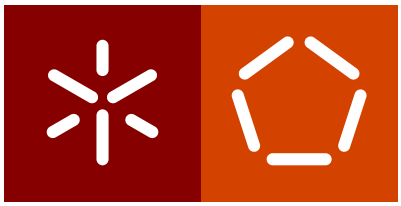
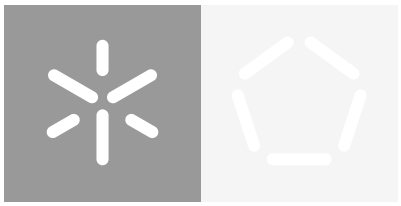


University of Minho
School of Engineering
Informatics Department

Luís Manuel Pacheco Neto

An Efficient and Accurate Framework for Large-Scale Sequences of DNA Barcodes

September 2021



University of Minho
School of Engineering
Informatics Department

Luís Manuel Pacheco Neto

An Efficient and Accurate Framework for Large-Scale Sequences of DNA Barcodes

Master dissertation
Master Degree in Integrated Master's in Informatics Engineering

Dissertation supervised by
Alberto José Proença
Eduardo Conde Sousa

September 2021

COPYRIGHT AND TERMS OF USE FOR THIRD PARTY WORK

ACKNOWLEDGEMENTS

I would like to thank Professor Alberto José Proença and Ph.D. Eduardo Conde Sousa, which lead me in the development of this thesis.

To my family for providing me the means and support to finish this milestone in my life.

To my friends for the guidance and patience that allowed me to grow throughout my entire path.

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity.

I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

ABSTRACT

DNA barcodes are short sequences of pre-defined gene regions that contain a sufficient amount of intra- and inter-species genetic information. High-throughput sequencing techniques are currently used to identify large sequences of DNA barcodes in a species genome, in a relatively short time.

Domain experts require adequate self-contained tools to accurately and efficiently process DNA barcode data in a reasonable time, taking advantage of current parallel and heterogeneous computing systems. They also expect to use these tools on different computing platforms, from laptops to high-performance servers, without requiring a broad knowledge in software engineering to develop efficient computational applications.

The main goal of this project was to develop a framework and associated user-friendly tools for domain experts to efficiently support DNA barcoding studies, providing an abstraction of the performance issues.

4SpecID is the key outcome of this work: an application software that integrates a semi-automated auditing and annotation tool for reference libraries, to ensure the quality standards of the compiled data, aiming to enable a grounded decision when identifying species from DNA barcodes. Its graphics interface aids the end user to specify the operations and it also simplifies data filtering and remote file handling.

The C++ ported version (from MATLAB) was fully tested and is more robust than the original version. Architecture features common to laptop and compute servers were exploited, namely parallel programming techniques and memory models.

The presented validation and performance results show significant improvements on execution times, not only on the sequential version, but also by using the available parallel capabilities of the underlying computing platforms.

Keywords: High-Throughput Sequencing; High Performance Computing; DNA barcoding; DNA metabarcoding

RESUMO

Os códigos de barras de ADN são pequenas sequência de regiões genéticas predefinidas que contêm uma quantidade suficiente de informação genética intra e interespécies.

Técnicas de sequenciamento de alto desempenho são usadas na identificação de grandes sequências de códigos de barras de ADN no genoma de uma espécie.

No entanto, é necessário que sejam desenvolvidas ferramentas adequadas para que os especialistas de domínio processem dados de código de barras de ADN de forma precisa e num intervalo de tempo viável, utilizando os sistemas de computação paralelos e heterogêneos que existem. Destas ferramentas é esperado que possam ser utilizadas recorrendo a diferentes plataformas de computação, de laptops a servidores de alto desempenho, sem exigir um amplo conhecimento em engenharia de software para serem utilizadas ou usadas para a criação de outras ferramentas.

O objetivo principal deste projeto é desenvolver uma estrutura que forneça uma abstração dos possíveis desafios de desempenho e permitir que especialistas no domínio tenham uma forma computacional eficiente para realizar um estudo de código de barras de DNA. Neste projecto desenvolveu-se uma ferramenta, 4SpecID, que visa permitir uma decisão fundamentada na identificação de espécies através de códigos de barras de DNA: uma auditoria semi-automática e ferramenta de anotação para bibliotecas de referência, para garantir os padrões de qualidade dos dados compilados.

Este projeto também explorou as vantagens das arquiteturas de servidores de computação e laptops mais comuns, como técnicas de programação paralela e modelos de memória. Os resultados de validação e desempenho apresentados mostram que é possível obter melhores tempos de execução utilizando as características disponíveis das plataformas subjacentes.

Palavras-chave: Codigos de barras ADN, Sequenciação de ADN, Computação de alto desempenho.

CONTENTS

1 INTRODUCTION 10
- 1.1 Challenges and Motivation 11
- 1.2 Goals 11
- 1.3 Document outline 12

2 DNA ANNOTATION AND AUDITING 13
- 2.1 DNA Barcodes 13
- 2.2 DNA Sequencing 14
  - 2.2.1 First Generation Sequencing 15
  - 2.2.2 Next-Generation Sequencing 15
- 2.3 Annotation of DNA barcodes 16
- 2.4 Auditing of DNA barcodes 17

3 EFFICIENT AND SCALABLE COMPUTING 18
- 3.1 Target platforms 19
  - 3.1.1 Portability 19
- 3.2 Balancing I/O Access and Data Processing 20
- 3.3 Scheduling Threads and Processes 21
  - 3.3.1 Shared-Memory Model 21
  - 3.3.2 Distributed-Memory Model 22
  - 3.3.3 Distributed Shared Memory Model 22

4 THE 4SPECID TOOL 23
- 4.1 Software architecture 24
- 4.2 Graphical User Interfaces 25
  - 4.2.1 4SpecID Projects 26
  - 4.2.2 Graph Visualizer 27
  - 4.2.3 Record Editor 29
  - 4.2.4 Statistical Results 30
  - 4.2.5 Data Filtering 30
- 4.3 Auditing Algorithm Implementation 32
  - 4.3.1 Datatypes 33
  - 4.3.2 File handling 34
  - 4.3.3 Data preprocessing 35
  - 4.3.4 Developing the parallel application 38
  - 4.3.5 Partition 38
  - 4.3.6 Communication 39

4.3.7 Agglomeration 39
4.3.8 Task Mapping 39
5 VALIDATION AND PERFORMANCE ANALYSIS 40
5.1 Datasets 40
5.2 Platforms 41
5.3 Performance analysis 42
5.3.1 Sequential Approach 42
5.3.2 Shared Memory Approach 43
5.3.3 Distributed Memory Approach 46
5.3.4 VTune analysis 46
6 CONCLUSION 49

LIST OF FIGURES

Figure 1	DNA barcoding scheme	13
Figure 2	4SpecID workflow	24
Figure 3	4SpecID User Interface	25
Figure 4	Graph Visualizer	28
Figure 5	Node bounding rectangle example	28
Figure 6	*Amazona barbandensis* graph component	29
Figure 7	*Amazona ochrocephala* graph component	29
Figure 8	Graphical edge removal example.	31
Figure 9	Custom filter dialog.	32
Figure 10	Auditing algorithm description.	33
Figure 11	Filter operation	36
Figure 12	Grouping operation	37
Figure 13	Compute server execution times on datasets: Canidae (top left), Aves (top right) and Culicidae (bottom).	44
Figure 14	Laptop execution times with all datasets	45
Figure 15	Server speedup	45
Figure 16	Laptop speedup	45
Figure 17	Server execution times with the hybrid memory model	46
Figure 18	Concurrency report from VTune	47
Figure 19	Thread state report from VTune	47
Figure 20	Top-down tree from VTune	48

LIST OF TABLES

Table 1	DNA Barcoding applications	14
Table 2	Namespaces description	24
Table 3	Datasets Numerical Description	40
Table 4	Target Hardware Platforms	41
Table 5	Execution times(s) in seconds of different sequential versions using the Compute Server	43

ACRONYMS

B

BOLD Barcode of Life Data. 11, 17, 32, 40

C

CLI Command-Line Interface. 24, 34, 35, 39, 42

CPU Central Processing Unit. 18–21, 47

CSV Comma Separated Values. 35

D

DNA Deoxyribonucleic Acid. 10–17

E

EDNA Environmental DNA. 20

G

GUI Graphical User Interface. 23, 25, 27, 34, 35

H

HPC High-Performance Computing. 11, 12

HTS High-Throughput Sequencing. 12

I

I/O Input/Output. 12, 18, 20, 34

M

MPI Message Passing Interface. 39

N

NGHRI National Human Genome Research Institute. 15

NGS Next-Generation Sequencing. 10, 14–16

NUMA Non Uniform Memory Access. 21

P

PCR Polymerase Chain Reaction. 15

R

RAM Random Access Memory. 21

RNA Ribonucleic Acid. 15

T

TSV Tab Separated Values. 20, 25, 33, 35

U

UI User Interfaces. 24

UMA Uniform Memory Access. 21

UX User Experience. 30

1

INTRODUCTION

Deoxyribonucleic Acid (DNA) barcoding is a method that aids domain-experts in the identification of species using DNA barcodes. This method consists of a series of steps, namely, the collection of samples, DNA sequencing and marker or DNA barcode selection.

