universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Acceleration of exact pairwise protein sequence alignment“

verfasst von / submitted by

Johannes Elias Tüchler BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2023 / Vienna, 2023

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 875
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Bioinformatik UG2002
Betreut von / Supervisor:	Univ.-Prof. Mag. Dr. Thomas Rattei

Abstract

The pairwise alignment of protein sequences is a central task in bioinformatics, which helps to identify biological relationships between protein sequences. Alignments can be performed with fast heuristics or with the exact Smith-Waterman algorithm, which is guaranteed to find the optimal local alignment between two sequences. Because of the quadratic time complexity of the Smith-Waterman algorithm, fast implementations are needed for large-scale protein similarity searches. Therefore, various parallelization approaches and adaptions for hardware accelerators (GPU, FPGA, Xeon Phi) have been developed for the Smith-Waterman algorithm.

In this work, the performance of some of the fastest CPU and GPU Smith-Waterman methods are compared under various parameter settings. The main result is that the fastest CPU method SWIPE outperforms the fastest GPU method CUDASW++ 3 on a full compute node with 24 CPU cores and 6 GPUs.

In the SIMAP (Similarity Matrix of Proteins) project, similarities between proteins of complete genomes are computed using the Smith-Waterman algorithm with compositional score adjustment. Because of the high runtime of the SIMAP workflow, in this work, possibilities of reducing the runtime are explored. Runtime profiling of the SIMAP workflow identified an inefficient compilation of a script and by optimizing the compilation process the runtime could be reduced to about a third.

Kurzfassung

Das paarweise Alignment von Proteinsequenzen ist eine zentrale Aufgabe der Bioinformatik, das dazu beiträgt, biologische Verwandtschaft zwischen Proteinsequenzen zu erkennen. Alignments können mit schnellen Heuristiken oder mit dem exakten Smith-Waterman-Algorithmus durchgeführt werden, welcher garantiert das optimale lokale Alignment zwischen zwei Sequenzen findet. Aufgrund der quadratischen Zeitkomplexität des Smith-Waterman-Algorithmus, sind für große Proteinvergleiche schnelle Implementierungen erforderlich. Daher wurden für den Smith-Waterman-Algorithmus verschiedene Parallelisierungsmethoden und Anpassungen für Hardwarebeschleuniger (GPU, FPGA, Xeon Phi) entwickelt.

In dieser Arbeit wird die Geschwindigkeit einiger der schnellsten Smith-Waterman-Methoden für CPU und GPU unter verschiedenen Parametereinstellungen verglichen. Das zentrale Ergebnis ist, dass die schnellste CPU-Methode SWIPE die schnellste GPU-Methode CUDASW++ 3 auf einem vollen Rechenknoten mit 24 CPU-Kernen und 6 GPUs übertrifft.

Im SIMAP-Projekt (Similarity Matrix of Proteins) werden Ähnlichkeiten zwischen Proteinen kompletter Genome mit Hilfe des Smith-Waterman-Algorithmus mit compositional score adjustment berechnet. Aufgrund der hohen Laufzeit des SIMAP-Workflows, werden in dieser Arbeit Möglichkeiten zur Verringerung der Laufzeit untersucht. Durch die Erstellung eines Laufzeitprofils für den SIMAP-Workflow wurde eine ineffiziente Kompilierung eines Skripts festgestellt und nach Optimierung des Kompilierungsprozesses konnte die Laufzeit auf etwa ein Drittel reduziert werden.

Contents

1 Introduction

The introduction is structured into five parts.

In 1.1, heuristic methods and the exact Smith-Waterman algorithm for pairwise protein sequence alignment are introduced. Parallelization strategies for accelerating the Smith-Waterman algorithm and commonly used hardware platforms (CPU, GPU, FPGA, Xeon Phi) are described.

Next, the compositional adjustment of substitution matrices which aims to improve the accuracy of protein sequence alignments is introduced 1.2.

In 1.3, the concept of orthology is described.

In 1.4, the SIMAP (Similarity Matrix of Proteins) project [1] is introduced. SIMAP represents a resource for pre-calculated protein similarities between complete genomes and is used by the orthology databases eggNOG [2] and STRING [3].

1.1 Protein sequence alignment

Protein sequence alignment is a widely used and powerful method for the comparison of proteins. Alignments can identify regions of similarity, which may be a consequence of biological relationships between the sequences. Protein sequences that share a common ancestor diverge over time by accumulating mutations followed by natural selection. The most common mutations at the protein level are the substitution of amino acids and the insertion or deletion of one or multiple amino acids [4]. A common application of protein sequence alignments is the inference of homologous proteins, and subsequently the inference of protein structure and function.

There exist various sequence alignment types. Pairwise sequence alignments are performed to compare two biological sequences in order to find conserved regions between the sequences. In the more sophisticated multiple sequence alignment, three or more sequences are compared in order to identify evolutionary relationships among a set of sequences. Furthermore, alignments can by global or local. In global alignment, the entire sequences are aligned (end-to-end alignment), which is only suitable if the compared sequences have approximately the same length and are rather similar [5]. Local alignments find regions with the highest similarity between the sequences. It is also suitable for more divergent or distantly related sequences, as it identifies conserved stretches while other potentially non-homologous parts of the sequences are not aligned.

In this thesis, the focus lies on pairwise local sequence alignments of proteins. The Smith-Waterman algorithm provides an optimal solution to this task, with the drawback of being rather runtime intensive. In order to speed up the alignment, various heuristics have been developed, which however are less sensitive than the Smith-Waterman algorithm.

In the following, heuristic approaches and the Smith-Waterman algorithm are described in more detail.

1.1.1 Heuristic alignment algorithms

Most heuristic pairwise alignment programs follow the seed-and-extend paradigm, where first short stretches of matching words ("seeds") between the compared sequences are searched, which are then extended to a full local alignment. With this approach, most pairs can be filtered out in the seed-searching phase, which speeds up the computation considerably. However, some hits, especially between distantly related sequences, might be missed due to this heuristic approach. The tradeoff between sensitivity and speed can be adjusted by tuning threshold parameters of the programs and this tradeoff is the most important measure for the comparison of the different methods. In the following, some of the most widely used heuristic programs for pairwise sequence alignment are described.

BLAST

The BLAST (Basic Local Alignment Search Tool) algorithm is one of the first implemented methods for similarity searches and is the most commonly used tool. BLAST has been developed by Altschul et al. in 1990 [6] and has been constantly revised since then [7, 8].

The BLAST algorithm is based on the observation, that significant alignments likely contain high-scoring segment pairs (HSPs), which are short local alignments without gaps between the two sequences. In the first step, a list of all words of a specified length that yield a score above a threshold when paired to the query sequence is generated. The scoring is based on a substitution matrix. Afterwards, the database sequences are scanned for exact matches to the words in the word list. An improved version of BLAST, called gapped BLAST [7] requires two word matches in the same diagonal within a specified distance in order to be a valid starting point for an extension. In a later improvement of the algorithm [8], the two-hit method was again replaced by a one-hit method, but it enables larger word sizes (and thus higher sensitivity) through the usage of a reduced alphabet. The matched region is extended in both directions without allowing gaps until the score drops below a threshold. This extended region represents an HSP - only HSPs which are statistically significant are kept. For all remaining HSPs, a gapped extension, parameterized by a substitution matrix and gap penalties, is performed, which represents the final alignment. One of the main advantages of BLAST is the statistical assessment of the significance of local similarities (Karlin-Altschul statistics [9]). The Expect value (E-value) describes how many hits one expects to find by chance in a similarity search given the scoring system and database size. The E-value is computed with the formula

$$E = Kmne^{-\lambda S}, \tag{1.1}$$

where m is the query sequence length, n is the database size, S is the alignment score, and K and λ represent parameters for the scoring system and the background amino acid frequencies respectively. All hits with E-values below a specified threshold are reported.

Besides the protein alignment program, the BLAST suite contains various programs for nucleotide and translated nucleotide searches. BLAST is considered to be one of the most sensitive heuristics, but it is relatively slow compared to other methods.

UBLAST

UBLAST [10] exploits the observation that similar sequences are likely to have multiple short words in common. For each query, the database sequences are sorted in order of decreasing number of unique words in common between the two sequences. For the first few sorted database sequences it is then tested whether they represent hits via an approach similar to BLAST. If several database sequences are rejected, the search is terminated. With this heuristic, alignments are constructed only for few database sequences, which makes it faster but also less sensitive compared to BLAST.

RAPSearch2

RAPSearch2 [11] is a fast protein similarity search tool and an improvement of RAPSearch [12]. In this method, the seeds are based on a reduced amino acid alphabet, where similar amino acids are clustered together and are represented by a single character [12]. The idea behind this, is that with the reduced alphabet the minimal seed length can be increased compared to the 20-amino acid alphabet. With this approach, about 3,000 times less seeds have to be evaluated, which however comes with a small loss in sensitivity. The heuristic extension algorithm is similar to BLAST.

DIAMOND

DIAMOND [13, 14] uses double indexing, where the seed locations in both the query and database sequences are determined. In contrast, most other methods use a single indexing of seed locations in the database. The double indexing increases the data locality and avoids any memory latency-related bottlenecks. Moreover, DIAMOND uses spaced seeds, where only a subset of the seed positions are used, and it uses a reduced amino acid alphabet. Spurious hits are gradually filtered out via various heuristic filter stages and potentially significant hits are aligned with the Smith-Waterman algorithm. According to the authors, DIAMOND reaches the same sensitivity as BLAST, with a speedup of 80-360x [14].

MMseqs2

MMseqs2 (Many-against-Many sequence searching) [15] is a software suite for fast searching and clustering of large datasets. The sequence search module consists of three consecutive stages with increasing sensitivity and decreasing speed. In the k-mer match stage, double similar-k-mer matches on the same diagonal are searched. In the ungapped alignment stage, alignments on the diagonals with double k-mer matches are performed with linear time complexity. The high-scoring alignments are passed to the final Smith-Waterman alignment stage. In all three search stages, local compositional bias correction

is performed, which assigns lower scores to matches involving overrepresented amino acids. Low-complexity regions in the database sequences are masked out during the k-mer matching and ungapped alignment stages. In a benchmark test performed by the authors, MMseqs2 reaches the sensitivity of BLAST while being 36 times faster [15].

1.1.2 Smith-Waterman algorithm

The Smith-Waterman algorithm [16] is a dynamic programming algorithm that computes the optimal local alignment between two sequences in O(mn) time, where m and n are the lengths of the sequences. It is parameterized by a substitution matrix and gap penalties. The substitution matrix assigns scores to each pair of amino acids, which describes the probability of an amino acid being replaced by another over evolutionary time. Commonly used substitution matrices, like BLOSUM [17] and PAM [18], are log-odds matrices and exist for various evolutionary distances. Gap penalties assign negative scores to gaps (insertions/deletions) in the alignment. The simplest gap penalty types, are the constant gap penalty, which assigns a fixed negative score to each gap, regardless of its length, and the linear gap penalty, which scores gaps as the product of a fixed score and the gap length. However, gaps which are multiple characters long, can result from a single evolutionary event, which is why it makes sense to penalize a single longer gap less than multiple gaps of length one. This is addressed by the affine gap penalty, which consists of a gap opening penalty (penalty for starting a gap) and a gap extension penalty (penalty for extending a gap by one) - the gap extension penalty is usually set to be lower than the gap opening penalty. The affine gap penalty yields biologically realistic alignments and is by far the most commonly used gap penalty type - it is implemented in all Smith-Waterman implementations described in this thesis. The modifications to the Smith-Waterman algorithm by Gotoh [19], enable the scoring with the affine gap penalty while maintaining a time complexity of O(mn). There exist also other gap penalty types, like double affine, logarithmic, log-affine, or convex, which are however rarely included in Smith-Waterman implementations [20].

The recurrence of the Smith-Waterman algorithm with affine gap penalties is defined as

$$
\begin{aligned}
H_{i,j} &= max \begin{cases} H_{i-1,j-1} + M(q[i], d[j]) \\ E_{i,j} \\ F_{i,j} \\ 0 \end{cases} \\
E_{i,j} &= max \begin{cases} E_{i,j-1} - G_{ext} \\ H_{i,j-1} - G_{open} - G_{ext} \end{cases} \\
F_{i,j} &= max \begin{cases} F_{i-1,j} - G_{ext} \\ H_{i-1,j} - G_{open} - G_{ext} \end{cases}
\end{aligned}
\tag{1.2}
$$

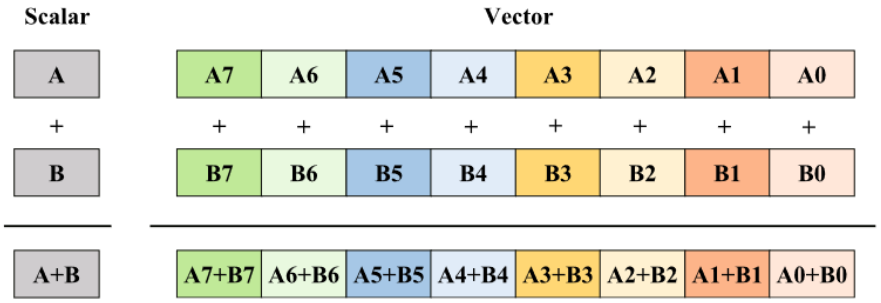

Figure 1.1: **Comparison of scalar and SIMD vector operations.** (Figure from [22])

The query sequence q contains residues q_i and the database sequence d contains residues d_j. The substitution matrix M contains scores for aligning two amino acids $M(q[i], d[j])$. The costs for opening a gap and extending a gap are denoted by G_{open} and G_{ext} respectively. The recurrence is initialized as $H_{0,j} = H_{i,0} = E_{0,j} = F_{i,0} = 0$ for $0 \leq i \leq |q|$ and $0 \leq j \leq |d|$. $H_{i,j}$, $E_{i,j}$ and $F_{i,j}$ store the local optimal alignment scores of the two prefixes q_i and d_j. E and F give the score for ending with a gap along q and d respectively. The highest value in H gives the overall optimal local alignment score. In order to obtain the optimal alignment path, a backtracking procedure is performed, by starting from the highest scoring cell and following the optimal path until a cell with a score of zero is reached [21].

Each cell in the alignment matrix is dependent on the upper, upper-left, and left cell neighbors. This dependency determines the order in which the values of the matrix can be computed.

1.1.3 Parallelization of the Smith-Waterman algorithm

Various parallelization strategies for the Smith-Waterman algorithm have been developed, with the goal to reduce the runtime of the alignment task. The parallelization approaches can be divided into vector-level, thread-level and process-level parallelization [22].

Vector-level parallelization can be achieved via the Single Instruction Multiple Data (SIMD) method. With SIMD, instead of performing multiple sequential scalar operations, a single operation can be performed on a whole data vector simultaneously Figure 1.1. The SIMD paradigm can be used on CPU, GPU and FPGA architectures.

Thread-level parallelization describes the parallel execution of multiple threads. Multi-threading can be implemented via tools like POSIX threads (pthreads), which is a low-level API, or OpenMP [23], which is a higher-level API.

