices sourced from Parasail [16]. A Python utility script automates the extraction and conversion of these matrices into C-compatible format, enabling easy integration and potential reuse in

other implementations. The supported matrices include the complete BLOSUM series [21] (BLOSUM30 through BLOSUM100), the PAM series [22] (PAM10 through PAM500), as well as nucleotide matrices DNAFULL and NUC44 from BLAST [23]. The tool also offers full support for configurable gap penalty models. Its flexible implementation can serve as a foundation for researchers wishing to extend, modify, or implement custom scoring schemes or alignment variants.

In addition to alignment, the software includes built-in functionality for similarity-based dataset filtering, enabling the removal of redundant sequences to reduce bias in machine learning models for peptide activity prediction.

The filtering algorithm operates by iteratively processing sequences and comparing each candidate sequence against all previously accepted sequences in the filtered set. For each comparison, *PID* is computed on-demand by counting exact character matches between the sequences and dividing by the length of the shorter sequence. A candidate sequence is added to the filtered set only if its similarity to all existing sequences in the set remains below a user-defined threshold. This guarantees that the resulting dataset contains no pair of sequences with similarity above the threshold. The algorithm has a time complexity of $\mathcal{O}(N^2)$ in the worst case, where N is the number of input sequences. The procedure is inherently sequential, as the decision for each sequence depends on the previously selected sequences. The simple character-by-character comparison is efficiently vectorized using SIMD instructions when available, maintaining reasonable performance for practical dataset sizes.

The command-line interface enables flexible parameter configuration, with sensible default settings for typical use cases and well-documented options for advanced customization. Built-in parameter validation helps ensure correct configuration, and detailed runtime reporting provides insight into computational performance and resource usage.

3.2. Efficiency: Specialized All-Versus-All Optimizations

SequenceAligner implements optimizations tailored for all-versus-all comparison scenarios, rather than generic pairwise alignment tasks. The CPU implementation features targeted SIMD enhancements with support for AVX512, AVX2, and SSE2, primarily optimizing vectorized conversion of character sequences into pre-mapped integer representations and efficient matrix initialization. This preprocessing step transforms raw nucleotide or amino acid sequences into index arrays using lookup tables, eliminating per-character translation during alignment and reducing runtime overhead. Although the core dynamic programming loops rely on compiler auto-vectorization rather than explicit SIMD intrinsics, the implementation incorporates memory prefetching and stack-based allocation for small matrices to improve cache efficiency. This preprocessing is particularly advantageous when the same sequences are involved in numerous alignment comparisons, as the time required for retrieval from cache is negligible in comparison to the alignment process.

A key efficiency enhancement is the implementation of a specialized memory pool allocator designed specifically for sequence storage. The allocator uses a linked-list structure of memory blocks, each sized at 4 MiB, to provide cache-friendly allocation patterns while minimizing memory fragmentation. Each 4 MiB block is allocated using huge page allocation when available on Linux systems, leveraging the `madvise(MADV_HUGEPAGE)` system call to request transparent huge pages that reduce translation lookaside buffer misses and improve memory access latency [24].

The pool allocator operates through a bump-pointer strategy within each block, where allocations advance a pointer through the available space without complex bookkeeping overhead. When a block becomes full, a new 4 MiB block is allocated and linked to the chain. All sequence string data is allocated contiguously within these blocks with 8-byte alignment, ensuring that sequences accessed sequentially during alignment operations exhibit improved spatial locality and cache utilization across different processor architectures.

This design benefits all-versus-all alignment workloads where the same sequences are repeatedly accessed throughout the computation. Since sequence data remains in the pool throughout the entire alignment process, subsequent accesses benefit from cache residency, reducing memory bandwidth requirements and improving computational throughput. The pool allocator eliminates the overhead

of frequent `malloc`/`free` operations while providing deterministic memory access patterns suited for high-performance computing applications.

The multithreaded implementation employs an adaptive batch-based work distribution strategy that balances load distribution with cache efficiency considerations. Rather than assigning individual sequence pairs to threads, which would incur significant synchronization overhead, the system distributes work in batches of consecutive rows from the upper triangular alignment matrix. Each thread acquires a mutex-protected batch of rows to process, with batch sizes dynamically adjusted based on the remaining work and number of active threads.

The work distribution algorithm implements adaptive batch sizing with two distinct phases. In the initial phase, covering the first 90% of sequence pairs, batch sizes equal the number of threads to ensure balanced workload distribution across all available cores. As the computation approaches completion and fewer rows remain, batch sizes are halved to prevent thread starvation and ensure that all threads remain productive until completion. This adaptive approach addresses the inherent load imbalance in triangular matrix computations, where later rows contain progressively fewer elements to process.