DNA barcodes are small DNA fragments or section with intra- and inter- species information, which can be used for species identification and complement traditional methods by selecting a marker.

This method also relies on DNA sequencing, which is the process of determining the order of the nucleotides in the DNA.

Recent advances in computational sequencing techniques explores parallelism to sequence multiple DNA fragments, leading to a very fast sequencing of an entire genome. This approach is highly scalable and is referred as *Next-Generation Sequencing (NGS)*.

DNA barcoding aids domain experts to survey and perform ecological studies in a variety of areas, such as biology, and biodiversity research, and also in forensics. This method can document and characterize the species diversity of ecosystems and it can also have a better coverage to identify rare taxa within an ecosystem.

The wide range of areas of DNA barcoding applications led sequencers manufactures to use standard file formats as sequencers output. These files led the scientific community to create repositories for publishing and annotate the existing data. During the annotation process experts may add metadata, which may be the species name, the storing institution, or the cluster to which the record represents or belongs. The existing repositories are growing exponentially due to the emergence and optimization of the DNA barcoding method and the sequencing technology.

The volume of the datasets is creating a computational problem and leading to the need of tools capable of handling large amounts of data, which can be easily used by domain experts. The available datasets are in repositories that promote a friendly environment for disclosing such information to the scientific community. This approach is a joint effort among scientists, although the rules that public repositories enforce the published data is still prone to errors, and not normalized.

1.1 CHALLENGES AND MOTIVATION

Advances in DNA sequencing allows the extraction of large amounts of biological information in the form of DNA barcodes. Several research areas use or are quickly adopting DNA sequencing, annotating the obtained data, leading to the generation of massive datasets that are prone to errors, since the data annotation process is dependent on the domain expert's competence. Consequently, the volume of uncertain data that DNA sequencing generates creates a data processing challenge; moreover, domain-experts are trying to develop solutions using *High-Performance Computing (HPC)* to address this challenge.

Currently, domain-experts analyse the DNA barcodes through a pipeline where different software tools are the stages of the pipeline. There are some examples of frameworks [Jun et al. (2015), Dufresne et al. (2019)] that allow the creation and execution of processing pipelines and offer a set of DNA barcodes processing tools. The architectural shift of computing platforms and the rate of data generation create a necessity of tools with improvements in terms of efficiency and scalability. However, these improvements should not impact on the accuracy of the results and should not add unnecessary complexity.

To address the need for efficiency and scalability, the integration of existing DNA processing software tools [Zhang et al. (2013), Stamatakis (2014)], that are already optimized to run in shared and distributed memory environments, can be an option in terms of development time, accuracy and performance. Additionally, to fully exploit the target computing platforms, in this case, laptops or servers, different programming paradigms may be required, including parallel paradigms, but may add complexity to the development due to specifics of the HPC environment.

1.2 GOALS

The main goal of this work is to develop an user-friendly tool to support semi-automated auditing method to unreliable public databases, namely *Barcode of Life Data (BOLD)* and GenBank, which are susceptible to operational errors as referred by Oliveira et al. (2016). The auditing method uses the metadata to audit each species in a database. To develop an algorithm that applies the auditing method, therefore, it is required to have in consideration the resulting accuracy and the computational efficiency. The development must also consider that the end-users need not be aware of low-level optimizations and parallel computing.

To address the computational efficiency problems and to create scalable tools, strategies of parallel and distributed computing, together with an efficient use of the computing platform resources, must be studied and applied. Special consideration must also be given to the development of the scheduler, since the algorithm may require additional data when executing. The integration of existing DNA barcodes processing tools/algorithms in the tool

can be used to attain the level of expected accuracy; however, these must be in a stage where the integration would not be impaired by them. Since the goal of this tool is to offer easier usability to the domain-expert, it is necessary to create an abstraction to the user so she/he can use the tool without having to be an expert in HPC.

As stated by Ferreira et al. (2006), the creation of modules or skeletons can be beneficial in cases where the domain expert wants to use different algorithms. With this, the use of C/C++ programming languages can increase the extensibility and abstraction of the framework, since they add the possibility of using a generic programming paradigm.

1.3 DOCUMENT OUTLINE

This dissertation is structured in four chapters after this introduction, summarised below.

Chapter 2, DNA Annotation and Auditing, introduces the concept of DNA Barcodes and the *High-Throughput Sequencing (HTS)* technologies, and describes how the combination of this concept and technologies are necessary to perform DNA barcoding studies. It also describes the data, the mathematical model, the implementation of the annotation algorithm and the workflow of this annotation, focusing on the non-automated component.

Chapter 3, Efficient and Scalable Computing, summarizes three main aspects when developing the proposed software components: (i) the target platforms, namely to know which hardware is present to develop high-performance software, since it highly depends on it; (ii) the balance of *Input/Output (I/O)* accesses and the data processing; and (iii) the scheduling of threads and processes, to fully exploit different paradigms at any computing platform.

Chapter 4, the 4SpecID tool, presents the application created to address the problems presented in the preceding chapters. The application has different features which are detailed and exemplified with images during the chapter.

Chapter 5, Validation and Performance Analysis, tests the tool's functionalities, discusses the produced results and thoroughly analyses the output results obtained with three representative tomograms. Such analyses are performed both on a laptop and on a computer server, demonstrating the solution's portability.

Chapter 6 provides the final findings about the tool construction process, its strengths and not so strong characteristics. The chapter ends with suggestions for potential improvements.

2

DNA ANNOTATION AND AUDITING

An organism's classification often depends on the study case and domain expert, becoming a difficult task for non-experts and preventing taxonomy advancements. However, this branch of biology has evolved from merely classifying species by their different visible features to using DNA barcoding to aid in this challenging task.

DNA barcoding is a rapid, accurate, and automatable method that relies on DNA barcodes to aid domain-experts in delimiting individuals of a species. Its primary goal is to rejuvenate taxonomy by offering a species identification routine that can accelerate species sorting and highlight different taxa, which can describe a new species.

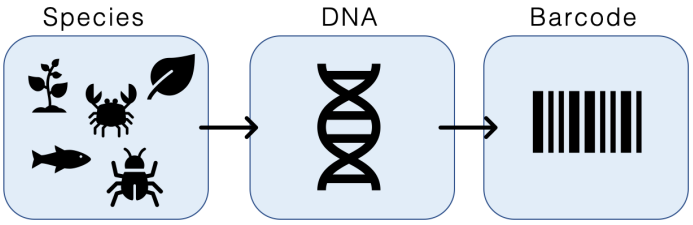


Figure 1: **DNA barcoding scheme**

This method is effective within diverse groups of animals used in different projects, and current research is trying to establish this method in other groups of organisms. With the DNA barcoding effectiveness establishment, different scientists can apply this method in several contexts, as seen in the 1. DNA barcoding is also known as DNA metabarcoding when its usage is towards identifying organisms from a sample containing more than one living beings; this sample may be soil, water, or sediment. [Hajibabaei et al. (2007), Hebert and Gregory (2005), Creer et al. (2016)]

2.1 DNA BARCODES

DNA barcoding merges the concepts of DNA barcodes (short sequences of pre-defined gene regions) and DNA sequencing to allow scientists to obtain more accurate DNA data with lower cost and time. Hebert et al. (2003) suggested DNA barcodes to simplify species identification in large taxonomic groups and speed up global biodiversity inventory. Bar-

Applications	Description
Identification of Disease Vectors	Identification of species that can cause infectious diseases to animals and humans. \citehealth
Sustaining Natural Resources	Monitorization of illegal trade of products made of natural resources.
Monitoring Water Quality	Improvement in the determination of quality and creation of better policies.
Food Safety	Differentiation of edible vegetables from poisonous plants. \citefood

Table 1: **DNA Barcoding applications**

coding DNA is a simple, low-cost, and standard diagnostic method capable of identifying organisms at any point in their life cycle, eliminating confusion and ambiguity.