Process-level parallelization describes the parallel computation on distributed memory systems - it enables the parallel computation on multiple compute nodes. This approach is implemented via the Message Passing Interface (MPI) standard [24].

Thread-level and process-level parallelization can also be achieved manually, by dividing the alignment workload in multiple chunks which then can be run in parallel. In the

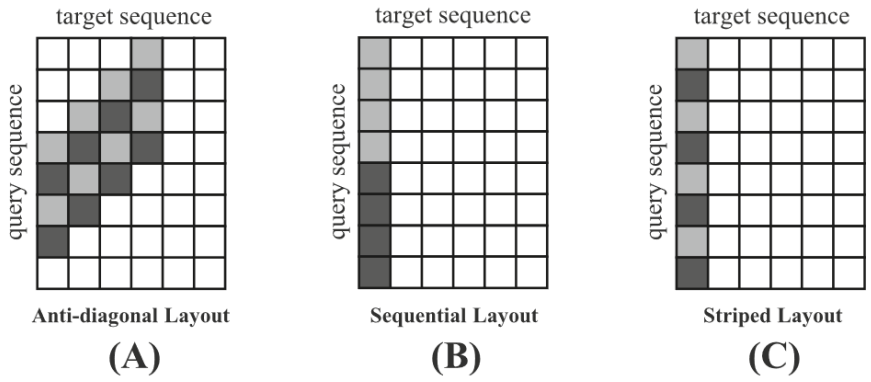

Figure 1.2: **Intra-sequence parallelization layouts.** The three layouts differ in the way the SIMD vectors (cells of the same color) are mapped to the matrix. (Figure from [22])

following, intra-sequence and inter-sequence vector-level parallelization approaches for the Smith-Waterman algorithm are discussed in more detail.

1.1.3.1 Intra-sequence parallelization

Intra-sequence parallelization accelerates the alignment of a single sequence pair [25]. The parallelization can be achieved by computing multiple cells of the dynamic programming matrix in parallel via a SIMD vector. These SIMD vectors can be mapped to the matrix in three different ways, as depicted in Figure 1.2.

In the *anti-diagonal layout* or *wavefront layout* (first introduced by Wozniak [26], Figure 1.2A), the SIMD vectors consist of cells from one anti-diagonal. This approach has the advantage, that the elements from one vector are completely independent of each other and can therefore be computed in parallel. However, as the length of the computed anti-diagonal is not always divisible by the fixed length of the SIMD vector, the vectors need to be filled with dummy symbols, which results in a waste of compute resources [22]. Furthermore, the loading of values along the anti-diagonal is less efficient compared to vectors parallel to the query sequence [27].

In the *sequential layout* (presented by Rognes and Seeberg [27], Figure 1.2B) the SIMD vector is parallel to the query sequence in a sequential manner. Because the elements of a vector are not completely independent of each other (each element is dependent on the element above it), at first, this dependency is ignored and any errors are corrected afterwards. In this error correction loop, called Lazy-F evaluation (also referred to as SWAT-like optimization in [27]), any errors in the F matrix and subsequently in the H matrix are corrected. In most cases, the F-values are below a threshold and thus can be ignored for the calculation of H-values, which greatly simplifies the calculation - otherwise, the somewhat time-consuming error correction has to be performed and it is checked whether a vertical gap yields a higher score than the initial stored score [27].

Also in the *striped layout* (developed by Farrar [28], Figure 1.2C) the SIMD vector is parallel to the query sequence, but this time in a striped manner. Compared to the sequential layout, the error correction loop of the striped layout is more efficient, because in the striped layout the data dependencies are moved from the inner loop (processing the query sequence) to the outer loop (processing the database sequence) [28, 22]. This means that the error correction has to performed only n times instead of mn times, where m and n are the lengths of the query and database sequence respectively.

Farrar [28] performed a runtime comparison of the anti-diagonal [26], sequential [27] and striped [28] layout implementations. With a gap open penalty of 10, gap extension penalty of 1 and BLOSUM50 (BLOSUM62) substitution matrix, the achieved speeds in MCUPS (million cell updates per second) are: anti-diagonal 351 (352), sequential 374 (816), and striped 1,817 (2,553). The performance of the anti-diagonal layout is not affected by the scoring system. The error correction loop runtime of the sequential and striped layout is affected by the scoring system - it takes longer if the H-values are high and the gap penalties are small [28]. Overall, the striped layout clearly performs the best, followed by the sequential layout.

1.1.3.2 Inter-sequence parallelization

In contrast to the intra-sequence parallelization, the inter-sequence parallelization approach does not speed up a single alignment, but it processes multiple alignment pairs in parallel. Therefore, this parallelization approach can only be applied if multiple alignments need to be performed, but this is the case for most alignment tasks. Different layouts of parallelizing the alignments are possible - commonly, either many one-to-one alignments are performed, or many sequences are compared to one sequence (many-to-one layout).

1.1.4 Hardware platforms for the Smith-Waterman algorithm

Smith-Waterman implementations have been developed for different hardware platforms (CPU, GPU, FPGA, Xeon Phi). In the following, the characteristics of the hardware platforms are described and some of the best performing Smith-Waterman tools for protein alignment are presented.

1.1.4.1 CPU

The central processing unit (CPU) is the central part of the computer, which performs all types of data processing operations. The basic components of a CPU are the control unit (orchestrates the execution of instructions), the arithmetic logic unit (performs arithmetic and bitwise operations) and the registers (quickly accessible storage). While the processor speed continuously increased over time due to increasing transistor counts and clock speed, more recently the trend goes towards increasing the compute capability with multi-core processors [29]. SIMD parallel processing on CPUs can be achieved via instruction set extensions, such as Streaming SIMD Extensions (SSE) and Advanced Vector Extensions (AVX).

CPU Smith-Waterman implementations

SSW (Striped Smith-Waterman) [30] is an implementation for nucleotide and protein sequences, which is available as a C/C++ library as well as a stand-alone tool. As the name implies, SSW is an intra-sequence method, which uses the striped layout. It is parallelized via SIMD and makes use of the Streaming SIMD Extensions 2 (SSE2) instruction set. SSW is based on Farrar´s Striped Smith-Waterman [28] and SWPS3's [31] improvement, which got extended by including alignment information additional to the optimal alignment score. It utilizes a query profile, which is calculated once for each query sequence and serves as a query-specific substitution matrix, which minimizes memory look-ups of substitution scores. Query/score profiles are commonly used techniques in alignment tools [27, 28, 32, 25].

SWIPE [32] follows an inter-sequence SIMD parallelization approach for the alignment of nucleotide and protein sequences. It was mainly written in C++ and partly in assembly language, and it utilizes the SSE2 and SSSE3 instruction sets. In SWIPE, sixteen database sequences are aligned in parallel to one query sequence, which means it follows the many-to-one layout. The alignments are initially performed with a 7-bit routine, which is able to calculate scores up to 127. In case of an overflow, an alignment is recalculated with a wider score range using 16 bits or 63 bits.

Two methods which utilize the rather new AVX2 instruction set are the Parasail library [33] which contains a Smith-Waterman implementation based on SSW, and libssa [34], which follows a similar approach as SWIPE.

1.1.4.2 GPU

In the past, GPUs have been typically used to render computer graphics. Nowadays they are also applied to various other tasks by performing general-purpose computing on GPUs. While the CPU is latency-optimized and suitable for general computing, the GPU is throughput-optimized and designed for specific compute tasks. A GPU consists of a high number of cores, which enables its high parallel capability. With the development of high-level languages, such as CUDA [35] and OpenCL [36], GPU programming became more accessible for the scientific community.

Many GPU Smith-Waterman implementations, make use of the CUDA framework, which is a parallel computing platform and API for NVIDIA GPUs. In a GPU program, part of the code runs on the CPU (host code) and another part is offloaded to the GPU via kernels that are launched from the CPU. In CUDA, each kernel is executed by multiple threads in parallel. Threads are grouped into blocks, and blocks are grouped into a grid. Before the computation, data has to be transferred from the CPU to the GPU, and after the kernel has finished the results have to be copied back to the CPU. GPUs have a complex memory hierarchy consisting of various memory levels with different access and speed. When writing GPU programs, an optimized memory management is important for achieving a good performance.

GPU Smith-Waterman implementations

CUDASW++ 3 [25] utilizes both CPU and GPU for the Smith-Waterman calculations. First, the sequence alignment workload is distributed over all available CPUs and GPUs based on their respective compute power. The CPU method is based on SWIPE - it performs inter-sequence SIMD parallelization by aligning sixteen database sequences to one query sequence in parallel. The GPU method is implemented conceptually similar to the CPU method - it uses a GPU SIMD parallelization via CUDA PTX SIMD video instructions and performs four alignments in parallel per thread. CUDASW++ 3 is an improvement compared to CUDASW++ 2 [37], which uses only the GPU for the Smith-Waterman calculations and uses the SIMT (single instruction, multiple threads) execution model which enables less data parallelism compared to SIMD.

ADEPT [21] uses only the GPU for the Smith-Waterman calculations, while the CPU is only involved in data preparation and data transfer between CPU and GPU. The algorithmic approach follows the intra-sequence parallelization strategy with anti-diagonal layout.

1.1.4.3 FPGA

A field-programmable gate array (FPGA) is an integrated circuit which can be configured for specific tasks. It consists of an array of logic blocks, an interconnection network, I/O blocks and memory blocks [38]. The wiring of the logic blocks can be programmed by the user ("field-programmable") in order to create custom instruction pipelines for the performed computation. FPGAs can also be flexibly re-programmed as needed. Compared to CPUs and GPUs, FPGAs work on much lower clock frequencies and reach lower peak performances, which is however often compensated by an increased performance efficiency and lower energy consumption [39, 29]. FPGAs are capable of massively parallel operations and can be used to accelerate specific parts of an algorithm. A drawback of FPGAs is that its programming is very different from CPU or GPU programming and thus difficult for many users. The configuration of FPGAs is usually specified via a hardware description language, like VHDL and Verilog. However, more recent FPGAs also support the high-level language OpenCL (based on the C programming language), which reduces the programming costs and increases the portability of FPGA code [40].

FPGA Smith-Waterman implementations

OSWALD [40] is an implementation for protein Smith-Waterman database searches based on Altera FPGAs. It follows a hybrid approach, which exploits both CPU and FPGA simultaneously for the Smith-Waterman calculations. The CPU method utilizes SIMD computing via SSE and AVX2 instruction sets, and the FPGA algorithm makes use of pipeline and vectorial parallelism. The Smith-Waterman implementation follows the inter-sequence parallelization scheme. OSWALD uses the OpenCL parallel programming framework, which launches the kernels on the target device and handles the memory management.

Some other FPGA Smith-Waterman implementations make use of the systolic array parallel processing approach [41, 42, 43]. SWIFOLD [44] and a proposed algorithm by Oliveira et al. [45] represent methods for DNA alignment.

1.1.4.4 Xeon Phi

The Xeon Phi is a x86-based manycore processor developed by Intel which was launched in 2010. It provides high multithreading capabilities, as Xeon Phi models contain up to 72 cores and each core is able to run four hardware threads. It can be used as a coprocessor chip or as a standalone CPU. Compared to GPU and FPGA accelerators, Xeon Phi has the advantage that it is x86-compatible and has support for OpenMP and MPI paradigms - thus it can run software developed for standard CPUs. Furthermore, it supports a subset of Intel´s latest AVX-512 instruction set which enables the use of 512-bit SIMD vectors. However, it does not support AVX-512BW, which is needed for low-range integer representations (8-bit), which would be optimal for Smith-Waterman implementations [32, 46]. The Xeon Phi series was discontinued in 2020.

Xeon Phi Smith-Waterman implementations

SWIMM 2.0 [46] is an protein Smith-Waterman implementation based on AVX-512 vector extensions. The method exploits data-level and thread-level parallelism. Data-level parallelism is achieved via an inter-sequence SIMD approach with support for different instruction sets (SSE4.1, AVX2, AVX-512F and AVX-512BW). Multithreading on the multicore or manycore architectures is achieved via OpenMP. SWIMM 2.0 was tested by the authors on two architectures: a Xeon Skylake CPU and a Xeon Phi. On the Xeon Skylake processor, AVX-512BW and AVX2 achieved similar performances - the double vector capacity by AVX-512BW is compensated by doubling the number of simultaneous instructions by AVX2. On Xeon Phi, AVX2 performs the best - AVX-512F performs clearly worse because of the lack of 8-bit integer operations.

SWAPHI [47] is another Smith-Waterman protein database search method designed for Xeon Phis. It makes use of AVX-512 extension by using SIMD vectors split into 16 32-bit-wide lanes. The authors tested an intra-sequence model (based on Farrar´s striped approach) and an inter-sequence model (similar to CUDASW++ 3), of which the latter performed better.

1.2 Compositional adjustment of substitution matrices

Amino acid substitution matrices are central for protein alignment methods, which score alignments based on substitution and gap scores. The most commonly used substitution matrices, such as PAM and BLOSUM matrices, are constructed as log-odds matrices. These are based on a statistical theory for ungapped local alignments, which assumes a random protein model in which amino acids occur independently with background frequencies p [48]. The score for an amino acid pair can be written in the form

$$s_{ij} = \frac{1}{\lambda} ln(\frac{q_{ij}}{p_i p_j}), \tag{1.3}$$

where q represents the target frequencies which are always positive and sum to 1, and λ is a positive scale factor for the matrix. Valid substitution matrices must have a negative expected score and at least one positive entry [48]. Background and target frequencies are derived from curated collections of alignments of homologous proteins. However, the standard substitution matrices are only optimal for the comparison of proteins with standard compositions (amino acid frequencies). For the comparison of proteins with biased compositions these matrices are not ideal [49]. Proteins with nonstandard amino acid compositions are not uncommon - examples are hydrophobic or cysteine-rich proteins and proteins encoded by AT- or GC-rich genomes [49, 50].

The compositional adjustment of substitution matrices yields matrices which are better suited for the comparison of proteins with divergent compositions [49]. The method described here has been developed by Altschul et al. [48, 51, 49]. The method takes the two compared sequences and a standard substitution matrix as an input, and transforms the matrix into an adjusted matrix which is tailored to the compositions of the two compared sequences. This task can be formulated by starting with a set of target frequencies q (as given by the standard substitution matrix) and two sets of background frequencies P and P' which are inconsistent with q. We then seek new target frequencies Q_{ij} which are as close to the original target frequencies q_{ij} as possible, while satisfying the consistency conditions

$$P_i = \sum_j Q_{ij}; \quad P'_j = \sum_i Q_{ij}. \tag{1.4}$$

In contrast to standard substitution matrices with a single set of background frequencies, this method yields asymmetric target frequencies and thus asymmetric substitution scores, which means that in the general case $Q_{ij} \neq Q_{ji}$. The compositional adjusted matrices with the background frequencies P and P' can be written as

$$S_{ij} = \frac{1}{\lambda} ln(\frac{Q_{ij}}{P_i P'_j}). \tag{1.5}$$

For the comparison of compositionally biased proteins, the compositionally adjusted matrices yield on average improved alignments and alignment scores [49]. However, according to the authors, it is not advisable to apply compositional adjustment universally to all alignment pairs, but only if at least one out of the following four criteria is fulfilled (conditional compositional adjustment mode) [49]:

- *Length ratio*: Compositional adjustment is performed if the sequence length ratio of the longer to the shorter sequence is less than 3. For higher length differences, the longer sequence might contain domains which are not present in the shorter sequence, which makes the compositional adjustment unreliable.