Thread-local progress tracking minimizes synchronization overhead by accumulating alignment counts locally within each thread and periodically updating shared progress counters through atomic operations. This approach reduces contention on shared variables while maintaining accurate progress reporting for user feedback. The implementation balances update frequency to provide responsive progress indicators without excessive synchronization overhead.

The sequence access pattern is optimized for both spatial and temporal locality through strategic prefetching of sequence data into processor caches. When retrieving sequence pointers, the implementation explicitly prefetches the corresponding character data using processor-specific prefetch instructions. This anticipates imminent access during alignment computation, reducing cache miss penalties.

The sequence storage layout maintains arrays of sequence lengths and pointers separate from the actual sequence character data stored in the memory pools. This separation enables efficient metadata access during batch processing while ensuring that sequence character data benefits from the cache-friendly allocation patterns of the pool allocator. The memory pool's contiguous allocation strategy ensures that sequences loaded during the initial phases of computation remain cache-resident for subsequent accesses, which is particularly beneficial given the all-versus-all comparison pattern where each sequence participates in multiple alignment operations.

Memory prefetching directives are strategically placed to load sequence data ahead of the alignment computation, taking advantage of the predictable access patterns inherent in all-versus-all alignment scenarios. Additional prefetching occurs within the similarity computation functions, where vectorized comparison operations prefetch upcoming sequence segments to maintain memory bandwidth utilization during SIMD-accelerated character matching operations.

The optional GPU acceleration through CUDA provides substantial performance improvements across all implemented algorithms. Modern GPU architectures demonstrate consistent acceleration capabilities for sequence alignment tasks, with particularly pronounced benefits for computationally intensive algorithms such as Smith-Waterman [15].

The software includes an optimization that automatically detects when the parameters of the affine gap Gotoh algorithm reduce to a linear gap case (i.e., when the gap open and extension penalties are equal, $\alpha = \beta$) and transparently switches to a more efficient Needleman–Wunsch implementation. This not only preserves correctness, as the recurrence relations become equivalent, but also improves performance by avoiding unnecessary affine penalty overhead. A user warning is also issued to warn the user that the affine model is configured redundantly, encouraging parameter clarity and optimal performance.

3.3. Scalability: Adaptive Memory Management

SequenceAligner addresses the critical challenge of processing datasets that produce results larger than available system memory through sophisticated memory management strategies. The implementation

automatically detects when the similarity matrix exceeds available RAM and seamlessly transitions to memory-mapped file storage with chunked organization optimized for both storage efficiency and subsequent analysis access patterns.

The triangular matrix storage option reduces memory requirements by 50% for symmetric results while maintaining computational efficiency through optimized index calculation. This optimization is crucial for large datasets where memory efficiency directly impacts computational feasibility. Result compression with configurable levels (0-9) provides additional space-time trade-offs, allowing users to optimize storage requirements based on available disk space and analysis timeline constraints. After analysis, the chunked storage organization enables efficient partial result access without loading entire matrices into memory.

For GPU computations, the software implements automatic batching when results exceed GPU memory capacity, maintaining high computational throughput while managing memory constraints transparently.

3.4. Accessibility: Simple and Extensible Codebase

SequenceAligner emphasizes code clarity and modularity to facilitate algorithmic extensions and educational use. The implementation adheres to standard C99 and employs a clean, modular design with minimal third-party dependencies, namely HDF5 and the optional CUDA toolkit. This design avoids complex dependency chains that can hinder deployment across diverse research environments.

Core alignment routines are implemented as self-contained modules with well-defined interfaces, allowing researchers to adapt or extend functionality with minimal code modification. For example, adding a custom substitution matrix requires defining the matrix as a 2D array and adding a single line to the substitution matrix collection. No additional changes to the code are necessary. This enables user-defined configurations without additional programming overhead. The substitution matrix abstraction and algorithm selection system support the integration of new alignment models without the need for major structural changes. This flexibility enables rapid prototyping of alternative scoring schemes or novel algorithmic variants.

The software's modular architecture separates algorithmic computation from input and output operations, memory management, and result handling. This separation supports independent optimization of each component and facilitates the incorporation of external libraries, such as employing Parasail's optimized alignment functions for specific use cases while maintaining the complete workflow infrastructure.

Platform abstraction layers unify system-specific operations such as thread management, file input and output, and hardware-specific optimizations. This approach ensures performance and portability across both Linux and Windows environments. The build system detects available CPU and GPU features and automatically applies the appropriate compilation optimizations.

The implementation is structured to preserve educational clarity while maintaining high performance. The codebase employs clear abstraction layers that encapsulate low-level optimizations such as SIMD vectorization, ensuring that performance enhancements do not compromise code readability. The linear and modular design with clear separation of concerns makes the codebase well-suited not only for research applications, but also as a resource for students and practitioners interested in understanding or modifying alignment algorithms.