Nowadays, several scientists from various research fields use DNA barcoding to identify species or operational taxonomic units from environmental samples, thus allowing the study of many characteristics of species communities. [Blaxter et al. (2005), Pavan et al. (2015), Candek and Kuntner (2015)]

However, DNA barcodes are useful if they contain a sufficient amount of intra-and inter-species genetic information. When comparing two DNA barcodes, the difference between them should be limited if both are of the same species, and if not sufficient to distinguish them, this property is known as "Barcoding Gap".[Candek and Kuntner (2015)] Therefore, choosing the gene zone or barcode is crucial because it able scientists to extract relevant information. There are already some well-established markers for animals, plants, bacteria, fungi, and protists.

2.2 DNA SEQUENCING

DNA sequencing is the method or technology to determine the order of the four nucleobases from a given DNA. This sequencing technologies has various applications, such as molecular analysis, infectious disease identification and novel genetic risk factors for cerebral vein thrombosis and cerebral research.

Initially, the Maxam-Gilbert and the chain-termination method comprised the first generation sequencing; the latter approach leads to sequencing the first genome, but with a very high cost and time. In the following years, various developments cause the development of NGS technologies, capable of attaining high throughput with lower cost and time.

The characteristics of NGS technologies propelled many DNA barcoding studies, although these studies lead to a computational challenge: the generation of an immense amount of data.

2.2.1 *First Generation Sequencing*

Sanger et al. (1977) and Maxam and Gilbert (1977) developed the first and the most common generation sequencing techniques: it efficiently scales up due to its low complexity. Sanger sequencing allowed the sequencing of the first human genome, which had $3 * 10^9$ base pairs, but it took about 15 years, the cooperation of several laboratories and with a cost over 100 million dollars. [Kchouk et al. (2017),Schadt et al. (2010)]

Despite this feat, Sanger sequencing techniques had limitations that made its applicability difficult, such as low throughput and very high cost. These limitations lead to a 70 million dollar initiative created by the *National Human Genome Research Institute (NGHRI)* that aimed to improve this technology. [Reuter et al. (2015)]

2.2.2 *Next-Generation Sequencing*

NGS technologies made a revolution in areas like genomics and genome-related diseases. Compared to Sanger sequencing, these sequencers have lower cost, higher throughput and could sequence the human genome in a matter of days [McCarthy (2010), Meyerson et al. (2010)]. The massively parallel process, which resembles the 'wash-and-scan' steps, can generate millions of DNA/*Ribonucleic Acid (RNA)* fragments [Schadt et al. (2010), Reuter et al. (2015)]. However, as a trade-off of high throughput, NGS cannot read the complete sequence of the genome. NGS requires a *Polymerase Chain Reaction (PCR)* amplification, which introduces errors in the template sequencing and amplification bias. Furthermore, as the reads length increases, there is an increase in noise and sequencing errors, which limit reads length; in contrast, as read length decreases, the assembly of the fragments starts to be a challenge due to the large amount of generated data and low connectivity. [Schadt et al. (2010), Schatz et al. (2010)]

FASTA and FASTQ File Format

From 1975, the data generated grew exponentially and led to massive datasets and led the scientific community to adopt standard formats to represent the data, the FASTA and FASTQ formats. In the context of this project and as an initial approach, the software tools should be able to read FASTA files. This file format, initially proposed by Pearson and Lipman (1988), was designed to ease the comparison and search of DNA barcodes. Programming languages, for instance Python and Perl, are notably valuable because they are scripting languages easy to understand and provide text processing libraries. Consequently, they give an easy handling of text-based files, in this concrete case, FASTA.

The FASTA format consists of two parts of information: the header followed by the respective sequence. The header offers a unique description of the sequence and may

contain additional information denoted identifiers, used to label and reference sequences in databases. Furthermore, the sequence is a string of characters representing a protein or a nucleic acid. The header starts with '>' or ';', which denotes a comment. Below is an example of a FASTA file format:

```
;LCBO - Prolactin precursor - Bovine
; a sample sequence in FASTA format
MDSKGSSQKGSRLLLLLVVSNLLLCQGVVSTPVCPNGPGNCQVSLRDLFDRAVMVSHYIHDLSS
EMFNEFDKRYAQGKGFITMALNSCHTSSLPTPEDKEQAQQTHHEVLMSLILGLLRSWNDPLYHL
VTEVRGMKGAPDAILSRAIEIEEENKRLLEGMEMIFGQVIPGAKETEPYPVWSGLPSLQTKDED
ARYSAFYNLLHCLRRDSSKIDTYLKLLNCRIIYNNNC*
```

Based on the FASTA file format, the Wellcome Trust Sanger Institute developed FASTQ, a new file format that bundles the reads and respective quality scores. FASTQ is the most common format in bioinformatics and it inherits all FASTA's format features from manufacturers, such as Illumina, the leader of the market of the sequencers. It also became the standard output of HTS. Each entry header starts with a '@', and in this format lines can be added, which begin with '+' and denote the quality scores for the sequence [Cock et al. (2010)]. Below an example of a FASTQ file format:

```
@SRR001666.1 071112_SLXA-EAS1_s_7:5:1:817:345 length=72
GGGTGATGGCCGCTGCCGATGGCGTCAAATCCCACCAAGTTACCCTTAACAACTTAAGGGTTTTCAAATAGA
+SRR001666.1 071112_SLXA-EAS1_s_7:5:1:817:345 length=72
IIIIIIIIIIIIIIIIIIIIIIIIIIIIII9IG9ICIIIIIIIIIIIIIIIIIIIDIIIIIII>IIIIII/
@SRR001666.2 071112_SLXA-EAS1_s_7:5:1:801:338 length=72
GTTCAGGGATACGACGTTTGTATTTTAAGAATCTGAAGCAGAAGTCGATGATAATACGCGTCGTTTTATCAT
+SRR001666.2 071112_SLXA-EAS1_s_7:5:1:801:338 length=72
IIIIIIIIIIIIIIIIIIIIIIIIIIIIII6IBIIIIIIIIIIIIIIIIIIIIIIGII>IIIII-I)8I
```

2.3 Annotation of DNA barcodes

DNA barcode analysis is the process of applying one or more methods to it to obtain useful information to understand its role, characteristics and arrangement.

With the development of NGS technologies the amount of DNA barcode data generated grew exponentially and lead to the creation of massive datasets, which approximated Moore's law and motivated the coordination of scientists to create public databases, and consequently to computational and statistical challenges. [Fosso et al. (2015), Montgomery et al. (2010), Reuter et al. (2015), Shendure et al. (2017)]

DNA barcodes are obtained by sequencing a genome and it is only possible to give them meaning by applying a DNA annotation method. This method consists of a clarification or statement that defines gene positions, coding regions within a genome and decides what these genes are doing.

The proposal DNA barcoding was towards the creation of reference libraries of DNA barcodes for already identified species and to perform a consistency analysis of DNA barcodes obtained by different investigation groups.

Different studies have shown that DNA barcoding is reliable. Organism groups that are already taxonomically well-studied allow studying the barcoding gap, thus supporting the development of algorithms to discover new species in organisms groups that are understudied.