- *Compositional distance*: If the two compared sequences have rather similar compositions, compositional adjustment is performed.
- *Compositional angle*: Compositional adjustment is performed if the two sequences show a similar compositional drift compared to the standard amino acid composition.
- *High pair frequencies*: Compositional adjustment is performed if in either of the two sequences the two most frequent amino acids account for a high proportion of the total composition.

If all criteria fail for an alignment pair, composition-based statistics [52] are applied to that pair. With composition-based statistics the whole substitution matrix is scaled by a constant factor, which depends on the sequence compositions.

1.3 Orthology inference

The identification of homologous relationships of genes is central for comparative and evolutionary genomics. Homologous genes can be further subdivided according to their type of evolutionary relationship. Of great importance is the distinction between orthologs and paralogs as defined by Walter Fitch [53]. Homologs resulting from a speciation event are referred to as orthologs, while those resulting from a duplication event are called paralogs. Orthologs are of particular importance because they can be seen as 'the same gene' in different species [54]. In most cases, they retain similar function and therefore they are essential for the functional annotation of genomes. In contrast, paralogs frequently tend to adopt different functional roles [55]. Moreover, the detection of orthologs plays a key role in phylogenomics because they can be used to infer species trees [56].

There exist various orthology databases, which differ in the applied method to infer the orthologs and in the covered taxonomic range. The Quest for Orthologs consortium [57] compiled a comprehensive list of orthology database resources, which is available on their website (`https://questfororthologs.org/orthology_databases`).

1.3.1 Methods for orthology detection

The methods for orthology inference can be classified into tree-based methods and graph-based methods.

Tree-based methods involve explicit models of the evolutionary history of genes in the form of gene trees. The most common approach is to combine such gene trees with species trees via an approach called tree reconciliation, in order to distinguish orthologs and paralogs [58]. The tree construction is performed either via distance-based (neighbor-joining or UPGMA) or character-based (maximum parsimony or maximum likelihood) methods [56].

Graph-based methods are based on pairwise sequence similarity searches. The approach relies on the assumption, that a gene is more similar to its orthologous counterpart in another species than to any other gene [59]. The most common approach to infer orthologs

is the bidirectional best hit method, which assigns orthologous relationships via symmetric best-matches [60]. This method requires a list of pairwise alignment scores between the sequences of two genomes as an input, which is most commonly obtained via the BLAST heuristic or the exact Smith-Waterman algorithm. There exist several clustering methods in order to group orthologs from multiple genomes into orthologous groups.

Besides, there are also hybrid approaches which include phylogenetic trees as well as sequence similarity-based methods [54]. There exist also approaches which additionally make use of synteny information [54]. Synteny describes the conservation of local gene order.

1.3.2 Comparison of orthology detection methods

Each method for the detection of orthology has its advantages and disadvantages and the best choice depends on the use-case. In general, tree-based methods tend to be more specific, while graph-based methods tend to be more sensitive [54]. The construction of trees is computationally expensive and is therefore only feasible for smaller datasets [54]. Furthermore, tree-based methods have difficulties when genome evolution does not follow a tree-like pattern, which is often the case for prokaryotes, where horizontal gene transfer represents an important evolutionary factor [61]. A challenge for graph-based methods, is the detection of differential gene loss. A paralogous pair might be falsely identified as orthologous, if the corresponding true orthologs of the respective genes are missing in both species [62]. Graph-based methods are easier to automate compared to tree-based methods and can also handle large number of genomes [54].

1.4 SIMAP: the similarity matrix of proteins

Protein sequence similarity searches are a fundamental method in bioinformatics, which build the basis of various bioinformatic tasks, such as the functional annotation of proteins or the prediction of orthologous groups [63]. As the task of all-against-all protein similarities of large databases is very computationally intensive, the Similarity Matrix of Proteins (SIMAP) database aims to provide a pre-calculated similarity matrix [1].

The SIMAP 1 project run from 2004 to 2014. Its initial concept was to provide sequence similarities of proteins from public databases and completely sequenced genomes. For the computation, a two-step algorithm was applied - first, putative hits were searched for with the FASTA heuristic [64], and then the remaining hits were re-calculated with the exact Smith-Waterman algorithm. In later SIMAP versions, its functionality was extended by including sequence-based features such as Interpro domains [65].

The database was frequently updated with newly available sequences. In these incremental updates, each new sequence had to be aligned to itself and to all sequences of the database, while the similarities between the old sequences did not have to be re-computed. Nevertheless, with the exponential growth of the number of known protein sequences and the resulting quadratic growth of the SIMAP database, a comprehensive database of all known proteins became infeasible [63].

The SIMAP 2 project started in 2015 and focuses on the main use case of SIMAP 1, which was comparative genomics. It computes sequence similarities between proteins of complete genomes, which forms the basis for large-scale inference of orthologous groups, e.g. in eggNOG [2] and STRING [3]. The algorithm for the computation of sequence similarities was modified, with the goal to become more sensitive, which means that distantly related hits are found more frequently. Instead of filtering the hits with the FASTA heuristic the new approach is based solely on the non-heuristic Smith-Waterman algorithm. Furthermore, it includes compositional score matrix adjustment [48, 51, 49], which is also used by default in protein-protein BLAST [6].

1.4.1 SIMAP workflow

The current SIMAP workflow consists of three alignment phases and is described below in more detail.

Alignment phase 1

In the first alignment phase, all pairwise sequence pairs are computed with the Smith-Waterman algorithm, which is parameterized by a substitution matrix, a gap open penalty and a gap extension penalty. After this step, all pairs with scores below a specified threshold are discarded. The computation is based on SWIPE and uses only the 7-bit routine, which means that the scores are only calculated up to 127, which is sufficient to test against the threshold which is lower than 127. In this alignment phase, the low-complexity regions of query and database sequences are translated to 'X'. Low-complexity filtering (performed by the program SEG [66]) prevents spurious hits because of compositionally biased regions and therefore improves the reliability of the similarity search.

Alignment phase 2

In the second alignment phase, the remaining pairs are computed again with the Smith-Waterman algorithm, but this time with compositionally adjusted substitution matrices. After this step, again all pairs below another threshold, which is higher than the phase 1 threshold, are discarded. The compositional score adjustment yields improved alignments when applied to the comparison of proteins with biased composition [49], but it also requires additional execution time, which is of particular importance in large-scale analyses. Therefore, low-scoring pairs are filtered out in phase 1 - these pairs are very unlikely to score higher than the second threshold with compositional score adjustment applied. The two thresholds affect the tradeoff between sensitivity and runtime - lower thresholds lead to more true positive hits, but also increase the runtime of the workflow. As in this phase, each pair has a unique substitution matrix, the computation is performed with SSW instead of SWIPE. Also in this alignment phase, the low-complexity regions of the query and database sequences are masked. Here, the SIMAP workflow differs from BLAST, where only the database is masked, which introduces asymmetry (scores might change

if the query and database sequence are swapped). However, in SIMAP the symmetry is important for two reasons - on the one hand, only half of the matrix (i.e. the upper triangle) has to be computed which halves the overall workload, on the other hand, it would not make sense to have two different scores for the same sequence pair.

Alignment phase 3

In the final alignment phase, the final score and alignment attributes are computed. The score computation uses the default BLAST parameters (BLOSUM62, gap open: 11, gap extension: 1) and is performed with SSW (for scores below 32,767) or SWIPE. The alignment attributes (sequence identity, sequence similarity, alignment start and end positions) are computed with SWIPE´s align function. For the compositional matrix adjustment, both sequences are masked, while for the score calculation both sequences are unmasked. All pairs are kept in this phase, and a result file with score and alignment attributes of each pair is created.

1.5 Problem description and research goals

The SIMAP project computes sequence similarities between proteins of complete genomes, which is a prerequisite for orthology inference. The method utilizes the exact Smith-Waterman algorithm, which provides sensitive results, but has the drawback of being runtime-intensive. The current SIMAP workflow was implemented in 2014. At that time, the focus lay on the correct implementation of a workflow which fully relies on the Smith-Waterman algorithm and makes use of compositional score adjustment, while less attention was given to runtime optimization. The goal of this master project is to improve the runtime of the SIMAP workflow, so that future SIMAP calculations remain computationally feasible.

The first aim of this project, is to compare the runtime of the fastest available Smith-Waterman implementations for protein sequence alignment. This runtime comparison forms the basis for deciding which alignment tool is the best choice for SIMAP.

The next aim of this project, is to profile the SIMAP workflow, so that runtime-intensive components of the workflow can be identified. Subsequently, modifications to the SIMAP code with the aim of reducing the runtime should be implemented. The final SIMAP version needs to be well tested, so that the correctness of future SIMAP calculations is ensured.

2 Data and Methods

All plots in this thesis have been created in Python with the matplotlib library.

2.1 Smith-Waterman comparison

2.1.1 Swiss-Prot sequence data

The UniProtKB/Swiss-Prot database (release 2022_02) [67] in FASTA format was downloaded from `https://ftp.uniprot.org`. Swiss-Prot is a manually curated non-redundant database of protein sequences.

All sequences which contain non-canonical or ambiguous amino acids are excluded from the analysis. The Smith-Waterman programs handle these special amino acids differently and thus would yield differing results. On the filtered dataset with only canonical amino acids, the programs should yield identical scores for each pair, which makes the verification of the correctness of the programs straightforward. Sequence statistics of the unfiltered and filtered Swiss-Prot database are depicted in Table 2.1.

	Swiss-Prot (unfiltered)	Swiss-Prot (canonical only)
Sequence number	567,483	564,823
Total amino acids	204,940,973	203,990,116
Length 1st quartile	169	169
Length median	295	295
Length 3rd quartile	449	449

Table 2.1: Sequence statistics of the Swiss-Prot database for the unfiltered database (left) and without non-canonical amino acid containing sequences (right).

2.1.1.1 Random sampling

For the various Smith-Waterman performance tests, random samples from the Swiss-Prot dataset were drawn via the bioawk tool [68]. First, the dataset was filtered by the specified sequence length range. As a next step, the specified number of sequences were randomly selected. In cases, where the number of desired sequences was greater than the number of sequences in the dataset, multiple copies of the same sequence were allowed - in order to keep the headers in the created dataset unique, a number was added to it.

2.1.2 Smith-Waterman program installation and parameters

SSW

The SSW software (v1.1) [30] was downloaded from `https://github.com/mengyao/Complete-Striped-Smith-Waterman-Library` and compiled as described in the README. The provided Python interface, which is according to the authors about as fast as the C/C++ interface, was used for calling the program. The pyssw.py script was modified to only output results of pairs scoring higher than a specified threshold. Furthermore, the ssw_lib.py script was extended with the BLOSUM45/50/62/80 substitution matrices.

SWIPE

The SWIPE (v2.1.0) [32] executable (source: `https://github.com/torognes/swipe`) was used. The maximum expect value to show in the output was set to 1e6, the maximum number of alignments and descriptions to show was set to the number of database sequences.

CUDASW++ 3

The source code of CUDASW++ (v3.1.2) [25] has been downloaded from `https://cudasw.sourceforge.net` and has been compiled following the instructions on the website using CUDA 11.6. The default query profile mode has been used for all calculations - the query profile variant mode did not run successfully.

ADEPT

The ADEPT library (v1.0) [21] was downloaded from `https://github.com/mgawan/ADEPT` and has been built with including the Python module of ADEPT using CUDA 11.6. The example script py_asynch_protein.py served as a template for calling the program. The input sequence file reading was modified so that all-against-all alignments between query and database sequences are performed. The calculation was performed in batches of size 50,000 which is about the maximum possible batch size to run successfully.

2.2 SIMAP testing

The original SIMAP code was downloaded from `https://github.com/tolot27/swipe/tree/sswlib` - the new version is available at `https://github.com/JoTue/swipe/tree/sswlib_2022`. The SIMAP code was compiled with the newest BLAST library (v2.13.0). All SIMAP calculations were performed with following (default) parameters:

-M BLOSUM50 -G 13 -E 2 -m 88 -s2 -b <#db_sequences> -v <#db_sequences>

2.2.1 Sequence data

Swiss-Prot

The same UniProtKB/Swiss-Prot database (release 2022_02) as in the Smith-Waterman comparison tests was used. Here, sequences with non-canonical characters were not filtered out.

ASTRAL40

The ASTRAL40 dataset (v1.55) [69] was downloaded from `https://scop.berkeley.edu/astral/`. It contains 2,577 protein domains with less than 40% identity to each other.

COG

The Clusters of Orthologous Genes (COG) database (2020 version) [70] was downloaded from `https://ftp.ncbi.nih.gov/pub/COG/COG2020/`.

2.2.2 Data preparation

For all used protein sequence datasets, the low complexity regions were soft-masked (converted into lowercase) via the tool SEG (with parameters: window size = 10, low (trigger) complexity = 1.8 and high (extension) complexity = 2.1). In SIMAP, both the query and the database are masked - otherwise the results would not be symmetrical. SIMAP takes three input files as an input, the soft-masked query, the soft-masked database, and a BLAST index of the hard-masked (lowercase characters converted to X) database. The program makeblastdb (v2.9.0+) of the BLAST+ software suite was used to create the BLAST index (-blastdb_version 4). The translation of lowercase characters into X or uppercase was performed in Python with the Biopython module [71].

In all three alignment phases, hard-masked query and database sequences are used. Only in the final score calculation step of phase 3, query and database sequences are internally unmasked (all characters in uppercase).

3 Results

3.1 Smith-Waterman performance comparison

The exact Smith-Waterman algorithm is a widely used local pairwise sequence alignment method. Various implementations have been developed, which are often tailored to a specific use case. In this project, the focus lies on the comparison of accelerated programs which are suitable for protein database searches, as it is for example performed in the SIMAP project. Four alignment tools are compared in more detail: the CPU methods SSW and SWIPE, and the GPU methods CUDASW++ 3 and ADEPT - a description of these methods can be found in 1.1.4. These are some of the most widely used and best performing Smith-Waterman tools for protein database searches. Methods for FPGA and Xeon Phi are not included because these hardware platforms are not available to me for testing. Regarding the CPU methods with AVX2 support, I was not able to compile libssa and SWIMM 2.0 (needs Intel compiler). Parasail performed worse than SSW (function: parasail_sw_striped_profile_avx2_256_sat) in preliminary tests and was therefore not included in the detailed comparison.

Usually, the publications of the alignment programs contain runtime comparisons to other tools. These comparisons can give a good first impression of a tool's performance, however, the alignment parameters, used hardware, and compared tools are commonly chosen so that the own program performs the best. For an objective assessment of the performance of different programs, a performance comparison framework was created, which is described in the following.