3.5. Open Source Development Model

The open-source nature of SequenceAligner facilitates community-driven improvements and specialization for diverse research needs. Researchers can easily fork the project to implement domain-specific modifications while maintaining the robust workflow infrastructure. This approach leverages the collaborative development model that has proven successful in computational biology, where community contributions drive innovation and ensure broad applicability across research domains.

The simple codebase structure enables researchers to contribute improvements, optimizations, or specialized variants back to the community, fostering a collaborative development environment that benefits the entire bioinformatics research community.

Open-source development principles are fundamental to advancing computational biology research. They enable algorithmic verification, customization for specific research needs, and collaborative improvement of methods. The ability to examine, modify, and extend alignment algorithms is crucial for researchers who need to adapt methods to novel biological questions or incorporate domain-specific knowledge into their analyses. Furthermore, open-source tools facilitate reproducible research by providing transparent implementations that can be verified and replicated by the scientific community.

3.6. Alignment Algorithms

SequenceAligner implements three commonly used pairwise sequence alignment algorithms, all based on dynamic programming for computing optimal alignments between two biological sequences. In the following descriptions of these algorithms, the notation $S_1 = s_1^1 s_1^2 \dots s_1^m$ and $S_2 = s_2^1 s_2^2 \dots s_2^n$ is used to represent two sequences of lengths m and n, respectively.

3.6.1. Needleman-Wunsch Algorithm with Linear Gap Penalties

The Needleman-Wunsch algorithm [25] computes global alignment using the recurrence relation:

$$F(i,j) = \max \begin{cases} F(i-1,j-1) + s(s_1^i, s_2^j) & \text{(match/mismatch)} \\ F(i-1,j) - d & \text{(deletion)} \\ F(i,j-1) - d & \text{(insertion)} \end{cases} \tag{2}$$

where $s(a,b)$ is the scoring function from the substitution matrix and d is the linear gap penalty (specified as a positive value). The boundary conditions are:

$$F(i,0) = i \cdot (-d) \quad \text{for } i = 0, 1, \dots, m \tag{3}$$

$$F(0,j) = j \cdot (-d) \quad \text{for } j = 0, 1, \dots, n \tag{4}$$

The optimal global alignment score is $F(n,m)$.

3.6.2. Gotoh Algorithm with Affine Gap Penalties

The Gotoh algorithm [26] extends global alignment with affine gap penalties using three matrices: $M(i,j)$ for matches/mismatches, $I_x(i,j)$ for horizontal gaps, and $I_y(i,j)$ for vertical gaps:

$$M(i,j) = s(s_1^i, s_2^j) + \max\{M(i-1,j-1), I_x(i-1,j-1), I_y(i-1,j-1)\} \tag{5}$$

$$I_x(i,j) = \max\{M(i,j-1) - \alpha, I_x(i,j-1) - \beta\} \tag{6}$$

$$I_y(i,j) = \max\{M(i-1,j) - \alpha, I_y(i-1,j) - \beta\} \tag{7}$$

where α is the gap opening penalty and β is the gap extension penalty.
The boundary conditions are:

$$M(0,0) = 0, \quad I_x(0,0) = I_y(0,0) = -\infty/2 \tag{8}$$

$$\text{For } j > 0: \quad I_x(0,j) = \max\{M(0,j-1) - \alpha, I_x(0,j-1) - \beta\} \tag{9}$$

$$M(0,j) = I_x(0,j), \quad I_y(0,j) = -\infty/2 \tag{10}$$

$$\text{For } i > 0: \quad I_y(i,0) = \max\{M(i-1,0) - \alpha, I_y(i-1,0) - \beta\} \tag{11}$$

$$M(i,0) = I_y(i,0), \quad I_x(i,0) = -\infty/2 \tag{12}$$

The optimal global alignment score is $M(n,m)$.

3.6.3. Smith-Waterman Algorithm with Affine Gap Penalties

The Smith-Waterman algorithm [27] identifies optimal local alignments using the same three-matrix approach as Gotoh, but with a zero lower bound to allow alignment termination at any position:

$$M(i,j) = \max\{0, s(s_1^i, s_2^j) + \max\{M(i-1,j-1), I_x(i-1,j-1), I_y(i-1,j-1)\}\} \tag{13}$$

$$I_x(i,j) = \max\{0, M(i,j-1) - \alpha, I_x(i,j-1) - \beta\} \tag{14}$$

$$I_y(i,j) = \max\{0, M(i-1,j) - \alpha, I_y(i-1,j) - \beta\} \tag{15}$$

The boundary conditions are:

$$M(0,j) = 0 \quad \text{for all } j \geq 0 \tag{16}$$

$$M(i,0) = 0 \quad \text{for all } i \geq 0 \tag{17}$$

$$I_x(0,j) = I_y(0,j) = -\infty \quad \text{for } j > 0 \tag{18}$$

$$I_x(i,0) = I_y(i,0) = -\infty \quad \text{for } i > 0 \tag{19}$$

$$I_x(0,0) = I_y(0,0) = -\infty \tag{20}$$

In practice, $-\infty$ is implemented as `INT_MIN/2` to prevent integer overflow during arithmetic operations, where INT_MIN denotes minimum value that can be stored in a 32-bit integer variable.