Different projects allow the scientific community to publish and retrieve data, such as BOLD system, which is an informatics workbench which helps to acquire, store, analyze and publish DNA barcode records. [Ratnasingham and Hebert (2007)]

2.4 Auditing of DNA barcodes

The public databases that the scientific community uses imposes problems, such as, prone to several errors, lack of data and metadata quality. Therefore, it is essential to establish auditing methodologies that define formal guidelines and enforce data quality rules to maintain and improve the value of the compiled and new DNA barcodes.

Methods of DNA auditing that can assure the quality of reference libraries and allow the domain-expert, with or without expertise, to use them with confidence are still nonexistent. However, there is some effort towards the development of DNA auditing methods. [Ratnasingham and Hebert (2007); Oliveira et al. (2016)] These developments are useful, since there is space for improvement and necessity of tools that aid in annotation and auditing.

This tool should be flexible in computational efficiency and capable of handling tasks of editing a dataset. The first aspect is that this kind of tool will be used in laptops, but the computational power is increasing; thus, it should be able to use the resources as they increase.

The second aspect is because domain-experts often need to modify or correct the existing records when auditing them. This editing operation is simplified with a visual graphical interface, which would support textual modifications of data in records or directly editing of a visual graph with the species, clusters associated edges connecting them. In addition to this, it should be possible to edit the dataset textually directly.

Finally, the domain-experts may conduct studies that use different datasets. These are independent of each other. Therefore, tools should contain an intuitive mechanism to store different studies' data and progress. [Zhang et al. (2013), Jun et al. (2015), Zheng et al. (2018), Schloss and Westcott (2011)]

3

EFFICIENT AND SCALABLE COMPUTING

Nowadays, the volume of data generated by diverse research areas is massive, and it must be processed. Therefore, it is necessary to understand that a simple sequential algorithm is not enough. Most chip manufacturers are designing computing platforms more complex and powerful. This approach used was to simplify cores to fit multiple cores in one chip. However, it has almost reached its limit due to heat dissipation and power consumption.

In this project's context, the end-users may not have access to powerful computer platforms; therefore, all the software developed must be portable. Nevertheless, the portability of the software must not impair its efficiency and scalability. It is also necessary to test the software's efficiency and scalability, and its portability is validated on different platforms.

The development of the software must take into account that different computing platforms may not have the same resources, and an algorithm that performs well on one platform may not perform so well in another. Therefore, it is necessary to create or use efficient strategies across all platforms or at least that is good enough for most platforms. For example, to process DNA barcoding data, it is necessary to read one or more files since it is a crucial step in processing DNA barcoding and a heavy I/O task. Further, the development of this part of the software must consider that different computing platforms may not respond in the same way.

Currently, most of the computing platforms have a multi-core *Central Processing Unit (CPU)* device that can execute instructions in parallel. Also, computing platforms have a set of multi-core CPUs interconnected through a fast local area network. Current computing platform characteristics led to the creation of parallel paradigms, and their usage explicitly takes advantage of the system's available cores, CPU devices or both the multicore devices and the interconnected CPU devices. However, the use of these paradigms adds complexity to the development of the proposed software and raises several issues, such as scheduling of resources and load balancing.

To aid the development of portable software, the usage of programming language like C++ is beneficial. The choice of C++ brings enormous advantages, in support to (i) object-oriented programming through multiple inheritances (in the construction of resources); (ii) generic programming (allowing to abstract in the API the different classes of resources); and

(iii) concurrent programming (to take advantage of runtime threads and synchronization structures native to C++). Furthermore, this language, when coupled with CMake, can address the functional dimension of the portability. It also supports concurrency and parallel programming by offering standard libraries like *thread*, *async*, and *future*. Furthermore, C++ libraries like OpenMP and MPI offer support for shared and distributed memory paradigms.

3.1 TARGET PLATFORMS

In this project's context, to validate theories and perform studies is necessary to rely on computer science. This necessity occurs due to the generation of a large volume of data and large scale studies. Nowadays, domain-experts may choose from a wide range of computing platforms, and this range encompasses platforms that vary from mobile to server; thus, the number of resources may vary.

Currently, the "free lunch" is over; the performance boost from an almost certain doubling of the clock rate propelling almost all applications every two years is no longer possible since the early years of this century. Furthermore, most chip manufacturers are designing computing platforms more complex and powerful by simplifying cores and fitting much more cores per chip; this leads to the beginning of a new era: the multicore and many-core processors era. As a consequence, almost all applications did not gain performance without changes in the program. However, only parallel and concurrent computing platforms can deliver computational power to process large volumes of data.

3.1.1 *Portability*

The architectural shift and the different characteristics of computing platforms (e.g., operating systems, stack) can influence the domain expert's choice. In this case of study, one of the objectives is not to restrain domain experts' choice of a computing platform. Therefore, when developing the software, it must be considered that the target platforms may have multiple CPU devices, each with several cores. Furthermore, the development of software must be portable in two dimensions: functionality and performance.

Functional Portability

The first dimension of portable software is the functional dimension, which means that it must produce the same results on any computing platform. Therefore, the software must have an implementation that follows a standard format or does not depend on the computing platform characteristics allowing the build of an executable using a familiar tool (e.g., Make, CMake).

Performance Portability

The second dimension is the performance dimension, which is more challenging to address when developing HPC software. Most of the computing platforms have multicore CPU devices. These devices are capable of executing instructions in parallel and a complex memory hierarchy with various levels that can significantly influence the software's performance. Therefore, the developer should take into consideration these characteristics. However, the development of parallel applications often requires different languages, libraries and tools leading to the development of applications that can only achieve its peak performance on a specific computing platform.

3.2 Balancing I/O Access and Data Processing

Nowadays, with the efforts of the scientific community is now possible to perform more extensive studies. However, the current data generation of DNA barcodes is accelerating due to the application of HTS to *Environmental DNA (eDNA)* samples. Besides, processing this volume of data is becoming a big data problem.

As stated in Chapter 2, the type of storage used in this project's context is file-based, more concretely the *Tab Separated Values (TSV)* file formats, and they may have a high number of independent records. As an initial approach, one may load and parse the file into memory and then process the data. However, one of the framework's goal is to offer the domain expert flexibility in the computing platform's choice. This goal forces the developer to consider that computing platforms may not have enough memory to load the whole file. Accordingly, the initial approach requires an enhancement to efficiently read the input files.

As an enhancement for the file reading, a producer/consumer pattern may be a solution. This pattern favors the decoupling of processes that produce and consume data at different rates, and the communication of data between producers and consumers relies on data queues. The usage of this pattern allows the framework to read data as it needs and simultaneously process it. However, two issues arise when using this pattern, the usage of already existing software and the balance of producers and consumers.

To address these issues, a method that efficiently assigns work to resources is essential to take full advantage of the underlying computing platform. Therefore, a scheduler that can dynamically assign or remove producers or consumers based on pre-defined metrics is an approach. However, existing software usage may require the loading of all data in memory, thus forcing consumers to wait until the load is complete. Consequently, due to these circumstances, this scheduler may be inadequate.

Finally, the success of the framework heavily relies on the study and test of different I/O access and data processing strategies and proper balancing of processes or threads that consume and produce data. Nevertheless, the development of the framework must

encompass that the computing platform may differ. Consequently, their specifications may also change. Therefore, any approach must take this into account to not constrain the domain expert's choice.

3.3 SCHEDULING THREADS AND PROCESSES

In this context, the computing platforms may vary in the number of CPU devices and their core population. A property of these computing platforms is concurrency, in which several tasks are simultaneously executing. These tasks are in progress simultaneously and may be running on more than one computing unit. The execution of these tasks happens within a process or thread. Therefore, an adequate scheduling of threads and processes is imperative to exploit the computing platform fully.

To exploit these computing platforms, it is necessary to employ a parallel paradigm that relies on a shared memory model, distributed memory model, or a distributed shared memory model that blends the former two.

3.3.1 *Shared-Memory Model*

In the shared-memory model, typically, there is a *Random Access Memory (RAM)* that can be accessed by the computing platform's processes and threads.

Most commonly, developers use threads to develop a parallel application because threads are the basic unit of CPU utilization, and they exist within a process and share resources with other threads, such as address space, code, and data.