3.1.1 Framework for Smith-Waterman implementation comparison

The aim is to create a program that enables reproducible runtime comparisons of alignment programs across different alignment parameter settings. The software consists of multiple Python scripts and the repository is available on GitHub: `https://github.com/JoTue/protein_alignment`. In the following, the various steps and functionalities of the program are presented.

Input file preparation

The program takes query and database files in FASTA format as an input. These FASTA files are converted to a layout which is suitable for all alignment programs. As ADEPT fails to read multi-line records, all sequences are converted to a single line. Furthermore, pipe characters in the header lead to different parsing of the header by the alignment programs, which is why they are replaced by underscores. BLAST and SWIPE require

a BLAST database of the database FASTA file as an input, which is created via the makeblastdb command line tool from the BLAST suite. If multi-threading via xargs is specified, the query file is split into multiple files. After splitting, each file approximately contains the same number of residues, which assures that the workload for each thread is similar.

Alignment

The compared alignment programs are available as command line tools. For each program, a Python wrapper was written which calls the program with the specified alignment parameters. One can specify the used substitution matrix, the gap open/extension penalties, the minimum score of hits to be reported and the number of used CPU cores and GPUs.

Before the alignment, the input files are copied to a specified temporary directory (e.g. /tmp or /dev/shm). Reading from and writing to a temporary file system with high read/write speed is important for a proper comparison. Alignment programs which are I/O intensive (like SWIPE) would be more affected by slow read/write speeds than other programs, which would severely bias the comparison.

Checking correctness of results

Checking that all alignment tools yield the correct results is an important part of the comparison framework. One the one hand, this validates the correctness of the algorithms, and on the other hand, it verifies the correct compilation and program invocation by the user.

The comparison framework contains output file parsers for each alignment tool. These convert the various output file formats into an unified format, which lists all hits with the sequence header names and the alignment scores. In the comparison framework, the results are checked by comparing the results of the different alignment tools with each other. If the results of all programs agree, this indicates that the results are correct. There are different levels of checking the results implemented. If an exact verification is required, the scores of all hits are compared. As this verification is quite runtime intensive, one can also specify to only compare the file sizes of the result files, which detects most errors.

Runtime analysis

The wall clock time of each program run is measured. The measured time interval starts right before calling the program and ends after the program has finished. The runtimes are written to a file and a bar plot is created. The runtimes are also converted to GCUPS (billion cell updates per second) which is a commonly used measure of speed for Smith-Waterman implementations. The GCUPS value is calculated by dividing the total number of computed cells (total query residues times total database residues) by the runtime. The GCUPS measure is particularly useful when comparing different alignment experiments with differing input sequence sizes.

If specified, besides the runtime also the CPU usage as measured by the /usr/bin/time command is reported. This information is used to verify that the actual CPU usage matches the theoretical CPU resources available for the calculation.

3.1.2 Performance comparison results

It is important to measure the effects of different parameters on the runtime of the different programs. This enables us to find the best performing parameter settings for each program and it shows the strengths and weaknesses of the individual programs. In the following, four experiments are performed which each show the effect of varying one parameter. If not stated otherwise, the following default settings are used for the calculations:

- **Input sequences:** Randomly selected from Swiss-Prot, sequence lengths between 20 and 1,000
- **Alignment parameters:** BLOSUM62 substitution matrix, 11/1 gap open/extension penalties, minimum reported score: 55
- **Computing resources:** 1 CPU core (AMD EPYC 7272 2.9GHz; used for SSW, SWIPE and CUDASW++ 3) and 1 GPU (PNY NVIDIA Tesla T4; used for CUDASW++ 3 and ADEPT)
- **Performance measurement:** Mean GCUPS value from three runs

3.1.2.1 Varying substitution matrix and gap penalties

The scoring system, which consists of a substitution matrix and gap penalties, has not only an effect on the resulting alignment, but also on the runtime of the computation. In Figure 3.1, four BLOSUM matrices (BLOSUM45/50/62/80), which are optimized for different evolutionary distances, are compared. For each matrix, various gap open and gap extension penalties were tested. The gap penalties tested here are the same as those available in BLAST. The speed of SSW (around 1.3 GCUPS), CUDASW++ 3 (8-9 GCUPS) and ADEPT (2-3 GCUPS) is almost constant across all substitution matrices and gap penalties, with the exception of the BLOSUM80 matrix, where all alignment tools are considerably slower. The performance of SWIPE varies much more across the different scoring parameters. Its speed is between 0.16 GCUPS (for BLOSUM80) and 8 GCUPS (for BLOSUM62). Interestingly, SWIPE seems to be optimized for the BLOSUM62 matrix and gap penalties of around 11/1, which is a commonly used scoring system and which is the default gap penalty setting in BLAST.

3.1.2.2 Varying sequence matrix dimensions

Sequence similarity searches are performed by comparing a set of query sequences to a sequence database. The number of sequences in the query and the database depend on the use case. Although in the typical case, few query sequences are compared to a

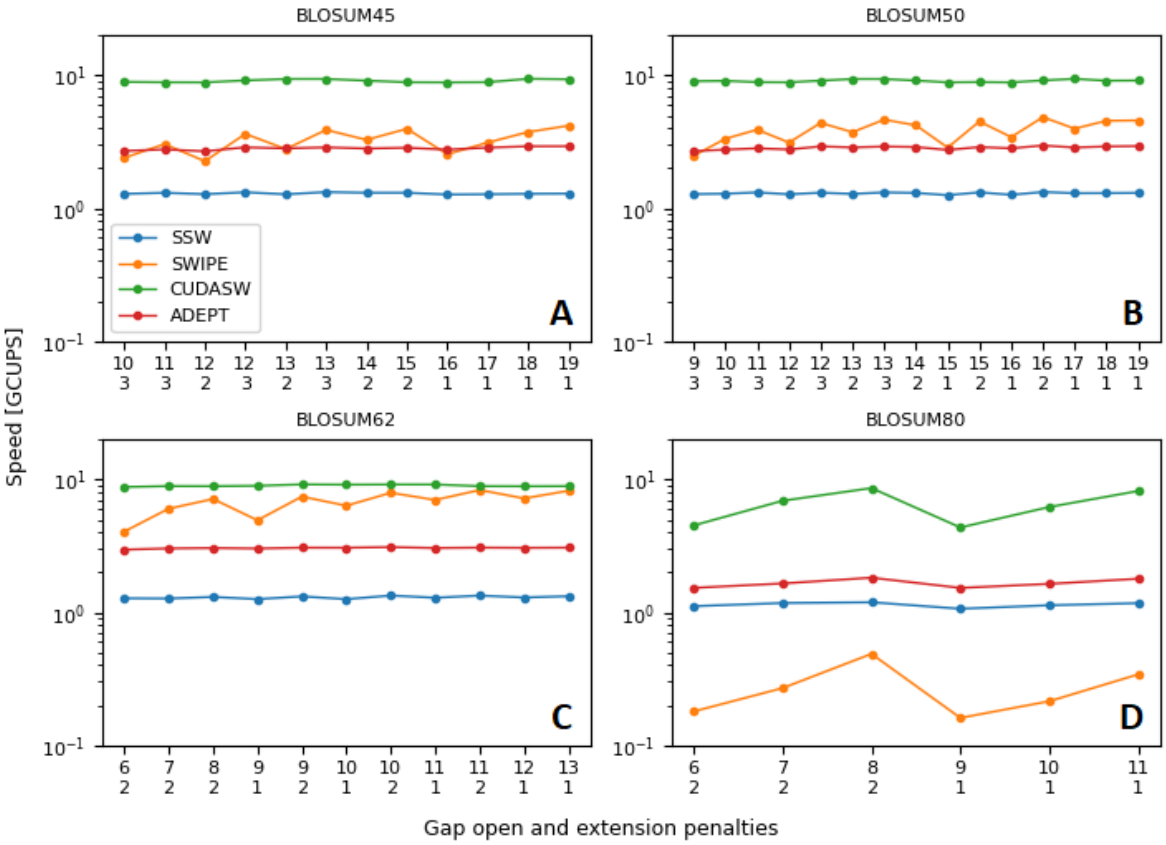

Figure 3.1: **Performance dependency on substitution matrix and gap penalties.** Each plot shows the results for a different substitution matrix: BLOSUM45 **(A)**, BLOSUM50 **(B)**, BLOSUM62 **(C)** and BLOSUM80 **(D)**. On the horizontal axis, the gap penalties are varied - the gap open penalty is written above the gap extension penalty. The speed in GCUPS is shown for SSW (blue), SWIPE (yellow), CUDASW++ 3 (green) and ADEPT (red). The query consists of 128 sequences and the database of 65,536 sequences.

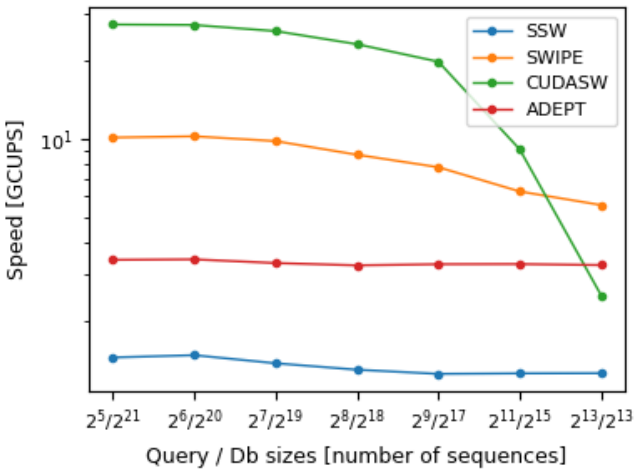

Figure 3.2: **Performance dependency on matrix dimensions.** Calculations with different query and database sizes are performed. The total workload amounts to 2^{26} alignment pairs in all calculations. The distribution of query and database sizes is varied on the horizontal axis. The dimensions of query/database sizes reach from $2^5/2^{21}$ to $2^{13}/2^{13}$.

large database, also queries and databases of the same size or even large queries and small databases can be compared. In the Smith-Waterman implementations query and database sequences are often treated differently (e.g. creation of query or score profiles which allow efficient loading of substitution values).

As can be seen in Figure 3.2, the alignment programs are affected differently by the matrix dimensions. For ADEPT, no effect can be seen. SSW is slightly faster and SWIPE is nearly twice as fast with small queries compared to a quadratic matrix. The performance of CUDASW++ 3 drastically drops as the database size gets smaller. Apparently, CUDASW++ 3 cannot parallelize efficiently and thus wastes compute resources if the database is too small.

For some alignment tasks, as for example in SIMAP, the matrix dimensions of the individual jobs can be chosen freely - in these cases the optimal matrix dimensions for the used alignment program should be chosen.

3.1.2.3 Varying sequence lengths

With increasing sequence lengths, the workload for the sequence alignment increases. In Figure 3.3, the effect of the sequence lengths on the performance of the Smith-Waterman programs is shown. The sequences are clustered into four length classes (20-150, 151-450, 451-750, 751-1,500). More than half of the Swiss-Prot sequences fall within the range of 151-450 (1st quartile: 169, median: 295, 3rd quartile: 449). In order to test the effect of the sequence length of query and database sequences separately, all 16 combinations of

query and database sequence length classes were calculated.

The performance of SWIPE is only slightly affected by the sequence lengths. The overall trend for the other alignment programs, is that the GCUPS value increases with increasing sequence lengths. Still, the runtime of an alignment between short sequences is faster than for a alignment involving longer sequences, but the cell updates per second are higher in the latter case. In other words, it is faster to fill one large alignment matrix than many smaller matrices of the same combined size, because of computing overhead besides the matrix filling. Particularly strong is the effect for SSW. By looking at the runtime (not shown in the plot), it becomes apparent that the runtime of SSW increases with increasing database sequence length, but the runtime does not increase with the query sequence length. That means, in SSW, aligning a 100-residue query sequence to a database sequence takes about as long as aligning a 1,000-residue query sequence to the same sequence. A possible explanation for this behavior of the striped approach, is that with increasing query length, the length of the stripes increases, which reduces the relative importance of the dependency between the stripes [32].

3.1.2.4 Varying number of used CPUs/GPUs

In order to speed up the computation of alignments, multi-threading can be applied, which allows to run multiple threads in parallel on multiple CPU cores and/or GPUs. In Figure 3.4, the performance of the alignment programs with different number of CPU cores and GPUs is shown. Multi-threading was achieved "manually" via the xargs command, by first splitting the query file into equally sized chunks and then aligning each of the chunks to the database via separate threads. With this approach, all programs scale almost linearly with the used compute resources - CUDASW++ 3 scales sublinearly due to its decrease in performance if the workload per thread gets too small. SWIPE contains an in-built multi-threading mode, which however scales clearly worse than the approach via xargs. Also in CUDASW++ 3 multiple CPUs and GPUs can be used for the calculations, but also here the xargs approach performs better for high numbers of CPU cores and GPUs. These results show, that it clearly pays off to perform multi-threading "manually" by splitting the workload beforehand, instead of using the built-in multi-threading modes of the alignment programs, which apparently are less efficient.

Interestingly, using the resources of the full compute node, SWIPE (using 24 CPU cores) performs better than CUDASW++ 3 (using 24 CPU cores + 6 GPUs). Apparently, the CPU computation of CUDASW++ 3, which is based on SWIPE, is clearly less efficient than SWIPE.