The optimal local alignment score is the maximum value found in the M matrix during computation: $\max_{0 \leq i \leq m, 0 \leq j \leq n} M(i,j)$.

Although all three algorithms share the same space and time complexity of $\mathcal{O}(mn)$, Smith-Waterman and Gotoh require more computationally intensive operations per matrix cell, due to local alignment scoring and affine gap penalty calculations, respectively, resulting in higher practical runtime compared to Needleman-Wunsch with linear gap penalties.

4. Performance Evaluation and Benchmarking

4.1. Experimental Setup

A comprehensive performance evaluation was conducted to assess SequenceAligner's effectiveness across various computational scenarios and to compare it against Parasail, which has been shown to be one of the fastest libraries for pairwise sequence alignment[16]. The evaluation utilized three carefully selected datasets representing different scales of bioinformatics analysis: AVPPred [28] (small-scale), AMP [8] (medium-scale), and Drosophila (large-scale) taken from The PeptideAtlas Project [29]. These datasets were chosen in order to benchmark performance with respect to the dataset size and average sequence length, and to assess the computational requirements. The statistical characteristics of the used datasets are given in Table 1.

Table 1
Statistical characteristics of the datasets used for performance evaluation.

Dataset	Sequences	Avg. Length	Pairwise Alignments	Scale Category
AVPPred	1,042	21.6	542,361	Small
AMP	9,409	30.5	44,259,936	Medium
Drosophila	58,746	17.9	1,725,516,885	Large

Hardware configuration for CPU testing utilized an AMD Ryzen 7 5700X3D processor based on the Zen 3 architecture, and the software was built using the C compiler from GCC (GNU Compiler Collection) with the target microarchitecture set to x86-64-v3. This microarchitecture level encompasses key performance features such as AVX2, FMA, and other advanced SIMD extensions, with AVX2 serving as the primary SIMD backend. The system featured 32 GB of DDR4-3200 memory in a dual-channel

configuration. CUDA implementation was tested on an NVIDIA GeForce RTX 4060 with 1024 CUDA threads.

For comparative analysis, the Parasail implementation was developed using Python multiprocessing with minimal features to enable fair all-versus-all comparison. The Parasail implementation discarded alignment results to minimize I/O overhead and was written to simulate a researcher needing a quick all-versus-all implementation while still being aware of the fastest available tools. On the other hand, SequenceAligner does store results on-the-fly which provides a more real-world usage scenario, but replicating it in Python would incur a more significant overhead which could skew results in SequenceAligner's favor.

4.2. Dataset Scale and Sequence Length Impact

The performance evaluation demonstrates clear scaling patterns related to both dataset size and average sequence length. Table 2 shows the computational time and throughput for SequenceAligner across different dataset characteristics using 16 CPU threads.

Table 2
Impact of average sequence length on throughput of SequenceAligner. Values in parentheses indicate the percentage difference relative to the baseline (AVPPred). *16 CPU threads, Needleman-Wunsch algorithm,* ***APS*** *= Alignments per second.*

Dataset	Average Sequence Length	Throughput (APS)
AVPPred	21.6 (baseline)	22,948,508 (baseline)
AMP	30.5 (+41.2%)	12,616,052 (-45.0%)
Drosophila	17.9 (-17.1%)	33,609,530 (+46.5%)

The results reveal that average sequence length significantly impacts per-alignment computational cost. Despite AMP requiring 81 times more alignments to be performed in comparison to AVPPred, it achieves only 55% of the throughput due to its 41% longer average sequence length. Conversely, the Drosophila dataset, with sequences 17% shorter than AVPPred, achieves 46% higher throughput despite its substantialy larger scale of 3,182 times more alignments.

4.3. CPU Threading Performance Analysis

Table 3 presents a threading efficiency analysis for different algorithms on the AMP dataset. Threading efficiency varies significantly by algorithm complexity. The Needleman-Wunsch algorithm, optimized for linear gap penalties in SequenceAligner, achieves speedup of 7.91 times with 16 threads. The more computationally intensive Gotoh and Smith-Waterman algorithms achieve superior scaling with a factors of 12.49 and 12.67, respectively. This indicates that complex algorithms benefit more from parallelization due to reduced thread management overhead relative to computation.