The shared memory model allows threads to use memory for communicating inside a program or application using primitives of synchronization, such as locks and barriers. This model can use two types of access: *Uniform Memory Access (UMA)* and *Non Uniform Memory Access (NUMA)*. The UMA the physical memory is shared by all processors uniformly; in contrast, the NUMA the memory access time depends on the distance of the memory relative to the processor, when the cache in one processor has an update, it must be reflected in the other processors.

The usage of this paradigm brings some complications: degradation of access time and data coherence. Additionally, when using computing platforms with more than one processor, the shared memory model loses performance due to the high latency to memory.

Libraries like OpenMP and C++ standard libraries may aid in developing this layer since they offer support to many issues of the shared-memory model.

3.3.2 *Distributed-Memory Model*

The distributed-memory paradigm uses processes, which are independent instances of a program in execution that have an address space and control block. In contrast with threads, its creation and context switching is more expensive and can only communicate with other processes through specific system mechanisms.

In this paradigm, it is most likely that processes are running in distinct processors. Furthermore, processors pass messages between them to exchange information. In many cases, the bottleneck in this paradigm is the network's topology, communication, and implementation of interconnection between nodes. This paradigm can take advantage of computing platforms like cluster systems due to the higher bandwidth.

In this memory model, the most common library is MPI because it offers point-to-point communication, collective communication and operations.

3.3.3 *Distributed Shared Memory Model*

The Shared-Memory and Distributed-Memory models have flaws, but it is possible to combine them and take advantage of their qualities. Nevertheless, the usage of both paradigms makes the development of software more difficult because the developer must take into account mechanisms like data placement, thread and process pinning, load balancing, and communication strategies. Further, these paradigms complement each other since it is possible to reduce the pass of messages and memory access overhead. Furthermore, the software development should use a mixture of these paradigms but allow the manual or automatic decision on how to schedule resources.

Finally, testing the software's scalability in terms of execution time and throughput with different configurations of threads and processes is crucial to design a software layer (scheduler) capable of determining, in runtime, the best configuration. Profiling is also a necessary step to improve performance since it allows identifying the platform configuration problems, the analyses of performance, and finding performance bottlenecks.

4

THE 4SPECID TOOL

The decision to create a new software tool was taken to address the needs and use existing developments described in Chapter 2. It targets this work's primary goals: to build an efficient tool capable of employing a semi-automated auditing method in datasets.

The initial efforts towards developing this tool started as a C++ program that ported an existing auditing algorithm written in Matlab [Conde-Sousa et al. (2019)] and to validate and compare the first draft of the C++ algorithm with the Matlab version.

The development of the 4SpecID tool started from the first draft to address the requisites that domain-experts have. The C++ programming language offers several features that can help the developer attain a portable software package, at a functional and performance level, giving the user control over the available resources with better execution performance than dynamically typed languages. After all, the code is type-checked before it is executed and has interfaces to use different types of resources.

To develop the desktop application, the Qt[1] framework is the one that offers the most benefits. This framework is free and open-source to create *Graphical User Interface (GUI)*s and allow the development of a GUI that can be deployed in almost every significant desktop, mobile or embedded platform, supporting different compilers.

Qt also offers more advantages, such as reliability and performance, it does not change its underlying codebase, and it is a native application that takes advantage of the platform itself. It is written in C++, which allows easy integration with the first work done in this project. The framework has SQL database access, which the developer can take advantage of to create projects. It has its thread management, allowing the developer to schedule different time-consuming operations at the same time.

Currently, the 4SpecID tool has a design based on the workflow in Figure 2. The steps consist of:

1. Creating a project where the user inserts the dataset to be analysed and, possibly, a distance matrix.

2. Auditing the data of the current project using a set of parameters.

1 https://doc.qt.io/qt.html

3. Manual correction of inconsistencies either by removing or editing records.
4. Auditing the corrected data.
5. Export the statistical results, current data, distance matrix and parameters.

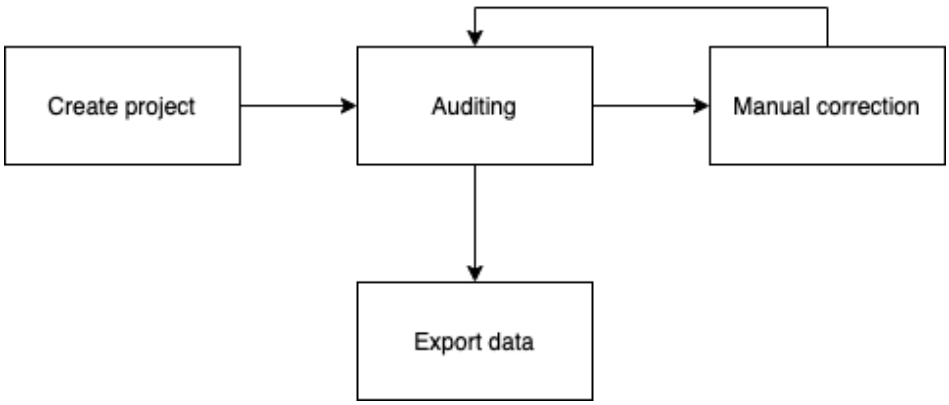


Figure 2: **4SpecID workflow**

4.1 SOFTWARE ARCHITECTURE

The C++ programming language usage was a decision, primarily taken because it is one of the best to create high-performance software packages. With this choice, there are more features to take advantage of and the language is also object-oriented and offers the possibility of creating namespaces, which allows the developer to create classes or namespaces that have specific functionalities. The design approach of the overall application is easier to debug, since it is possible to identify which component is failing, and offers a modular platform that is easy to extend and maintain.

Namespace	Functionality	Classes	Structs
ispecid::fileio	Handle read/write operations in files	None	None
ispecid::datatypes	Data structures necessary to the application	Record, Species, Dataset, DistanceMatrix	GradingParameters, Neighbour
ispecid::network	Handle data requests through network	None	None
ispecid::engine	Run the auditing algorithm	IEngine	None
utils	Preprocessing functions and miscellaneous functions	None	None

Table 2: **Namespaces description**

Table 2 describe the developed namespaces that the *Command-Line Interface (CLI)* and *User Interfaces (UI)* use to handle the data. The UI interacts with the system through these methods, allowing the complete decoupling of the business layer and presentation layer.

4.2 GRAPHICAL USER INTERFACES

The auditing of a dataset is a task that the most common computer user can handle.

To correct a dataset, the user must load the file in a text editor or *Excel*, since the data is commonly stored in TSV files. These files may contain too much irrelevant information, and to perform a correction, it is necessary to have the auditing results correcting datasets a difficult task.

To ease these tasks, the development of 4SpecID aims to give a flexible GUI application that offers to edit and visualize capabilities to the domain-expert. This GUI also offers the capability to perform the auditing algorithm and to alter its specific configurations. The GUI consists in 4 components that are presented in Figure 3 and described below.