3.1.2.5 Conclusion

The results of the performance comparison experiments point out the strengths and weaknesses of each alignment method. SSW´s performance is rather constant across different matrix dimensions and scoring systems and it performs best if the query sequence lengths are rather long. The performance of SWIPE depends strongly on the used scoring system. CUDASW++ 3 should be only used if the database size is rather high, otherwise

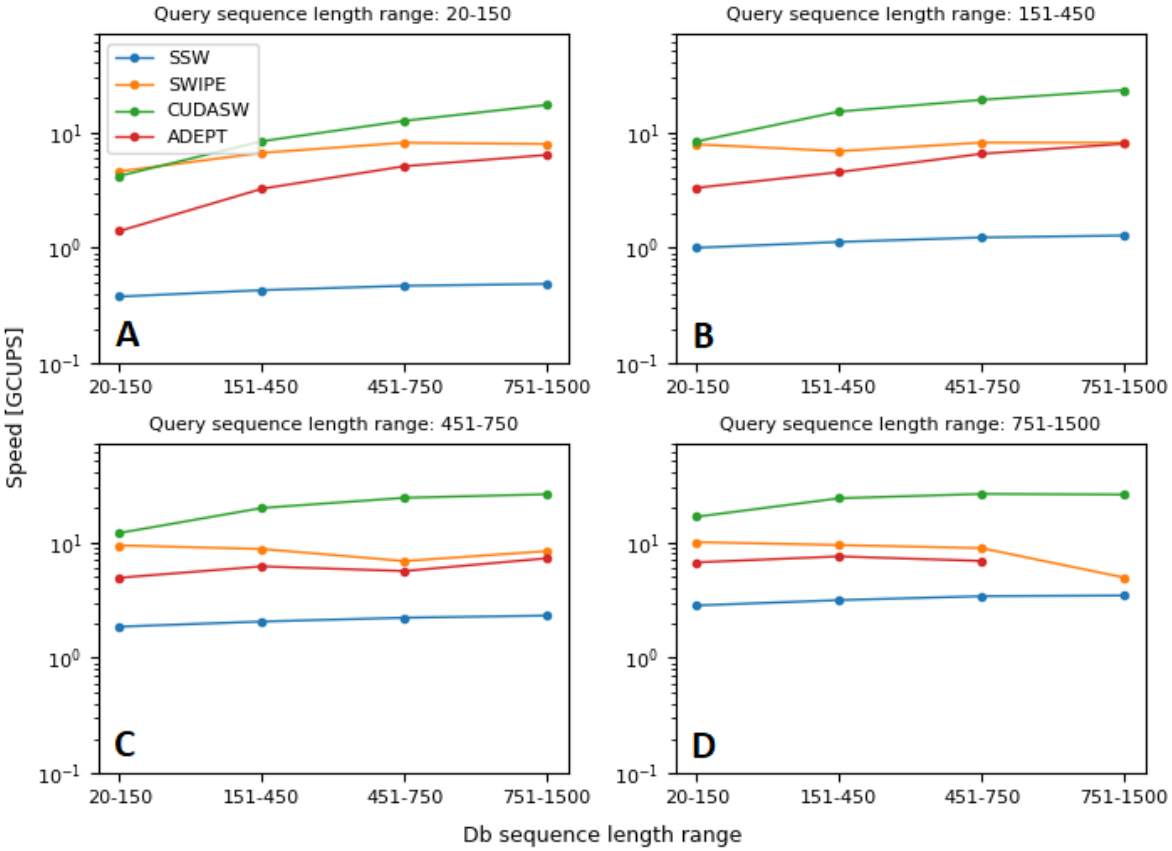

Figure 3.3: **Performance with varying sequence lengths.** Sequences were clustered into four length ranges: 20-150, 151-450, 451-750, 751-1,500. Each plot shows the result for one query length range: 20-150 **(A)**, 151-450 **(B)**, 451-750 **(C)**, 751-1,500 **(D)**. All 16 combinations of query and database sequence lengths were calculated. The query consists of 128 sequences and the database of 131,072 sequences. ADEPT failed in the setting where query and database sequence lengths are both 751-1,500.

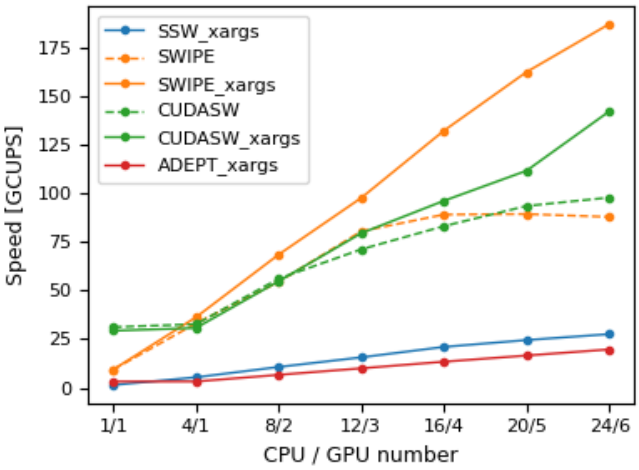

Figure 3.4: **Performance with different number of used CPUs/GPUs.** Calculations were performed with different number of CPU cores and GPUs, ranging from 1 CPU core/1 GPU up to 24 CPU cores/6 GPUs, which are the full computing resources of the used compute node. The query consists of 256 sequences and the database of 524,288 sequences. For all programs, multi-threading via xargs was performed - for SSW and SWIPE the thread number is identical to the number of used CPU cores, and for CUDASW++ 3 and ADEPT there are as many threads as used GPUs. SWIPE and CUDASW++ 3 also have multi-threading modes implemented, which are shown in the comparison. ADEPT also has a mode for using multiple GPUs, which however could not be run successfully. SSW does not provide multi-threading.

it wastes many compute resources. The performance of ADEPT is quite constant across all tested parameters.

For an overall comparison of the four alignment programs, Figure 3.2 is the most suitable, as it shows the performance with a typical scoring system and average sequence lengths and is therefore representative for most use cases. The single CPU/GPU peak performances of each program (which is achieved for all programs at small query and large database sizes) are: CUDASW++ 3 (27.5 GCUPS), SWIPE (10.2 GCUPS), ADEPT (3.4 GCUPS), SSW (1.5 GCUPS). So, the performances of the four programs are quite different. For the CPU methods, SWIPE is about seven times faster than SSW. Regarding the GPU methods, CUDASW++ 3 is about eight times faster than ADEPT. Also interesting is the comparison between the best performing CPU and GPU methods - here, CUDASW++ 3 (1 CPU core + 1 GPU) is almost three times faster than SWIPE (1 CPU core). However, considering the higher cost of GPUs compared to CPUs, SWIPE is probably preferable. Furthermore, the GPU tools require more effort to correctly configure on the compute system compared to CPU methods.

3.2 SIMAP

The SIMAP project computes sequence similarities between proteins of complete genomes. Because of the high computational workload of the alignment tasks, a faster method would be very beneficial. In the following, the runtime of the current SIMAP workflow is profiled and improvements to the different phases of the workflow are discussed.

3.2.1 Profiling the SIMAP workflow

The first task, was to profile the SIMAP workflow, in order to get an overview over the runtime performance of the different parts of the workflow. The SIMAP code was modified, so that the times of the individual alignment phases are measured. The time measurement is only performed if the SIMAP code is compiled with the "TIME_PAIRCOUNT" compile option, which reports the runtime and number of processed alignment pairs of each alignment phase. Although the additional overhead due to these measurements is small, if the information is not needed, the code should be compiled without the "TIME_PAIRCOUNT" option for optimal performance.

In Figure 3.5, the runtimes of the three alignment phases are depicted - phase 2 is further split up into three parts: sequence composition calculation, matrix adjustment, and sequence alignment. Phase 2 takes up by far the most runtime, while phase 1 and phase 3 combined amount to only around 12% of the runtime. Surprisingly, the alignment part in phase 2 takes about nine times longer than the alignment in phase 1, although much less alignment pairs are computed (the time per computed pair is about hundred times higher in phase 2 compared to phase 1). However, one has to consider that the two alignment settings are not the same - in phase 1, all pairs are computed with the same substitution matrix, while in phase 2, each pair is computed with an unique compositionally adjusted substitution matrix. SWIPE, the fastest Smith-Waterman implementation for CPU, can

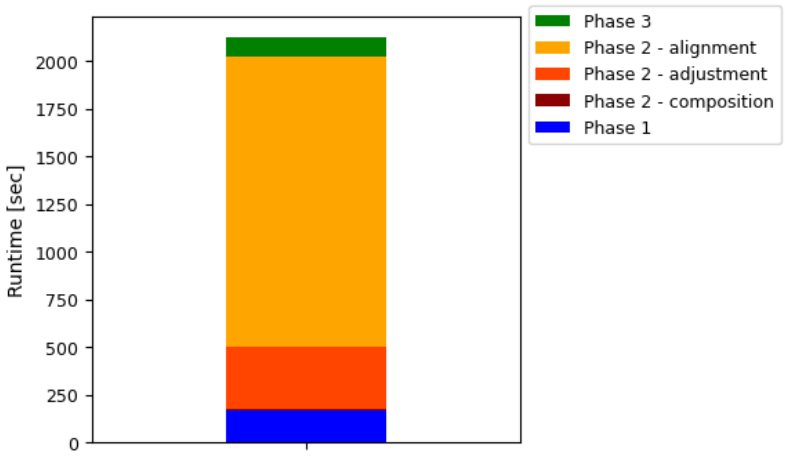

Figure 3.5: **Runtime profiling of the SIMAP workflow.** The overall runtime of SIMAP is the combination of the runtimes of the three alignment phases. A matrix of 8 million pairs from Swiss-Prot was computed using a single CPU core.

only be applied in phase 1, because it computes multiple alignment pairs in parallel, which is however only possible if all pairs are scored with the same substitution matrix. Therefore, in phase 2, the slightly slower method SSW is used, which can be applied because it computes only one alignment pair at a time.

3.2.2 SIMAP runtime improvements

In this chapter, modifications to the SIMAP workflow, with the aim to reduce the overall runtime, are presented. The modifications involve the improvement of the alignment part of phase 2, which could speed up the workflow considerably. A smaller runtime improvement could be achieved by revising the sequence composition calculation. Furthermore, the code was brought up to date, so that it makes use of the newest versions of the Smith-Waterman programs and libraries. The new version of the SIMAP workflow is available on GitHub: `https://github.com/JoTue/swipe/tree/sswlib_2022`.

3.2.2.1 Phase 2 alignment

As was already mentioned in 3.2.1, the computation of the alignments in phase 2 accounts for more than two thirds of the overall runtime. This huge runtime of the phase 2 alignment (performed by SSW) compared to the phase 1 alignment (performed by SWIPE) is surprising, as it does not fit the findings of the Smith-Waterman performance comparison from 3.1.2 - there, SSW was only about seven times slower than SWIPE.

The alignment by SSW consists of two parts: the query profile creation, and the Smith-Waterman algorithm. In alignment tasks where the same substitution matrix is used for each pair, the query profile creation has to be performed only once for each query.

However, in phase 2 of the SIMAP workflow, each pair uses a separate substitution matrix, which is why the query profile creation has to be performed for each pair. This leads to some extra overhead - however, the query profile creation takes up only around 5% of the phase 2 alignment and therefore cannot not fully explain the high runtime. SSW is built as a library, which provides an API that can be used by C++ programs. This is utilized in the SIMAP workflow, where SSW´s alignment functions are directly included and compiled together with the SIMAP code. However, the SSW library also provides an command line executable - this executable was also used in the Smith-Waterman performance comparison.

The plan was to replace the SSW alignment in phase 2 of SIMAP by the SSW executable and to compare the runtimes of both versions. This new approach also required a restructuring of the workflow. In the original SIMAP workflow it is iterated over all query sequences, and in phase 1 one query sequence is compared to all database sequences. For the remaining pairs, phase 2 and phase 3 are run directly after another for each pair, before going to the next pair. In the new version, this would require to call the SSW executable for each pair separately. However, repeatedly calling the SSW executable would imply substantial overhead - in standard alignment tasks, the executable would be called only once, taking all query and database sequences as an input. Therefore, the workflow is changed, so that the phase 2 alignment is performed for all pairs of one query sequence together, which means that the SSW executable is called only once for each query sequence. Because the SSW executable reads the input from files, the query and database sequences, and compositional adjusted substitution matrices are first written to files. Furthermore, the SSW code had to be changed slightly, so that it reads in the substitution matrices and creates query profiles separately for each pair.

As can be seen in Figure 3.6 (middle column), this new implementation which uses the SSW executable performs clearly better and reduces the overall runtime of the SIMAP workflow by half. The alignment part of phase 2 becomes about seven times faster. However, there is some additional overhead due to writing the sequences and substitution matrices to files. The new implementation brought the desired performance improvement, but the reason of the low performance of the original implementation remained unclear. After investing much time in tweaking the SSW alignment functions in the SIMAP code, which all brought no improvements, I finally found out that the error was not in the code itself, but in the compilation of the code. While the compilation of the SWIPE scripts of SIMAP was performed "correctly" via compilation into object files followed by linking into an executable, for the SSW script (ssw.cc) no object file was created explicitly. Although this produced a correctly working executable, it was poorly optimized. After improving the compilation process by also producing an object file for the SSW script, the performance improved significantly (Figure 3.6, last column). The alignment in phase 2 slightly surpassed the performance of the implementation using SSW executable and also the phase 3 alignment became faster. Furthermore, the extra workload of writing the sequences and substitution matrices to files is omitted. The overall runtime of the SIMAP workflow could be reduced to about a third of the runtime of the original version. Compared to the version using the SSW executable, the new version is not only more

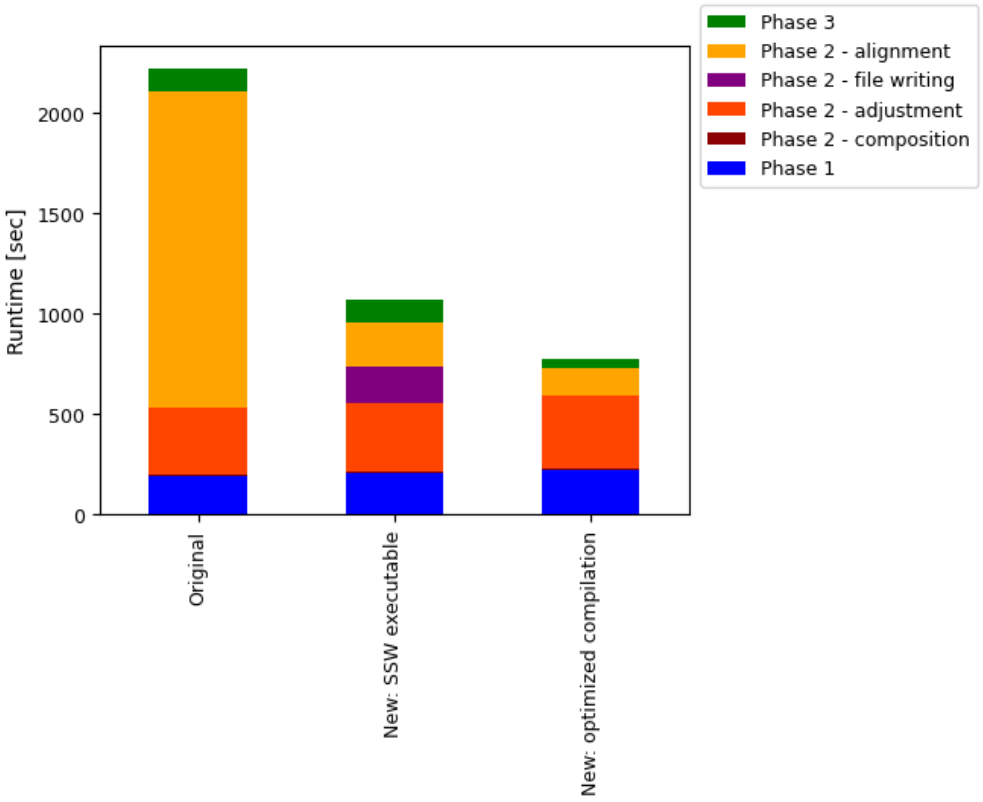

Figure 3.6: **Runtime improvement of phase 2 alignment.** The runtimes of the original SIMAP workflow, the version using the SSW executable in phase 2, and the version with an optimized compilation are compared. A matrix of 8 million pairs from Swiss-Prot was computed using a single CPU core.

performant, but also has the advantage of simplifying program usage and maintainability, since it does not require an SSW executable file. Therefore, the new SIMAP workflow implementation contains the original alignment code, but with an optimized compilation procedure.