Table 3
Impact of CPU thread count on execution time and scalability of algorithms on AMP dataset (44.26M alignments). Values in parentheses indicate the speedup relative to the single-thread case.

Algorithm	Execution Time (seconds)			
	1 thread	4 threads	8 threads	16 threads
Needleman-Wunsch	27.748	7.081 (3.92×)	3.578 (7.76×)	3.508 (7.91×)
Gotoh Affine	166.204	42.118 (3.95×)	21.244 (7.82×)	13.305 (12.45×)
Smith-Waterman	174.372	44.692 (3.90×)	22.253 (7.84×)	13.757 (12.68×)

Table 4
Performance comparison of SequenceAligner and Parasail on all three datasets in the terms of total execution time and alignments per second. Speedup shows how many times the SequenceAligner outperforms Parasail in terms of alignments per second. *16 CPU threads, **APS** = Alignments per second.*

Dataset	Algorithm	SequenceAligner		Parasail		APS
		Time (s)	APS (in millions)	Time (s)	APS (in millions)	Speedup
AVPPred	NW	0.024	22.949	0.296	1.831	12.53×
	GA	0.084	6.440	0.290	1.871	3.44×
	SW	0.088	6.130	0.320	1.693	3.62×
AMP	NW	3.508	12.616	24.738	1.789	7.05×
	GA	13.305	3.327	23.704	1.867	1.78×
	SW	13.757	3.217	27.125	1.632	1.97×
Drosophila	NW	51.340	33.610	854.508	2.019	16.65×
	GA	180.672	9.551	910.110	1.896	5.04×
	SW	185.188	9.318	982.731	1.756	5.31×

4.4. Comparison of SequenceAligner and Parasail for All-Versus-All Sequence Alignment

A comprehensive comparison between SequenceAligner and Parasail demonstrates significant performance advantages for SequenceAligner across all tested configurations. Table 4 presents the performance comparison for 16-thread CPU implementations.

SequenceAligner consistently outperforms Parasail across all datasets and algorithms. The most dramatic improvements occur with the Needleman-Wunsch algorithm, where SequenceAligner achieves 7.05 to 16.65 times speedup due to the separation of linear and affine gap penalty models. For affine gap penalty algorithms (Gotoh and Smith-Waterman), SequenceAligner maintains substantial advantages of 1.78 to 5.31 times speedup.

The performance gap increases with dataset scale, suggesting SequenceAligner's optimizations become more effective with larger computational workloads. This is particularly evident in the Drosophila dataset where SequenceAligner achieves its highest relative performance gains. The results indicate that Parasail performs consistently across different sequence lengths, maintaining relatively stable throughput across datasets, while SequenceAligner's performance varies more significantly with sequence characteristics, achieving optimal performance on shorter sequences.

It is important to note that the purpose of these comparisons is not to suggest that SequenceAligner should replace Parasail entirely. Rather, these results demonstrate that SequenceAligner can compete with and even surpass established implementations for short to medium-length sequences in applications when all-versus-all sequence alignment are needed. Both Parasail's and SequenceAligner's architecture (being C99 codebases) allows Parasail to be integrated as an alternative alignment backend within SequenceAligner, providing users with the best of both worlds: optimized performance for short sequences through SequenceAligner's native implementation, and Parasail's proven effectiveness for very long sequences where its optimizations become more pronounced.

4.5. CUDA Acceleration Analysis

CUDA implementation demonstrates exceptional performance improvements across all datasets and algorithms. Table 5 compares CUDA against the best CPU performance (16 threads) for SequenceAligner.

CUDA acceleration provides substantial performance improvements across all scenarios, with speedups ranging from 2.40 to 8.59 times. The Gotoh and Smith-Waterman algorithms consistently achieve strong GPU acceleration (5.12 to 8.59 times speedups), while Needleman-Wunsch shows varied results with speedup ranging from 2.40 to 5.24 depending on dataset characteristics.

CUDA performance demonstrates optimal throughput characteristics influenced by both sequence length and dataset scale. The Drosophila dataset achieves the highest absolute throughput of 80.82M APS for Needleman-Wunsch, benefiting from both its shorter average sequences (17.9 amino acids) and

Table 5
Comparison of CPU and CUDA implementations of alignment algorithms available in SequenceAligner. Speedup shows how many times the CUDA implementation outperforms the CPU implementation in terms of alignments per second. ***APS*** *= Alignments per second.*