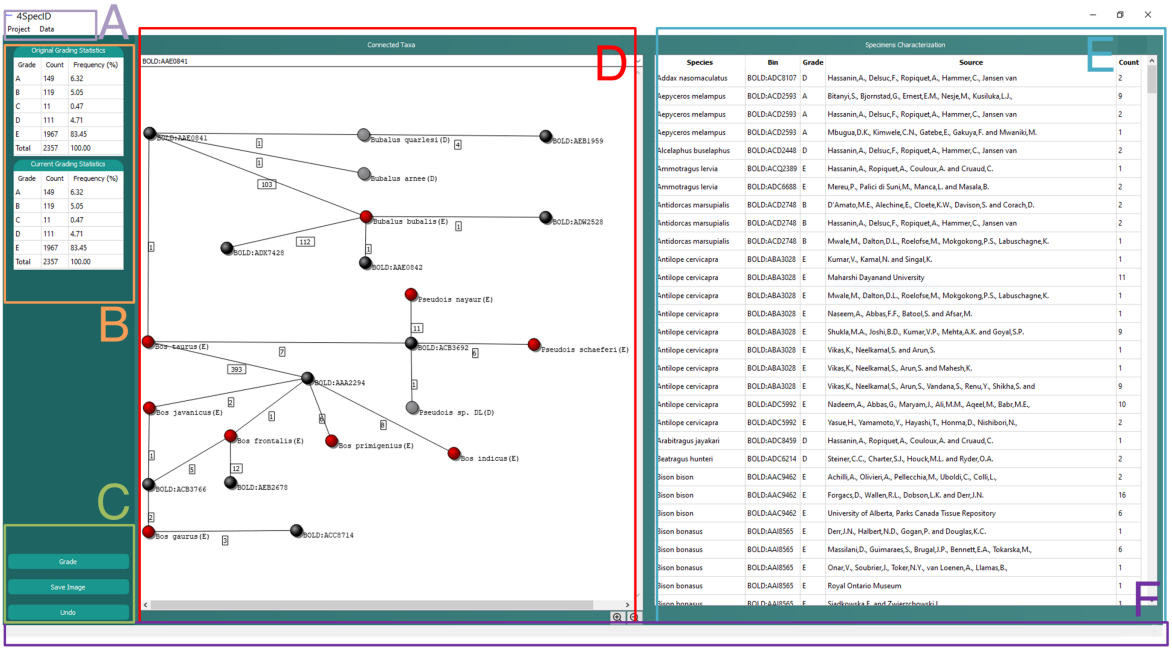


Figure 3: **4SpecID User Interface**

A - NAVIGATION BAR contains all the actions to create, load, save, and delete projects and export results, this component also handles the action of load distance matrices, conditional data filtering and auditing parameters.

B - STATISTICAL RESULTS presents the number of and the percentage of species within each grade.

C - ACTION BUTTONS shows the main actions that the GUI offers.

D - GRAPH VISUALIZER shows a partial representation of the dataset as a graph. The user can change the representation through the **RECORD EDITOR** or through a search bar. It can also delete records by deleting the edges that connect a species with a cluster.

E -RECORD EDITOR the component where the user can edit, delete or sort records.

F - STATUS VIEWER gives a textual feedback to the user when the application is doing a task.

(A) Navigation Bar; (B) Statistical Results; (C) Action Buttons; (D) Graph Visualizer; (E) Record Editor; (F) Status Viewer.

The components must have a connection that allows them to communicate between themselves to function correctly. It is necessary to have this link to propagate actions from one component to component.

4.2.1 *4SpecID Projects*

The application allows the end-user to create projects to work with different datasets. To create a project, the user must insert a name and load a dataset file; the project also contains parameters that influence the auditing algorithm, which can be modified whenever the user wants. Other tasks of handling projects are simplified and intuitive with a similar behaviour to how a text editor loads, saves and creates new files.

To allow the existence of projects, it is necessary to have a persistence mechanism within the tool. Many approaches can address this necessity; one of them is to store every project in a file; this would force the user to manage his projects. To release the user from the managing responsibility, the tool has a built-in relational database engine. The database engine's choice must consider several aspects, such as performance, availability, and infrastructure options; however, in this particular use case it is not complex enough to evaluate these characteristics in-depth. The application only needs a performant and light-weight engine that supports insert, update, and delete operations on data.

After pondering about the characteristics that the engine should have, the decision was to integrate the SQLite3 engine with 4SpecID. This engine is an appropriate choice if the developer needs to handle files as a persistency mechanism. In terms of functionality, SQLite3 is an engine that works as a library that an executable program uses. It accesses the database by reading and writing directly to the disk. In contrast, engines like MySQL or Oracle require that the application that wants to access the database must have some mechanism for interprocess communication with a separate process. This characteristic makes the SQLite3 engine easier to integrate with the tool and less prone to developer programming errors.

4.2.2 *Graph Visualizer*

Graph Visualizer is a component that aims to give a graphical interpretation of the dataset. It displays a 2D graph where the nodes are either a name of species or a cluster. The node also has a color that represents the grade in the case of a species node. Otherwise, the color is black and means that the node is a cluster. The edge that connects the nodes has a label that represents how many records have the same pair of species name and bin.

Both entities, nodes and edge, are classes were each *Node* object may have several *Edge* objects. These objects also extend to *QGraphicsItem* and *QObject* with this the *Node* and *Edge* classes inherit the communication mechanism through signals and slots from QObject and light-weight foundation for to create the graphical item from the *QGraphicsItem*. Nodes and edges have an overide method, *draw*, that implements the logic to draw their representation with Qt.

To draw the graphical items nodes and edges, the Graph Visualizer inherits from *QGraphicsScene* the capability of containing the graphical items. To visualize them is also necessary to associate a *QGraphicsView* object with a *GraphicsScene*.

To generate the representation of the data, it is necessary to consider that, in the worst case, a dataset may have thousands or millions of nodes. In this case, there are two problems to address: where to draw and how to draw the objects. With many objects to show in an GUI, it is necessary to fit them in a small space. This fitting of objects could make the representation more difficult to understand. A naive approach would be to draw the objects far from each other, but this leads to the second problem. When displaying a representation, it is necessary to draw it every frame. Consequently, the application must also draw all the graphical objects. Therefore, it exerts a very high pressure on the graphical component of the application leading to a severe decrease in its responsiveness.

In Figure 4, the visualizer only shows a graphical representation of a subset of the dataset, which is the graph component of the selected species or cluster. This choice of design has a decrease of items to represent, leading to a less dense representation, and it decreases the amount of work that the application has every frame. However, the representation has less information, and it does not guarantee that all items will fit in the scene.

Visualizer draw algorithm

The design approach aims, primarily, to reduce the load and density of the selected graph component, but this alone does not make the representation more understandable. To achieve a sufficient level of understandability it is necessary to organize where the data is in the scene. A naive approach to this would be to randomize the nodes' position inside the scene, leading to neighbour nodes being far apart.

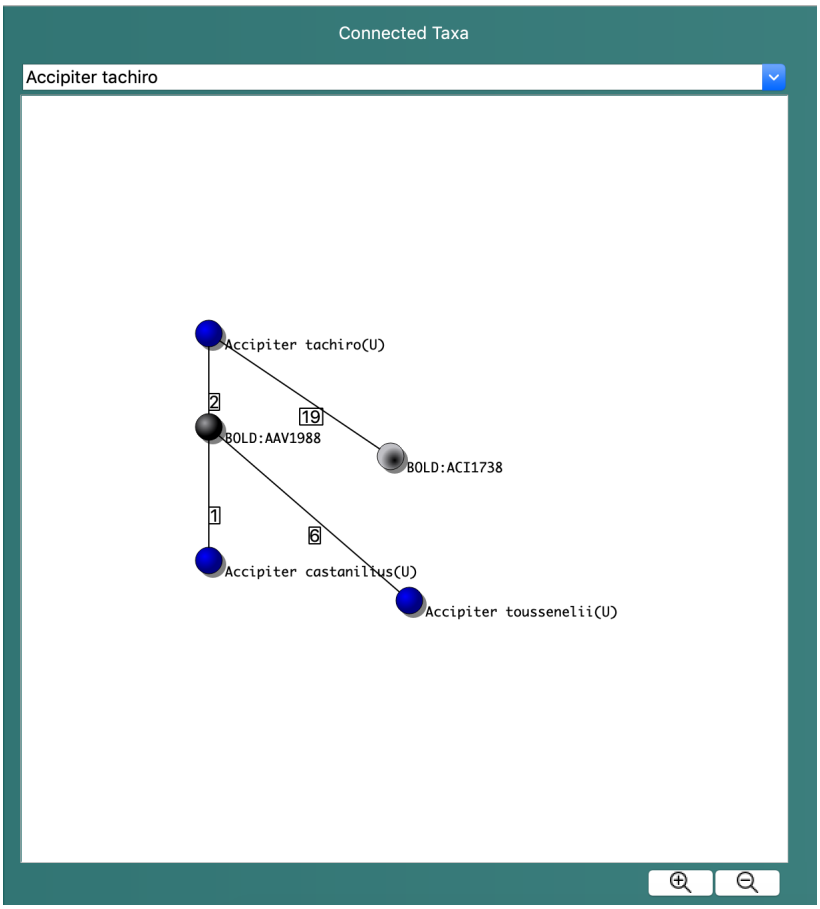


Figure 4: **Graph Visualizer**

The approach in 4SpecID is to take advantage of the graph representation of the data and display it like a tree, where the selected node is the root. The drawing algorithm starts by placing the root node in the center of the scene, then it collects its neighbours and draws them below the root and repeats the process until there are no more nodes to draw.