3.2.2.2 Phase 2 sequence composition calculation

The compositional matrix adjustment function, takes the sequence compositions (amino acid frequencies of a sequence) as an input. Both, sequence composition calculation and matrix adjustment are performed by functions from the BLAST library. In the current SIMAP implementation, for each query sequence it is iterated over the database sequences which remained after phase 1, and the sequence compositions are computed. This is inefficient, because the sequence compositions of one database sequence will be computed multiple times (for each comparison to a query sequence separately).

In the new implementation, the compositions of all database sequences are computed

once at the beginning of the workflow and they are stored in a data structure. This reduces the runtime complexity from $O(N^2)$ to O(N), where N is the number of database sequences. As each sequence composition is represented by an array of 28 amino acid frequencies, the additional memory requirement of storing them is negligible and it does not increase the peak memory usage of the SIMAP workflow.

Although, due to this change the composition calculation becomes more efficient, the effect on the overall runtime is very small because the composition calculation took up less than 1% of the runtime anyway. Initially, I thought the runtime of the composition calculation is higher, because in SIMAP calculations where the input sequences are not copied to a temporary directory with high read/write speed (which is bad practice), the composition calculation accounts for about 5% of the overall runtime. Although, the improvement is quite small, the new SIMAP version contains the improved composition calculation procedure.

3.2.2.3 Updating scripts, libraries and compiler

The current SIMAP workflow was implemented in 2014, which means that various parts of the SIMAP code are outdated and should therefore be brought up-to-date. New versions of the SWIPE and SSW scripts have become available and they have been updated in the SIMAP code. Furthermore, the SIMAP code can now be compiled with the newest BLAST library (2.13.0) and the newest GNU Compiler Collection (12.2.0), which required some minor modifications in the SIMAP code. Altogether, these changes merely affect the runtime of the SIMAP workflow, but it is important to keep software up-to-date in order to assure that it compiles and runs on the latest computing systems.

3.2.2.4 Validation of new implementation

In order to validate the correctness of the new SIMAP implementation, its results are compared to the old SIMAP implementation, which is assumed to be correct. For the validation, all-against-all calculations of all Swiss-Prot sequences are performed. Such a thorough testing is important, because some errors might only occur in special sequences or alignments - possible sources of error for alignment methods are very short/long sequences, sequences with non-canonical amino acids, or alignments without positive optimal scores. In a smaller test with randomly selected sequences, possible bugs could be missed. The reported alignment attributes for a pair in the SIMAP results file are: sequence IDs, score, alignment coordinates, and percentage of similar/identical matches.

All results files of the separate SIMAP jobs were compared via the command-line tool diff, which compares two files line-by-line and reports any differences. The results of the two SIMAP versions are not completely identical. There are some pairs with slightly differing scores and also some additional hits in the new implementation. By taking a closer look at those pairs, it became apparent that all differing pairs contain sequences with (non-canonical) selenocysteine residues. The reason for the different scores lies in the different BLAST library versions the SIMAP implementations are compiled with (old version: 2.2, new version: 2.13). In SIMAP, the BLAST library is used for the

compositional matrix adjustment. Since BLAST 2.4, selenocysteine is scored as cysteine instead of as 'X' (unspecified or unknown amino acid), which explains the differing alignment scores between the two versions. All pairs, which do not contain selenocysteine, have identical scores and alignment attributes.

Besides checking the correctness of the results, the runtimes of the two versions were measured. In order to use the compute resources on the compute cluster efficiently, the workload was split up into small jobs which each run about 1-3 hours. However, the hardware and available number of CPU cores were not the same for all jobs. Nevertheless, on average the compute environment used for the calculations of both SIMAP versions was very similar, and therefore, also the runtimes are comparable.

The total runtimes were 425 hours for the old version and 160 hours for the new version. Estimating that on average 14 CPU cores were used for the calculations, the runtimes per CPU core were 5,948 hours (248 days) for the old version and 2,242 hours (93 days) for the new version. So, in this calculation the runtime of the new version was about 38% of the old version.

3.2.2.5 Runtime comparison of old and new SIMAP version

In Figure 3.7 the runtimes of the old and new SIMAP version are compared. The new version contains the improved sequence composition calculation and the optimized SSW script compilation, which is mainly responsible for the runtime improvement. In the shown example, the runtime was reduced to one third of the original version. In the new version, the runtime fractions of the different phases are: Phase 1 (26%), Phase 2 - matrix adjustment (50%), Phase 2 - alignment (18%) and Phase 3 (6%).

3.2.3 Further SIMAP modifications

Besides the implemented improvements presented in the previous chapter, two further modifications to the SIMAP workflow, with the aim to reduce the phase 2 runtime, have been developed. However, these modifications have an effect on the accuracy of the SIMAP results - they are therefore not part of the default SIMAP parameter settings, but they can be applied by setting command line options.

3.2.3.1 Skipping phase 2 for high-scoring pairs

In phase 1, scores up to 127 are computed and all pairs with scores higher than a specified threshold (default: 55) are kept for phase 2. In phase 2, all pairs with compositionally adjusted scores higher than another threshold (default: 75) are kept for phase 3. However, pairs with very high (unadjusted) scores (e.g. >100) are very likely to have adjusted scores higher than 75 - for these pairs it would be unnecessary to perform the phase 2 alignment, as they could be bypassed directly to phase 3.

A new option was added to the SIMAP workflow, which allows to set a threshold score - all pairs with phase 1 scores higher than this threshold are bypassed directly to phase 3. The lower the threshold, the more pairs skip phase 2 and the lower the overall runtime,

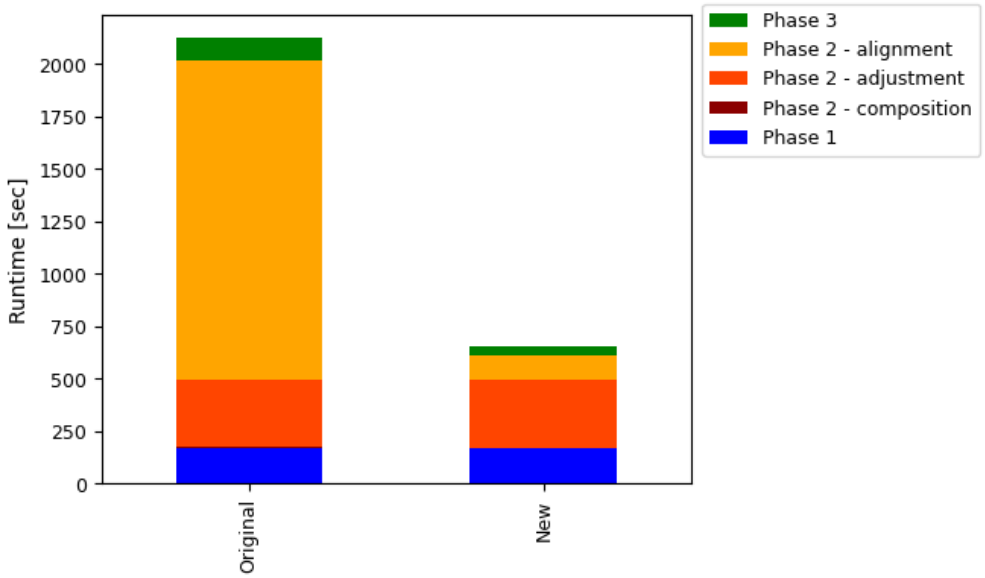

Figure 3.7: **Runtime comparison of original and new SIMAP implementation.** The phase 2 composition calculation and alignment, as well as phase 3 are faster in the new version. A matrix of 8 million pairs from Swiss-Prot was computed using a single CPU core.

but also the higher the number of "wrongly" bypassed pairs (pairs which would have lower adjusted scores than the phase 2 threshold).

In Figure 3.8A the impact of different thresholds for skipping phase 2 on the runtime is compared. Regardless of the threshold, the runtime stays constant. The reason for this lies in the score distribution of the alignment pairs (Figure 3.8B), which shows that there are very few pairs with very high scores and most pairs lie in a grey zone where its more accurate to use composition adjustment to determine if the pair is a hit. As an example, if pairs scoring at least 100 are bypassed directly to phase 3, only about 1.4% (8,672 out of 623,912 pairs) of the pairs are bypassed (Table 3.1). Furthermore, some pairs were bypassed to phase 3 which would have adjusted scores below 75, which means that the results of the SIMAP run are not identical to the default version. However, one can debate whether it makes sense to include these pairs in phase 3 - this slightly reduces the specificity, but it might prevent that some true hits are filtered out because of reduced compositionally adjusted scores.

3.2.3.2 Changing thresholds for performing matrix adjustment

The matrix adjustment of phase 2 accounts for about 50% of the total runtime and is the most runtime intensive part of the new SIMAP workflow, which is why reducing its runtime would be very beneficial. The matrix adjustment is performed via the BLAST library. However, as the authors of the matrix adjustment method stated, it is not

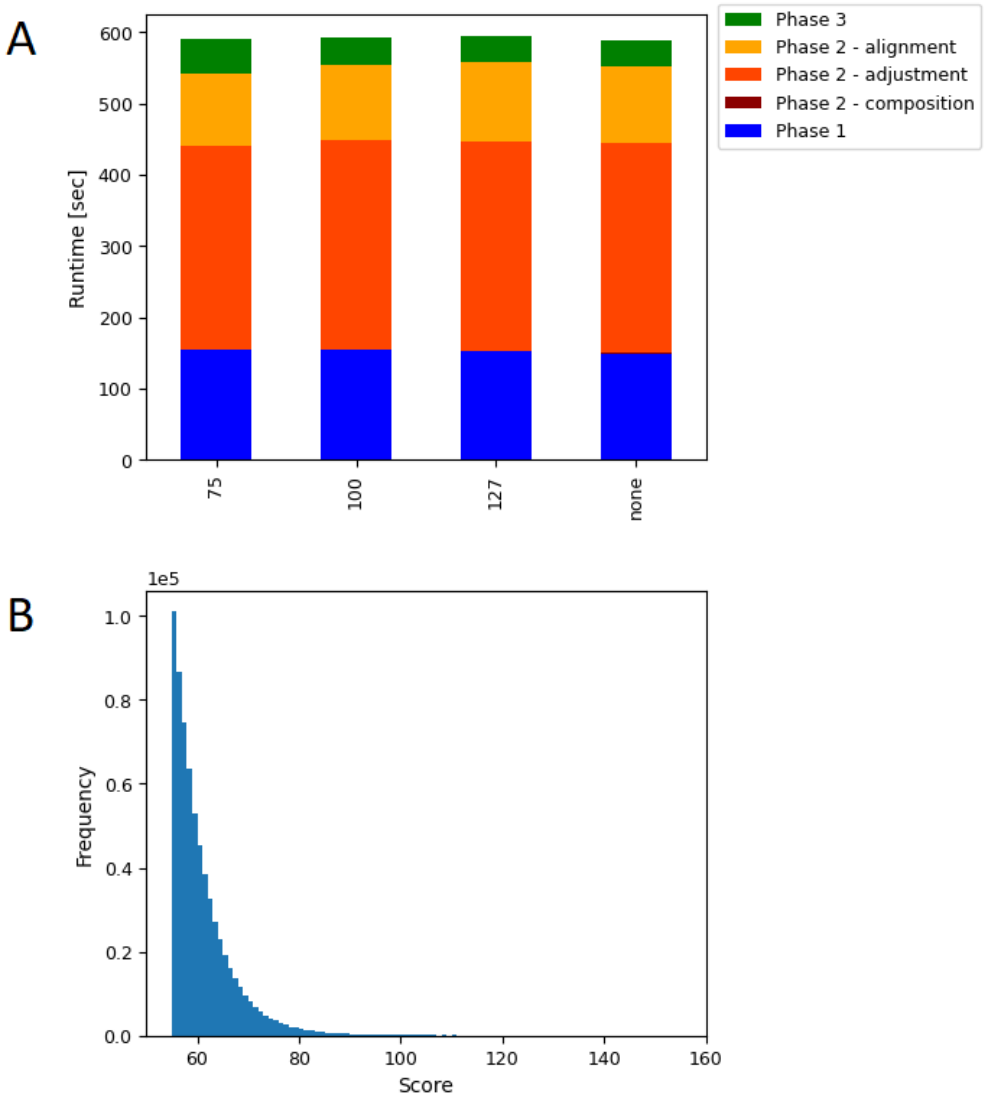

Figure 3.8: **Comparison of different score thresholds for skipping phase 2.** A matrix of 8 million pairs from Swiss-Prot was computed on a single CPU core. **(A)** The runtimes of four runs with varying thresholds (75, 100, 127, none) are plotted. **(B)** The score distribution (with parameters: standard BLOSUM50, gap open: 13, gap extension: 2) of alignment pairs with scores above 55 is shown.

Skipping threshold	Skipped pairs	Phase 3 pairs
75	27,968 (4.5%)	36,262
100	8,672 (1.4%)	26,032
127	5,819 (0.9%)	25,725
None	0 (0.0%)	25,673

Table 3.1: **Comparison of the results of different score thresholds for skipping phase 2.** A matrix of 8 million pairs from Swiss-Prot was computed and the percentage of skipped pairs out of the total 623,912 pairs that remained in phase 2 are denoted. The number of pairs which are computed in phase 3 are denoted in the last column.

advisable to perform matrix adjustment universally for all pairs, but only if certain sequence composition criteria are fulfilled. Therefore, the conditional matrix adjustment mode is the default mode in BLAST as well as in SIMAP - only if at least one of the four criteria is fulfilled, matrix adjustment is applied to the pair, otherwise, composition-based statistics are used, which is much less runtime intensive. However, in calculations involving randomly selected Swiss-Prot sequences (Figure 3.7), matrix adjustment was applied to about 88% of the pairs. Therefore, the question arises whether more stringent criteria thresholds provide similarly accurate results, while saving runtime by performing matrix adjustment for fewer alignment pairs.

Unfortunately, the interface to the BLAST library functions does not allow to set custom thresholds for the criteria. Therefore, I modified the BLAST library functions, so that they take threshold values for the criteria (length ratio, compositional distance, compositional angle) as an input. If non-default thresholds are used, the SIMAP code has to be compiled with this modified BLAST library version and with setting the compile option "COMPO_THRESHOLDS". This complicates the maintainability of the code, as the modifications to the BLAST library would have to be redone when newer BLAST versions become available.

In order to compare the accuracies of different criteria thresholds, two datasets are used: the ASTRAL40 database and the COG database. For each dataset, all-against-all alignments were performed and the alignment scores were sorted. By traversing through all scores from high to low, the number of added true positives (the sequences of the pair are from same ASTRAL-classification/COG) and false positives were identified. In Figure 3.9, the results are depicted. Unexpectedly, for both datasets the mode without matrix adjustment performed the best, followed by the conditional mode with custom (more stringent) thresholds. Furthermore, the less pairs for which matrix adjustment is done, the faster is the workflow (Table 3.2). So, according to these experiments, the compositional matrix adjustment would be obsolete. However, I want to point out that compositional matrix adjustment is well established in protein database searches (e.g. default in BLAST) and it has been tested previously that SIMAP homology searches with compositional adjustment are beneficial for eggNOG orthology assignments.