Dataset	Algorithm	CPU (16 threads)		CUDA		APS Speedup
		Time (s)	APS (in millions)	Time (s)	APS (in millions)	
AVPPred	NW	0.024	22.949	0.009	58.223	2.54×
	GA	0.084	6.440	0.011	49.418	7.67×
	SW	0.088	6.130	0.011	50.316	8.21×
AMP	NW	3.508	12.616	0.670	66.049	5.23×
	GA	13.305	3.327	1.610	27.486	8.26×
	SW	13.757	3.217	1.602	27.623	8.59×
Drosophila	NW	51.340	33.610	21.350	80.820	2.40×
	GA	180.672	9.551	34.201	50.452	5.28×
	SW	185.188	9.318	36.197	47.670	5.12×

massive scale (1.73 billion pairwise comparisons). This combination allows GPU parallelization to excel through two complementary mechanisms: shorter sequences reduce per-alignment computational overhead while the enormous number of alignments enables effective utilization of GPU's massive parallelization capabilities. The AMP dataset, with moderate sequence lengths (30.5 amino acids) and fewer total alignments (44.26M), still demonstrates strong GPU acceleration with excellent speedups maintained across all algorithms, particularly for the more computationally intensive affine gap penalty methods where the increased computational complexity better offsets GPU memory management overhead.

4.6. Computational Efficiency and Resource Utilization

The evaluation reveals distinct performance characteristics for different computational approaches. CPU implementations excel at sustained throughput with predictable scaling, while CUDA provides superior absolute performance with algorithm-dependent acceleration patterns.

SequenceAligner's CPU implementation achieves high efficiency through several optimizations: SIMD vectorization using AVX2 instructions, cache-aware memory access patterns, and optimized algorithmic implementations for specific gap penalty models. The threading model employs dynamic work distribution to maintain load balancing across cores.

CUDA implementation benefits from massive parallelization capabilities, processing thousands of alignment pairs simultaneously. Despite minimal optimization effort (1-2 days compared to weeks for CPU), CUDA consistently outperforms highly optimized CPU code, indicating significant untapped potential for GPU-accelerated sequence alignment.

4.7. Performance Summary and Future Directions

The comprehensive benchmarking demonstrates SequenceAligner's great performance across all evaluated metrics. CPU implementations achieve 1.78 to 16.65 times speedup over the all-versus-all Parasail Python script, while CUDA acceleration provides speedup ranging from 2.40 to 8.59 over CPU implementations.

These results strongly suggest that future development should prioritize CUDA implementations for all-versus-all sequence alignment, with CPU implementations serving as essential fallbacks for specific use cases. While GPU acceleration provides superior throughput for the majority of bioinformatics applications, CPU implementations remain valuable for very long sequences that may not fit efficiently in GPU memory, or for users without access to high-performance GPU hardware. Current CPU implementations, while highly optimized and effective, cannot match the raw computational throughput achievable with GPU acceleration for typical sequence analysis workflows. This performance advantage

is particularly relevant for machine learning applications where all-versus-all comparisons are commonly required.

The absence of publicly available CUDA kernels for all-versus-all sequence alignment represents a significant gap in the bioinformatics software ecosystem. The demonstrated performance advantages, combined with the increasing prevalence of GPU hardware in computational biology, indicate that GPU-accelerated implementations should become the standard for large-scale sequence analysis workflows.

SequenceAligner establishes new performance benchmarks for sequence alignment software while maintaining algorithmic accuracy and providing practical storage solutions through integrated HDF5 output. The substantial performance improvements, particularly for large-scale datasets, enable previously computationally prohibitive analyses to be performed on standard hardware configurations.

5. Conclusion

SequenceAligner provides an integrated solution for all-versus-all pairwise sequence alignment, addressing key workflow and scalability challenges commonly encountered in large-scale bioinformatics analyses. By providing a complete, ready-to-use solution rather than requiring assembly from multiple libraries, SequenceAligner eliminates substantial barriers to entry for researchers seeking to conduct comprehensive sequence analyses. The tool also supports similarity-based filtration, enabling the creation of non-redundant datasets suitable for machine learning-based peptide activity prediction, thereby helping to reduce bias and improve model generalization.

The software's emphasis on four core principles - convenience, efficiency, scalability, and accessibility - ensures that researchers can obtain rigorous, reproducible results without requiring specialized computational expertise or extensive software development. The complete workflow integration, from sequence parsing to result storage, democratizes access to high-performance sequence alignment capabilities across diverse research environments.

The modular architecture and transparent algorithms of SequenceAligner make it suitable for both high-performance research and educational use. Its clear component separation supports easy integration of alternative methods or optimizations, such as Parasail's functions, while preserving the benefits of a unified workflow. Combined with an open source, minimal dependency C99 codebase, this design encourages community-driven development and facilitates customization, aligning with the collaborative, reproducible ethos of computational biology.