When drawing the graph, in the step of obtaining the neighbours of the current node, an already visible node may be included. Therefore, it is necessary to verify if a node is already visible and not draw it.

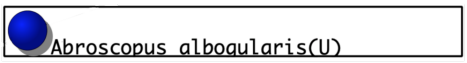


Figure 5: **Node bounding rectangle example**

Nodes on the same level are separated horizontally and vertically by fixed size to reduce the amount of overlapping of nodes, visually nodes are ellipses, but, to Qt, the node is a rectangle that contains the node itself and the space that holds the node name (Figure 5).

Figure 6 and 7 show how the application draws components with 7 nodes.

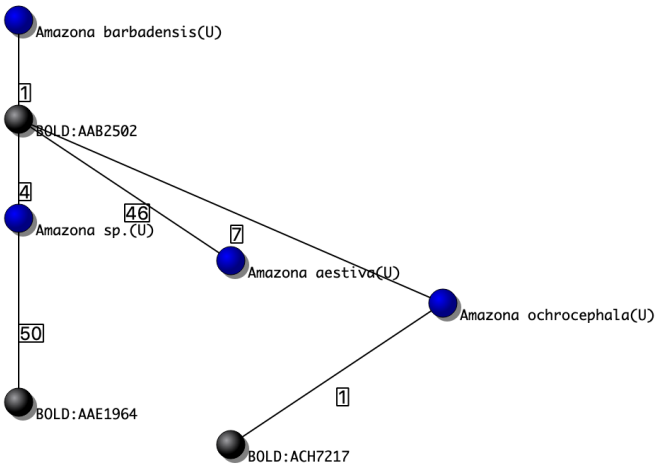


Figure 6: ***Amazona barbandensis*** **graph component**

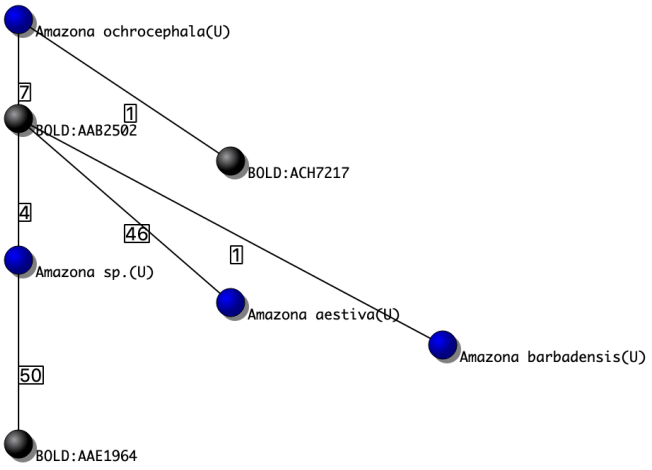


Figure 7: ***Amazona ochrocephala*** **graph component**

4.2.3 *Record Editor*

This editor shows the necessary information for the auditing and correction process. The editor lines are the result of a grouping operation applied before the load of the dataset. Each line may represent one or more entries of the initial dataset. This choice of design

allows the user to navigate and perform actions more efficiently.The information consists of triplets of a species name, bin, and storing institution. Each triplet has associated a grade and a count useful to proceed with the manual correction.

The editor allows the user to edit the dataset and guarantee that the data's edition is visible through the other components; for example, if the user edits the species' name, the graph visualizer must be updated. It also allows the deletion of triplets; the user may delete one or more triplets by selecting them. It also allows the user to change the graph visualizer's current component by clicking on a species name or bin cell.

This component's design is similar to Microsoft Excel, which is a standard application for handling data processing and visualization. Therefore, it can offer a *User Experience (UX)* that is already known to the end-users of the 4SpecID. Within the use case of this project, a two-dimensional array of items can represent the data. By creating a subclass of *QAbstractItemModel* the *RecordModel* class, which provides a standard interface for data to manipulate our data model it is possible to take advantage of the *QTableView*.

The Qt framework offers the *QTableView* class that displays items from a model. This class provides a standard approach to design flexible tables using Qt's model/view architecture. It also implements the interfaces specified by the *QAbstractItemView* class, which allows the display of data provided by models derived from the class *QAbstractItemModel*.

4.2.4 *Statistical Results*

In Figure 2, when conducting manual corrections, domain-experts must evaluate the results of the auditing algorithm. The 4SpecID tool has two result tables to aid in the evaluation task: one with the initial auditing results and one with the current auditing results. These tables present the number of species and frequencies of each grade, and this information allows the domain-experts not only to evaluate the results but to compare the progress of the corrections performed.

The tables use the same software approach as the record editor: *QTableView* and a subclass of *QAbstractItemModel* the *ResultsModel*.

4.2.5 *Data Filtering*

The possible corrections that a domain-experts can apply are editing of a record field or deleting entire records. However, this operation can be extensive since the number of cells can be very high, due to the grouping of the data in triplets. Therefore, if it is necessary to remove all the records with a species name A and a cluster B, the domain-expert would need to select several records and then delete them.

The 4SpecID tool aims to aid and improve how domain-experts handle the data with different methods to correct the data. Only having the record editor, the correction can be error-prone due to the number of records that have to be selected and deleted. To address this issue, there are two more methods that the domain-expert can use to correct the data: visual deletion through the graph visualizer and custom filter that require the user to have some prior knowledge on logical conditions.

Graph Visualizer Filtering

As stated in Subsection 4.2.2, the graph visualizer presents a subset of data as a graph. Naturally, a graph has edges and nodes, and in this use case, nodes are species or clusters, and edges represent the relation between a species and cluster. The graphical representation of this relation is a line, and it represents all the records that contain a specific cluster and species name. The tool uses the graphical representation to offer another method of correction to domain-experts. The method consists of the elimination of an edge between a species and a cluster node. This elimination has a reflection on the data; more specifically, all records that the edge represents are removed from the dataset.

The user can perform this action by clicking on the edge and selecting the *Remove edge* option as Figure 8 shows.

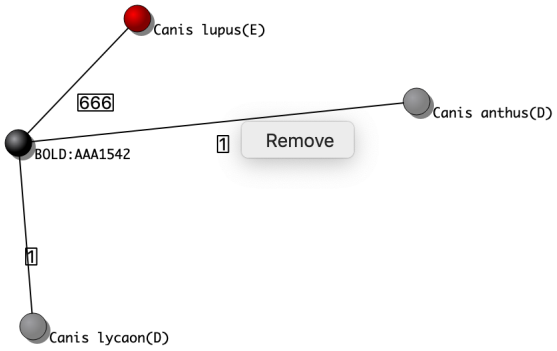


Figure 8: **Graphical edge removal example.**

To implement this correction method, the existing subclasses of *QGraphicsItem*: node and edge are used since the edge object stores two nodes, and each node object stores the species name; it is possible to filter the dataset using this information. Since the graph visualizer is a subclass of *GraphicsScene* class, it can handle mouse events that are used to create the pop-up menu and highlight when hovering the edge.

Custom Filters

The record editor and the edge removal methods increase the tool's usability, but it also provides a more sophisticated approach to filter data. Using the filter concept, the tool provides a prompt where the user can create a logical proposition to filter the data. These propositions may have different prepositions that are composed of a conjunction or disjunction. The tool allows the user to apply the filter to keep or remove the data points that satisfy the preposition.

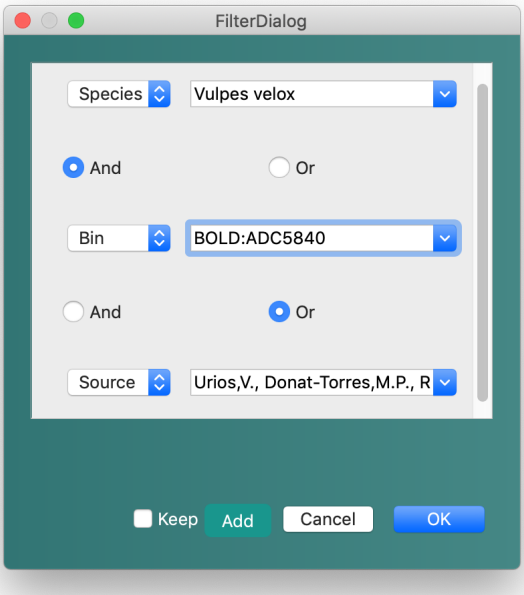


Figure 9: **Custom filter dialog.**

The application uses the *QDialog* class to implement this functionality. This dialog has autocompleted combo boxes that only have items present in the dataset to prevent human errors. After graphically creating the proposition, the application creates a function representing the predicate used to filter the data points.