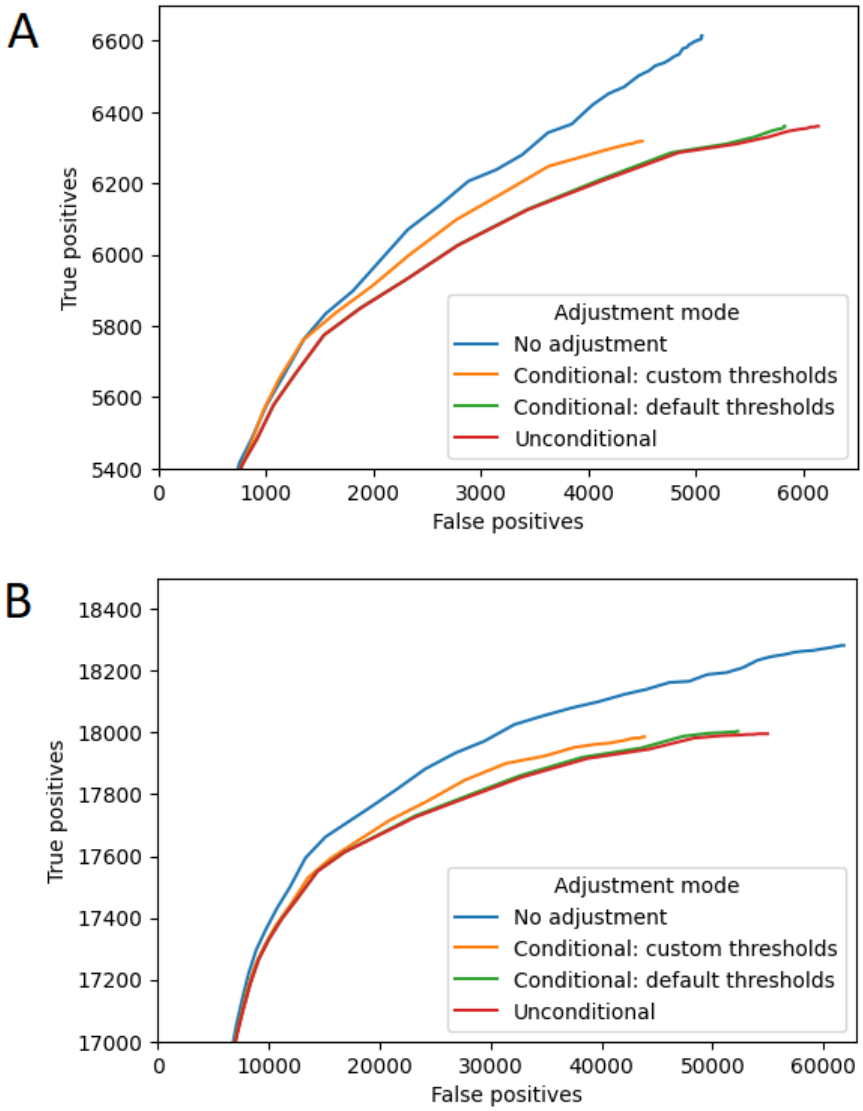

Figure 3.9: **Plotting true positive against false positive predictions for different compositional matrix adjustment modes.** All-against-all alignments were performed for the ASTRAL40 **(A)** and COG dataset **(B)**. The alignment scores were sorted in decreasing order and the alignment pairs were classified as either true or false positive. The compared compositional matrix adjustment modes are: no adjustment (blue), conditional adjustment with customized thresholds (length ratio: 1.5, compositional distance: 0.08, compositional angle: 35) (orange), conditional adjustment with default thresholds (length ratio: 3.0, compositional distance: 0.16, compositional angle: 70) (green), unconditional adjustment (red).

	ASTRAL40		COG	
Adjustment mode	Runtime (sec)	Adj. pairs (%)	Runtime (sec)	Adj. pairs (%)
No adjustment	72	0	439	0
Conditional (custom)	146	43	1119	40
Conditional (default)	203	90	1621	91
Unconditional	214	100	1698	100

Table 3.2: **Runtime and percentage of pairs for which compositional adjustment was performed.** The results for four different compositional matrix adjustment modes and two datasets (ASTRAL40 and COG dataset) are shown. As expected the runtime increases with the number of pairs for which matrix adjustment is performed. In the "no adjustment" mode, the alignment phase 2 is skipped completely.

4 Discussion

The aim of this thesis was to benchmark accelerated Smith-Waterman programs and to improve the runtime of the SIMAP workflow, which is used to perform exact pairwise sequence alignments between proteins of complete genomes. In the following, the results of this work and potential future SIMAP improvements are discussed.

4.1 Possible future SIMAP improvements

For the runtime optimization of the SIMAP workflow, its runtime profiling was crucial. This lead to the detection of an inefficient compilation of the SSW-script. The optimization of the compilation brought the main improvement of the new SIMAP version and the runtime of the workflow could be reduced to about a third, which will save compute resources for future SIMAP calculations. The runtime profiling of the workflow can also guide future efforts to improve it. The Smith-Waterman calculations account for only about half of the runtime, while the other half is taken up by the computation of composition adjusted matrices. For this thesis, my task was to remain the results of the old workflow unchanged, as it has been tested for subsequent integration into eggNOG and STRING workflows. However, in the future, parts of the SIMAP workflow such as the composition adjustment could be changed. Any changes always have to planned with the tradeoff between sensitivity and speed in mind.

4.1.1 Changing the Smith-Waterman method

One of the main ideas to improve the SIMAP workflow in the beginning of the project was to replace the Smith-Waterman programs included in SIMAP by faster methods, which was also the motivation behind the performed Smith-Waterman comparison in 3.1.

In this work, four Smith-Waterman programs for CPU (SSW and SWIPE) and GPU (CUDASW++ 3 and ADEPT) were compared under various parameters. When comparing methods for different hardware, besides the runtime also the cost of the hardware is crucial for the decision which method is the best choice for large scale analyses like for SIMAP. Although the best performing GPU method CUDASW++ 3 is faster than the best performing CPU method SWIPE on a single GPU to single CPU core comparison, SWIPE performs better on a full compute node with 24 CPU cores and 6 GPUs. In addition, the cost of the used 2x12-core CPU is clearly lower compared to the six GPUs. Another drawback of Smith-Waterman implementations for GPU is that they require more effort by the user to correctly configure, which is tedious if the compute environment changes or GPUs/GPU nodes with different specifics are used for the computations.

In SIMAP, two different alignment settings occur. In phase 1, all pairs are computed with the same standard substitution matrix, which is performed by SWIPE. In phase 2 and phase 3, unique compositionally adjusted substitution matrices are used for each pair, which is performed by SSW.

In phase 1, SWIPE could be replaced by CUDASW++ 3. CUDASW++ 3 (using 1 CPU core + 1 GPU) is about three times faster than SWIPE (using 1 CPU core). Phase 1 amounts for about 26% of the overall runtime, which means that a faster method would have a significant effect on the overall runtime. However, because of the higher costs of GPU and higher availability of CPU resources on compute clusters, SWIPE seems to be the better choice at the moment.

In phase 2 and phase 3, the fast inter-sequence parallelization methods SWIPE and CUDASW++ 3 cannot be directly applied because of the differing substitution matrices due to the compositional adjustment. In SSW, the additional overhead due to changing the substitution matrix between the alignment pairs is rather small. ADEPT is like SSW an intra-sequence parallelization method, which is about twice as fast on one GPU compared to SSW on one CPU core - nevertheless, it is not straightforward to adapt the algorithm of ADEPT to use separate substitution matrices for each pair, and such adaption might reduce the performance substantially. To my knowledge, there exist also no other GPU methods for Smith-Waterman alignments with compositionally adjusted matrices. Also regarding CPU methods, I could not find methods which are particularly designed for such a task, although in SSW such adaption is quite trivial. It would be interesting to explore, whether SWIPE could be adapted to use separate substitution matrices in a manner so that it is still faster than SSW. This would require a re-implementation of the score profile creation in SWIPE.

The two CPU methods compared in this work rely on the rather old SSE2 and SSSE3 SIMD instruction set extensions. Since then, newer instruction sets with additional instructions and wider vectors have become available (SSE4, AVX, AVX2, AVX-512). Especially, the AVX2 instruction set seems promising as it doubles the vector size compared to SSEx and many modern CPUs support AVX2. Only few Smith-Waterman methods have been developed for AVX2 - the Parasail library contains a Smith-Waterman implementation which is based on SSW, and libssa is based on SWIPE. SWIMM 2.0 can utilize AVX2 or the newer AVX-512 instruction set, which again doubles the vector size compared to AVX2. In future studies, I suggest to further explore CPU Smith-Waterman implementations based on newer instruction sets and to evaluate possibilities for integrating them into SIMAP.

4.1.2 Changing the compositional matrix adjustment method

In the following, alternatives to the compositional matrix adjustment method, which accounts for about half of the SIMAP runtime, are discussed. The currently implemented method has been developed by Altschul et al. [48, 51, 49] and is described in detail in 1.2. This approach is well established and it is used by default in all BLAST programs involving protein sequences. To my knowledge, there are no similar methods for the automatic compositional adjustment of substitution matrices. There are several other

methods, which focus on constructing matrices for specific organisms (*Plasmodium* [72, 73] and *Mollicutes* [74]) or protein classes (transmembrane proteins [75, 76, 77]) with biased compositions. However, these methods require substantial curatorial effort and are less suitable for general approaches like SIMAP, which is used for comparing protein sequences of all kind of organisms and protein classes.

A recent and quite different approach is taken by the DEDAL algorithm [78]. DEDAL (Deep Embedding and Differentiable ALignment), is a deep learning model, which aims to improve the detection and alignment accuracy of remote homologs. The Smith-Waterman alignment of DEDAL uses position-specific substitution scores and gap penalties. In the training phase, the parameterization of the scoring parameters is learned by end-to-end optimization based on a large set of known alignments. A contextual embedding of the sequences is used, where each position-specific score depends on the full sequence. A drawback of DEDAL is that its Smith-Waterman implementation is considerably slower compared to the standard (position-unspecific) Smith-Waterman methods, which makes its inclusion in SIMAP infeasible, even when low-scoring pairs are filtered out beforehand (like in phase 1 of SIMAP).

An idea for a new approach, is to learn the composition adjusted matrices (from the Altschul et al. method) from the sequence compositions via a machine learning model. In the original method the matrices are calculated via an iterative procedure (Newtonian method) which is rather runtime intensive. The goal of the new approach would be to approximate the matrix adjustment, which should speed up the procedure while ideally maintaining similar accuracy. Assuming an amino acid alphabet of length 20, the machine learning model would take a standard substitution matrix and two vectors of length 20 as an input (the sequence compositions), and would need to output a 20x20 adjusted substitution matrix. Training data could be generated by the original method. The performance and feasibility of this idea would have to be tested. The new method could also directly benefit BLAST searches, as there the same compositional adjustment method is used.

Bibliography

[1] Roland Arnold, Florian Goldenberg, Hans-Werner Mewes, and Thomas Rattei. SIMAP—the database of all-against-all protein sequence similarities and annotations with new interfaces and increased coverage. *Nucleic Acids Research*, 42(D1):D279–D284, January 2014.

[2] Jaime Huerta-Cepas, Damian Szklarczyk, Davide Heller, Ana Hernández-Plaza, Sofia K Forslund, Helen Cook, Daniel R Mende, Ivica Letunic, Thomas Rattei, Lars J Jensen, Christian von Mering, and Peer Bork. eggNOG 5.0: a hierarchical, functionally and phylogenetically annotated orthology resource based on 5090 organisms and 2502 viruses. *Nucleic Acids Research*, 47(D1):D309–D314, January 2019.

[3] Damian Szklarczyk, Annika L Gable, Katerina C Nastou, David Lyon, Rebecca Kirsch, Sampo Pyysalo, Nadezhda T Doncheva, Marc Legeay, Tao Fang, Peer Bork, Lars J Jensen, and Christian von Mering. The STRING database in 2021: customizable protein–protein networks, and functional characterization of user-uploaded gene/measurement sets. *Nucleic Acids Research*, 49(D1):D605–D612, January 2021.

[4] Stephen F. Altschul and Mihai Pop. Sequence Alignment. In Kenneth H. Rosen, Douglas R. Shier, and Wayne Goddard, editors, *Handbook of Discrete and Combinatorial Mathematics*. CRC Press/Taylor & Francis, Boca Raton (FL), 2nd edition, 2017.

[5] Valery O. Polyanovsky, Mikhail A. Roytberg, and Vladimir G. Tumanyan. Comparative analysis of the quality of a global algorithm and a local algorithm for alignment of two sequences. *Algorithms for molecular biology: AMB*, 6(1):25, October 2011.

[6] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, October 1990.

[7] S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3402, September 1997.

[8] Sergey A. Shiryev, Jason S. Papadopoulos, Alejandro A. Schäffer, and Richa Agarwala. Improved BLAST searches using longer words for protein seeding. *Bioinformatics (Oxford, England)*, 23(21):2949–2951, November 2007.

[9] S. Karlin and S. F. Altschul. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proceedings of the*

National Academy of Sciences of the United States of America, 87(6):2264–2268, March 1990.

[10] Robert C. Edgar. Search and clustering orders of magnitude faster than BLAST. *Bioinformatics (Oxford, England)*, 26(19):2460–2461, October 2010.

[11] Yongan Zhao, Haixu Tang, and Yuzhen Ye. RAPSearch2: a fast and memory-efficient protein similarity search tool for next-generation sequencing data. *Bioinformatics (Oxford, England)*, 28(1):125–126, January 2012.

[12] Yuzhen Ye, Jeong-Hyeon Choi, and Haixu Tang. RAPSearch: a fast protein similarity search tool for short reads. *BMC Bioinformatics*, 12(1):159, May 2011.

[13] Benjamin Buchfink, Chao Xie, and Daniel H. Huson. Fast and sensitive protein alignment using DIAMOND. *Nature Methods*, 12(1):59–60, January 2015.

[14] Benjamin Buchfink, Klaus Reuter, and Hajk-Georg Drost. Sensitive protein alignments at tree-of-life scale using DIAMOND. *Nature Methods*, 18(4):366–368, April 2021.

[15] Martin Steinegger and Johannes Söding. MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nature Biotechnology*, 35(11):1026–1028, November 2017.

[16] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, March 1981.

[17] S Henikoff and J G Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences of the United States of America*, 89(22):10915–10919, November 1992.

[18] M. O. Dayhoff. A model of evolutionary change in proteins. *Atlas of protein sequence and structure*, 5:suppl–3, 1987. Publisher: National Biomedical Research Foundation.

[19] O. Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162(3):705–708, December 1982.

[20] Richard Barnes. A Review of the Smith-Waterman GPU Landscape. 2020.

[21] Muaaz G. Awan, Jack Deslippe, Aydin Buluc, Oguz Selvitopi, Steven Hofmeyr, Leonid Oliker, and Katherine Yelick. ADEPT: a domain independent sequence alignment strategy for gpu architectures. *BMC Bioinformatics*, 21(1):406, December 2020.