Performance benchmark on three peptide datasets demonstrated the effectiveness of specialized all-versus-all optimizations, with alignment rates ranging from 27.4 to 80.8 million sequence pairs per second on consumer-grade NVIDIA GeForce RTX 4060. Furthermore, due to specialized optimizations, SequenceAligner performs up to 16.65 times more alignments in comparison to Parasail when all-versus-all sequence alignments are needed. The CUDA-accelerated GPU implementation outperformed a 16-thread CPU implementation by a factor of up to 8.59, highlighting the substantial benefits of hardware-specific optimization.

The adaptive memory management capabilities enable processing of datasets with results that exceed available system memory, making previously computationally prohibitive analyses feasible for routine research applications. The demonstrated scalability across different dataset sizes and hardware configurations, combined with cross-platform compatibility and straightforward deployment, makes SequenceAligner accessible to researchers with common computational resources, which is crucial for democratizing advanced sequence analysis capabilities and enabling broader participation in computational biology research.

SequenceAligner successfully bridges the gap between high-performance computing requirements and practical usability, providing the bioinformatics community with a tool that handles the scale and complexity demands of modern biological research while maintaining the accuracy, reproducibility, and extensibility essential for scientific applications. The software represents a contribution to the open-source computational biology ecosystem that prioritizes both performance and accessibility, fostering collaborative development and community-driven innovation in sequence analysis methodologies.

6. Acknowledgments

This project was supported by the Croatian Science Foundation/Hrvatska zaklada za znanost (grant no: UIP-2019-04-7999) and by the University of Rijeka (grant no: uniri-23-78 and uniri-23-16). The authors would like to thank the Parasail development team for providing substitution matrices that enabled comprehensive algorithm validation.

7. Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT to help improve sentence conciseness and check for grammatical errors. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the final content of the publication.

References

[1] C. Zhang, Q. Wang, Y. Li, A. Teng, G. Hu, Q. Wuyun, W. Zheng, The historical evolution and significance of multiple sequence alignment in molecular structure and function prediction, Biomolecules 14 (2024). URL: https://www.mdpi.com/2218-273X/14/12/1531. doi:10.3390/biom14121531.

[2] J. Chao, F. Tang, L. Xu, Developments in algorithms for sequence alignment: A review, Biomolecules 12 (2022). URL: https://www.mdpi.com/2218-273X/12/4/546. doi:10.3390/biom12040546.

[3] Uniprot: the universal protein knowledgebase in 2023, Nucleic acids research 51 (2023) D523–D531.

[4] D. A. Benson, K. Clark, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, E. W. Sayers, Genbank, Nucleic acids research 42 (2013) D32.

[5] B. E. Suzek, H. Huang, P. McGarvey, R. Mazumder, C. H. Wu, Uniref: comprehensive and non-redundant uniprot reference clusters, Bioinformatics 23 (2007) 1282–1288. URL: https://doi.org/10.1093/bioinformatics/btm098. doi:10.1093/bioinformatics/btm098. arXiv:https://academic.oup.com/bioinformatics/article-pdf/23/10/1282/49812789/bioinformatics_23_10_1282.pdf.

[6] M. Campillos, M. Kuhn, A.-C. Gavin, L. J. Jensen, P. Bork, Drug target identification using side-effect similarity, Science 321 (2008) 263–266. URL: https://www.science.org/doi/abs/10.1126/science.1158140. doi:10.1126/science.1158140. arXiv:https://www.science.org/doi/pdf/10.1126/science.1158140.

[7] A. M. Altenhoff, J. Levy, M. Zarowiecki, B. Tomiczek, A. Warwick Vesztrocy, D. A. Dalquen, S. Müller, M. J. Telford, N. M. Glover, D. Dylus, C. Dessimoz, Oma standalone: orthology inference among public and custom genomes and transcriptomes, Genome Research 29 (2019) 1152–1163. URL: http://genome.cshlp.org/content/29/7/1152.abstract. doi:10.1101/gr.243212.118. arXiv:http://genome.cshlp.org/content/29/7/1152.full.pdf+html.

[8] E. Otovic, M. Njirjak, D. Kalafatovic, G. Mausa, Sequential properties representation scheme for recurrent neural network-based prediction of therapeutic peptides, Journal of chemical information and modeling 62 (2022) 2961–2972.

[9] L. Wei, C. Zhou, H. Chen, J. Song, R. Su, Acpred-fl: a sequence-based predictor using effective feature representation to improve the prediction of anti-cancer peptides, Bioinformatics 34 (2018) 4007–4016.

[10] V. Boopathi, S. Subramaniyam, A. Malik, G. Lee, B. Manavalan, D.-C. Yang, macppred: a support vector machine-based meta-predictor for identification of anticancer peptides, International journal of molecular sciences 20 (2019) 1964.

[11] L. Yu, R. Jing, F. Liu, J. Luo, Y. Li, Deepacp: a novel computational approach for accurate identification of anticancer peptides by deep learning algorithm, Molecular Therapy Nucleic Acids 22 (2020) 862–870.

[12] S. Gull, N. Shamim, F. Minhas, Amap: Hierarchical multi-label prediction of biologically active and antimicrobial peptides, Computers in biology and medicine 107 (2019) 172–181.

[13] M. Njirjak, L. Žužić, M. Babić, P. Janković, E. Otović, D. Kalafatovic, G. Mauša, Reshaping the discovery of self-assembling peptides with generative ai guided by hybrid deep learning, Nature machine intelligence (2024) 1–14.

[14] M. Farrar, Striped smith–waterman speeds database searches six times over other simd implementations, Bioinformatics 23 (2006) 156–161. URL: https://doi.org/10.1093/bioinformatics/btl582. doi:10.1093/bioinformatics/btl582. arXiv:https://academic.oup.com/bioinformatics/article-pdf/23/2/156/49820591/bioinformatics_23_2_156.pdf.

[15] S. A. Manavski, G. Valle, Cuda compatible gpu cards as efficient hardware accelerators for smith-waterman sequence alignment, BMC bioinformatics 9 (2008) 1–9.

[16] J. Daily, Parasail: Simd c library for global, semi-global, and local pairwise sequence alignments, BMC bioinformatics 17 (2016) 1–11.

[17] P. J. Cock, T. Antao, J. T. Chang, B. A. Chapman, C. J. Cox, A. Dalke, I. Friedberg, T. Hamelryck, F. Kauff, B. Wilczynski, et al., Biopython: freely available python tools for computational molecular biology and bioinformatics, Bioinformatics 25 (2009) 1422.

[18] J. Köster, Rust-bio: a fast and safe bioinformatics library, Bioinformatics 32 (2016) 444–446.

[19] J. R. Rideout, G. Caporaso, E. Bolyen, D. McDonald, Y. V. Baeza, J. C. Alastuey, A. Pitman, J. Morton, Q. Zhu, J. Navas, K. Gorlick, J. Debelius, Z. Xu, M. Aton, llcooljohn, J. Shorenstein, L. Luce, W. V. Treuren, J. Chase, charudatta navare, A. Gonzalez, C. J. Brislawn, W. Patena, K. Schwarzberg, teravest, J. Reeder, I. Sfiligoi, shiffer1, nbresnick, D. K. D. Murray, scikit-bio/scikit-bio: scikit-bio 0.6.3, 2025. URL: https://doi.org/10.5281/zenodo.14640761. doi:10.5281/zenodo.14640761.

[20] The HDF5® Library & File Format - version 1.14.6., https://www.hdfgroup.org/solutions/hdf5/, Accessed 27-05-2025.

[21] S. Henikoff, J. G. Henikoff, Amino acid substitution matrices from protein blocks., Proceedings of the National Academy of Sciences 89 (1992) 10915–10919.

[22] D. Mo, A model of evolutionary change in protein, Atlas of protein sequence and structure, vol 5, suppl 3 (1978) 345–352.

[23] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, D. J. Lipman, Basic local alignment search tool, Journal of Molecular Biology 215 (1990) 403–410. URL: https://www.sciencedirect.com/science/article/pii/S0022283605803602. doi:https://doi.org/10.1016/S0022-2836(05)80360-2.

[24] M. Gorman, Understanding the Linux Virtual Memory Manager, Prentice Hall PTR, USA, 2004.

[25] S. B. Needleman, C. D. Wunsch, 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.

[26] O. Gotoh, An improved algorithm for matching biological sequences, Journal of molecular biology 162 (1982) 705–708.

[27] T. F. Smith, M. S. Waterman, et al., Identification of common molecular subsequences, Journal of molecular biology 147 (1981) 195–197.

[28] N. Thakur, A. Qureshi, M. Kumar, Avppred: collection and prediction of highly effective antiviral peptides, Nucleic Acids Research 40 (2012) W199–W204. URL: https://doi.org/10.1093/nar/gks450. doi:10.1093/nar/gks450. arXiv:https://academic.oup.com/nar/article-pdf/40/W1/W199/18783632/gks450.pdf.

[29] F. Desiere, E. W. Deutsch, N. L. King, A. I. Nesvizhskii, P. Mallick, J. Eng, S. Chen, J. Eddes, S. N. Loevenich, R. Aebersold, The peptideatlas project, Nucleic Acids Research 34 (2006) D655–D658. URL: https://doi.org/10.1093/nar/gkj040. doi:10.1093/nar/gkj040. arXiv:https://academic.oup.com/nar/article-pdf/34/suppl_1/D655/3924258/gkj040.pdf.