[22] Zeyu Xia, Yingbo Cui, Ang Zhang, Tao Tang, Lin Peng, Chun Huang, Canqun Yang, and Xiangke Liao. A Review of Parallel Implementations for the Smith–Waterman Algorithm. *Interdisciplinary Sciences: Computational Life Sciences*, 14(1):1–14, March 2022.

[23] L. Dagum and R. Menon. OpenMP: an industry standard API for shared-memory programming. *IEEE Computational Science and Engineering*, 5(1):46–55, January 1998. Conference Name: IEEE Computational Science and Engineering.

[24] David W. Walker and Jack J. Dongarra. MPI: a standard message passing interface. *Supercomputer*, 12:56–68, 1996. Publisher: ASFRA BV.

[25] Yongchao Liu, Adrianto Wirawan, and Bertil Schmidt. CUDASW++ 3.0: accelerating Smith-Waterman protein database search by coupling CPU and GPU SIMD instructions. *BMC Bioinformatics*, 14(1):117, December 2013.

[26] A. Wozniak. Using video-oriented instructions to speed up sequence comparison. *Computer applications in the biosciences: CABIOS*, 13(2):145–150, April 1997.

[27] T. Rognes and E. Seeberg. Six-fold speed-up of Smith-Waterman sequence database searches using parallel processing on common microprocessors. *Bioinformatics (Oxford, England)*, 16(8):699–706, August 2000.

[28] Michael Farrar. Striped Smith-Waterman speeds database searches six times over other SIMD implementations. *Bioinformatics (Oxford, England)*, 23(2):156–161, January 2007.

[29] Enzo Rucci, Carlos García, Guillermo Botella, Armando De Giusti, Marcelo Naiouf, and Manuel Prieto-Matías. State-of-the-Art in Smith–Waterman Protein Database Search on HPC Platforms. In Ka-Chun Wong, editor, *Big Data Analytics in Genomics*, pages 197–223. Springer International Publishing, Cham, 2016.

[30] Mengyao Zhao, Wan-Ping Lee, Erik P. Garrison, and Gabor T. Marth. SSW Library: An SIMD Smith-Waterman C/C++ Library for Use in Genomic Applications. *PLoS ONE*, 8(12):e82138, December 2013.

[31] Adam Szalkowski, Christian Ledergerber, Philipp Krähenbühl, and Christophe Dessimoz. SWPS3 – fast multi-threaded vectorized Smith-Waterman for IBM Cell/B.E. and ×86/SSE2. *BMC Research Notes*, 1(1):107, October 2008.

[32] Torbjørn Rognes. Faster Smith-Waterman database searches with inter-sequence SIMD parallelisation. *BMC Bioinformatics*, 12(1):221, December 2011.

[33] Jeff Daily. Parasail: SIMD C library for global, semi-global, and local pairwise sequence alignments. *BMC Bioinformatics*, 17(1):81, February 2016.

[34] Jakob Tobias Frielingsdorf. Improving optimal sequence alignments through a SIMD-accelerated library. 2015.

[35] CUDA Toolkit Documentation v12.0 — landing 12.0 documentation.

[36] John E. Stone, David Gohara, and Guochun Shi. OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems. *Computing in Science & Engineering*, 12(3):66–73, May 2010. Conference Name: Computing in Science & Engineering.

[37] Yongchao Liu, Bertil Schmidt, and Douglas L. Maskell. CUDASW++2.0: enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions. *BMC Research Notes*, 3(1):93, April 2010.

[38] Haigang Yang, Jia Zhang, Jiabin Sun, and Le Yu. Review of advanced FPGA architectures and technologies. *Journal of Electronics (China)*, 31(5):371–393, October 2014.

[39] Mário Véstias and Horácio Neto. Trends of CPU, GPU and FPGA for high-performance computing. In *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–6, September 2014. ISSN: 1946-1488.

[40] Enzo Rucci, Carlos Garcia, Guillermo Botella, Armando E De Giusti, Marcelo Naiouf, and Manuel Prieto-Matias. OSWALD: OpenCL Smith–Waterman on Altera's FPGA for Large Protein Databases. *The International Journal of High Performance Computing Applications*, 32(3):337–350, May 2018. Publisher: SAGE Publications Ltd STM.

[41] Xia Fei, Zou Dan, Lu Lina, Man Xin, and Zhang Chunlei. FPGASW: Accelerating Large-Scale Smith–Waterman Sequence Alignment Application with Backtracking on FPGA Linear Systolic Array. *Interdisciplinary Sciences: Computational Life Sciences*, 10(1):176–188, March 2018.

[42] M.N. Isa, K. Benkrid, T. Clayton, C. Ling, and A.T. Erdogan. An FPGA-based parameterised and scalable optimal solutions for pairwise biological sequence analysis. In *2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 344–351, June 2011.

[43] Yoshiki Yamaguchi, Hung Kuen Tsoi, and Wayne Luk. FPGA-Based Smith-Waterman Algorithm: Analysis and Novel Design. In Andreas Koch, Ram Krishnamurthy, John McAllister, Roger Woods, and Tarek El-Ghazawi, editors, *Reconfigurable Computing: Architectures, Tools and Applications*, Lecture Notes in Computer Science, pages 181–192, Berlin, Heidelberg, 2011. Springer.

[44] Enzo Rucci, Carlos Garcia, Guillermo Botella, Armando De Giusti, Marcelo Naiouf, and Manuel Prieto-Matias. SWIFOLD: Smith-Waterman implementation on FPGA with OpenCL for long DNA sequences. *BMC Systems Biology*, 12(5):96, November 2018.

[45] Fabio F. de Oliveira, Leonardo A. Dias, and Marcelo A. C. Fernandes. Proposal of Smith-Waterman algorithm on FPGA to accelerate the forward and backtracking steps. *PLOS ONE*, 17(6):e0254736, June 2022. Publisher: Public Library of Science.

[46] Enzo Rucci, Carlos Garcia Sanchez, Guillermo Botella Juan, Armando De Giusti, Marcelo Naiouf, and Manuel Prieto-Matias. SWIMM 2.0: Enhanced Smith–Waterman on Intel's Multicore and Manycore Architectures Based on AVX-512 Vector Extensions. *International Journal of Parallel Programming*, 47(2):296–316, April 2019.

[47] Yongchao Liu and Bertil Schmidt. SWAPHI: Smith-waterman protein database search on Xeon Phi coprocessors. In *2014 IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors*, pages 184–185, June 2014. ISSN: 2160-052X.

[48] Yi-Kuo Yu, John C. Wootton, and Stephen F. Altschul. The compositional adjustment of amino acid substitution matrices. *Proceedings of the National Academy of Sciences*, 100(26):15688–15693, December 2003.

[49] Stephen F. Altschul, John C. Wootton, E. Michael Gertz, Richa Agarwala, Aleksandr Morgulis, Alejandro A. Schaffer, and Yi-Kuo Yu. Protein database searches using compositionally adjusted substitution matrices. *FEBS Journal*, 272(20):5101–5109, October 2005.

[50] H. Wan and J. C. Wootton. A global compositional complexity measure for biological sequences: AT-rich and GC-rich genomes encode less complex proteins. *Computers & Chemistry*, 24(1):71–94, January 2000.

[51] Y.-K. Yu and S. F. Altschul. The construction of amino acid substitution matrices for the comparison of proteins with non-standard compositions. *Bioinformatics*, 21(7):902–911, April 2005.

[52] A. A. Schäffer, L. Aravind, T. L. Madden, S. Shavirin, J. L. Spouge, Y. I. Wolf, E. V. Koonin, and S. F. Altschul. Improving the accuracy of PSI-BLAST protein database searches with composition-based statistics and other refinements. *Nucleic Acids Research*, 29(14):2994–3005, July 2001.

[53] W. M. Fitch. Distinguishing homologous from analogous proteins. *Systematic Zoology*, 19(2):99–113, June 1970.

[54] D. M. Kristensen, Y. I. Wolf, A. R. Mushegian, and E. V. Koonin. Computational methods for Gene Orthology inference. *Briefings in Bioinformatics*, 12(5):379–391, September 2011.

[55] Toni Gabaldón and Eugene V. Koonin. Functional and evolutionary implications of gene orthology. *Nature reviews. Genetics*, 14(5):360–366, May 2013.

[56] Rosa Fernández, Toni Gabaldon, and Christophe Dessimoz. Orthology: Definitions, Prediction, and Impact on Species Phylogeny Inference. page 2.4:1, 2020. Publisher: No commercial publisher | Authors open access book.

[57] Yannis Nevers, Tamsin E. M. Jones, Dushyanth Jyothi, Bethan Yates, Meritxell Ferret, Laura Portell-Silva, Laia Codo, Salvatore Cosentino, Marina Marcet-Houben, Anna Vlasova, Laetitia Poidevin, Arnaud Kress, Mark Hickman, Emma Persson, Ivana Piližota, Cristina Guijarro-Clarke, OpenEBench team the Quest for Orthologs Consortium, Wataru Iwasaki, Odile Lecompte, Erik Sonnhammer, David S. Roos, Toni Gabaldón, David Thybert, Paul D. Thomas, Yanhui Hu, David M. Emms, Elspeth Bruford, Salvador Capella-Gutierrez, Maria J. Martin, Christophe Dessimoz, and Adrian Altenhoff. The Quest for Orthologs orthology benchmark service in 2022. *Nucleic Acids Research*, 50(W1):W623–632, May 2022.

[58] R. D. Page and M. A. Charleston. From gene to organismal phylogeny: reconciled trees and the gene tree/species tree problem. *Molecular Phylogenetics and Evolution*, 7(2):231–240, April 1997.

[59] Malte Petersen, Karen Meusemann, Alexander Donath, Daniel Dowling, Shanlin Liu, Ralph S. Peters, Lars Podsiadlowski, Alexandros Vasilikopoulos, Xin Zhou, Bernhard Misof, and Oliver Niehuis. Orthograph: a versatile tool for mapping coding nucleotide sequences to clusters of orthologous genes. *BMC Bioinformatics*, 18(1):111, February 2017.

[60] R. Overbeek, M. Fonstein, M. D'Souza, G. D. Pusch, and N. Maltsev. The use of gene clusters to infer functional coupling. *Proceedings of the National Academy of Sciences of the United States of America*, 96(6):2896–2901, March 1999.

[61] Vincent Daubin and Gergely J. Szöllősi. Horizontal Gene Transfer and the History of Life. *Cold Spring Harbor Perspectives in Biology*, 8(4):a018036, April 2016.

[62] Austin L. Hughes and Robert Friedman. Differential loss of ancestral gene families as a source of genomic divergence in animals. *Proceedings. Biological Sciences*, 271 Suppl 3(Suppl 3):S107–109, February 2004.

[63] Thomas Rattei, Patrick Tischler, Stefan Götz, Marc-André Jehl, Jonathan Hoser, Roland Arnold, Ana Conesa, and Hans-Werner Mewes. SIMAP—a comprehensive database of pre-calculated protein sequence similarities, domains, annotations and clusters. *Nucleic Acids Research*, 38(suppl_1):D223–D226, January 2010.

[64] D. J. Lipman and W. R. Pearson. Rapid and sensitive protein similarity searches. *Science (New York, N.Y.)*, 227(4693):1435–1441, March 1985.

[65] Typhaine Paysan-Lafosse, Matthias Blum, Sara Chuguransky, Tiago Grego, Beatriz Lázaro Pinto, Gustavo A. Salazar, Maxwell L. Bileschi, Peer Bork, Alan Bridge, Lucy Colwell, Julian Gough, Daniel H. Haft, Ivica Letunić, Aron Marchler-Bauer, Huaiyu Mi, Darren A. Natale, Christine A. Orengo, Arun P. Pandurangan, Catherine Rivoire, Christian J. A. Sigrist, Ian Sillitoe, Narmada Thanki, Paul D. Thomas, Silvio C. E. Tosatto, Cathy H. Wu, and Alex Bateman. InterPro in 2022. *Nucleic Acids Research*, page gkac993, November 2022.

[66] J. C. Wootton and S. Federhen. Analysis of compositionally biased regions in sequence databases. *Methods in Enzymology*, 266:554–571, 1996.

[67] The UniProt Consortium. UniProt: the Universal Protein Knowledgebase in 2023. *Nucleic Acids Research*, 51(D1):D523–D531, January 2023.

[68] Heng Li. lh3/bioawk, December 2022. original-date: 2012-01-06T03:05:25Z.

[69] Naomi K. Fox, Steven E. Brenner, and John-Marc Chandonia. SCOPe: Structural Classification of Proteins—extended, integrating SCOP and ASTRAL data and classification of new structures. *Nucleic Acids Research*, 42(D1):D304–D309, January 2014.

[70] R. L. Tatusov, E. V. Koonin, and D. J. Lipman. A genomic perspective on protein families. *Science (New York, N.Y.)*, 278(5338):631–637, October 1997.

[71] Peter J. A. Cock, Tiago Antao, Jeffrey T. Chang, Brad A. Chapman, Cymon J. Cox, Andrew Dalke, Iddo Friedberg, Thomas Hamelryck, Frank Kauff, Bartek Wilczynski, and Michiel J. L. de Hoon. Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics (Oxford, England)*, 25(11):1422–1423, June 2009.

[72] Umadevi Paila, Rohini Kondam, and Akash Ranjan. Genome bias influences amino acid choices: analysis of amino acid substitution and re-compilation of substitution matrices exclusive to an AT-biased genome. *Nucleic Acids Research*, 36(21):6664–6675, December 2008.

[73] Kevin Brick and Elisabetta Pizzi. A novel series of compositionally biased substitution matrices for comparing Plasmodium proteins. *BMC Bioinformatics*, 9:236, May 2008.

[74] Claire Lemaitre, Aurélien Barré, Christine Citti, Florence Tardy, François Thiaucourt, Pascal Sirand-Pugnet, and Patricia Thébault. A novel substitution matrix fitted to the compositional bias in Mollicutes improves the prediction of homologous relationships. *BMC Bioinformatics*, 12:457, November 2011.

[75] P. C. Ng, J. G. Henikoff, and S. Henikoff. PHAT: a transmembrane-specific substitution matrix. Predicted hydrophobic and transmembrane. *Bioinformatics (Oxford, England)*, 16(9):760–766, September 2000.

[76] T. Müller, S. Rahmann, and M. Rehmsmeier. Non-symmetric score matrices and the detection of homologous transmembrane proteins. *Bioinformatics (Oxford, England)*, 17 Suppl 1:S182–189, 2001.

[77] Santiago Rios, Marta F. Fernandez, Gianluigi Caltabiano, Mercedes Campillo, Leonardo Pardo, and Angel Gonzalez. GPCRtm: An amino acid substitution matrix for the transmembrane region of class A G Protein-Coupled Receptors. *BMC Bioinformatics*, 16:206, July 2015.

[78] Felipe Llinares-López, Quentin Berthet, Mathieu Blondel, Olivier Teboul, and Jean-Philippe Vert. Deep embedding and alignment of protein sequences. *Nature Methods*, 20(1):104–111, January 2023. Number: 1 Publisher: Nature Publishing Group.