4.3 Auditing Algorithm Implementation

Reference mtDNA libraries, such as BOLD [Ratnasingham and Hebert (2007)] or GenBank [Coordinators (2017)], are indispensable tools in the barcoding approach, allowing the collection and organization of a considerable amount of DNA sequences (almost 5.5M at BOLD) and associated metadata. An endeavor of such magnitude requires a global synthesis of data, including regular auditing for consistency evaluation of DNA barcodes

and corresponding annotation [Oliveira et al. (2016), Conde-Sousa et al. (2019)]. Indeed, this type of approach's success depends on the representativeness of databases, which is only reachable by the contribution of the general scientific community. This collaborative effort's drawback is that public databases are susceptible to inaccuracies, such as poor quality of the molecular data or imprecise/conflicting taxonomic identifications. To address this task, an auditing algorithm for (global and local) reference libraries must ensure the compiled data's quality standards [Conde-Sousa et al. (2019)].

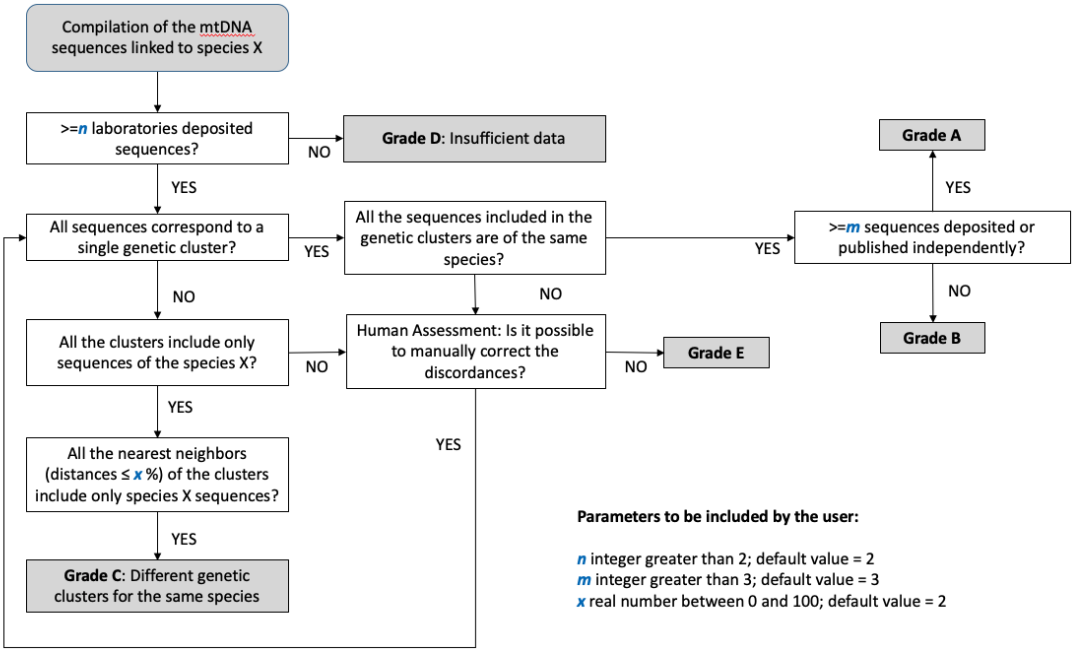


Figure 10: **Auditing algorithm description.**

The first version of the auditing algorithm is in Matlab, but the algorithm may be executed on databases with up to millions of entries, making parallelism and efficiency unavoidable topics. To address these topics, the 4SpecID tool integrates the implementation of this auditing algorithm in C++.

The C++ implementation reads the data from a TSV and then converts each line in *Record* objects. After loading all the data in memory, a filter is applied to all the records, removing all the records that do not have a species name, cluster or storing institution; the remaining records are grouped by species name. Finally, the audit algorithm is applied to the grouped records. The subsequent sections describe the steps of its development.

4.3.1 *Datatypes*

To encode the data necessary necessary the correct execution of the auditing algorithm there are three different data structures the *Record*, *Species*, *GradingParameters*, and *Neighbour*. The

first two are classes since they combine variables, functions, and data structures; the last one is a struct because it is only necessary to read and write from it. These data structures are used in both CLI and GUI applications.

Record

This class encodes every record that had been read from the input file. Since there is information that is not relevant for the algorithm, this class only stores the species name, cluster, and institution storing. These attributes type is *std::string*.

Species

This class represents a species along with all clusters and storing institutions. In this context a *Species* is an aggregation of *Record* objects and when aggregating the *Record* objects the clusters and storing institutions are stored in *std::unordered_set* to guarantee that the *Species* object only contains unique values. It also has a grade attribute, whose type is *std::string*, that will be assigned after the auditing algorithm execution.

Neighbour

The algorithm also requires a comparison of neighbour clusters. Therefore, this data structures encapsulates two clusters as a *std::string* and their distance which is a *double*.

GradingParameters

This structure stores the minimum number of sources and records and the maximum distance between two clusters. It encodes the necessary data for the audit algorithm; if the user does not assign any values, the application will use, as default values, 2 for the number of sources, 3 for the number of records and 2.0 for the distance.

4.3.2 *File handling*

The first step of this tool is to read the dataset to memory. It is one of the most crucial operations since it is an operation that requires the computer to read from a file that is on disk.

The 4SpecID tool has two applications: one that uses the command-line and one that is a full-fledged desktop application. Both tools can audit data, but they differ in some aspects, and one of them is how they handle the I/O of files.

The command-line application only requires that the dataset be loaded to the program memory, but the desktop application requires that it is necessary to insert into the database after reading the file.

CLI application

The CLI application is a lightweight version of the GUI application; thus, handling files is simpler. The application uses the *csv-parser* library to read the file to memory. The *fileio* namespace, offers different functions to deal with TSV and *Comma Separated Values (CSV)* files; these functions design provide the user with an interface that abstract all the underlying complexity, such as resource management.

The namespace functions are templates to give the user flexibility. For instance, the user may not want to directly convert the records in the file to *Record* objects, which is the default behavior when reading a file. To address this, the function may receive a *InMapper*; these objects apply a transformation to the lines that are read, allowing the user to parse and transform the data to its own use case.

GUI application

The GUI application uses the QT framework to handle the input files. The framework already provides libraries to read, parse and insert the records in the database.

4.3.3 *Data preprocessing*

After having all the data in memory, it must pass through a data preprocessing to ensure that the data that is going to be audited is correct. This phase is necessary to remove inaccuracies, poor data quality and data inconsistencies that may be present in the data.

The first step of this stage is to assess data quality; in this context, if the necessary information does not exist in the record, it will be filtered, thus, it won't be considered as a data point. The second step is the data grouping, a dataset may contain a very high number of records, and they may not be unique; this step creates *Species* objects by aggregating *Record* objects. The aggregation stores the *Species* objects in a *std::unordered_map* where the key is the species name and the value the corresponding *Species* object.

Filter

Data filtering is a process to detect corrupt records in the dataset. It identifies the incomplete records of the data and deletes them. This process is also known as data cleansing, and it may be performed with different approaches; in this project context, it was developed a function that performs this task.

Our use case considers that any record that does not have a species name, cluster or storing institution is incorrect and incorrect data can lead to false results and induce the domain-experts in wrong corrections on the data. For instance, a dataset contains a species that has only been studied by one institution (consider the institution name A); however,

when inserting the data, an error occurs, and one of the records is inserted without the storing institution. When it is performed the aggregation of records, this species will have two storing institutions since they are stored as a *std::string*, the "A" and "" institutions. The current example will not grade the species with D and will produce incorrect results.

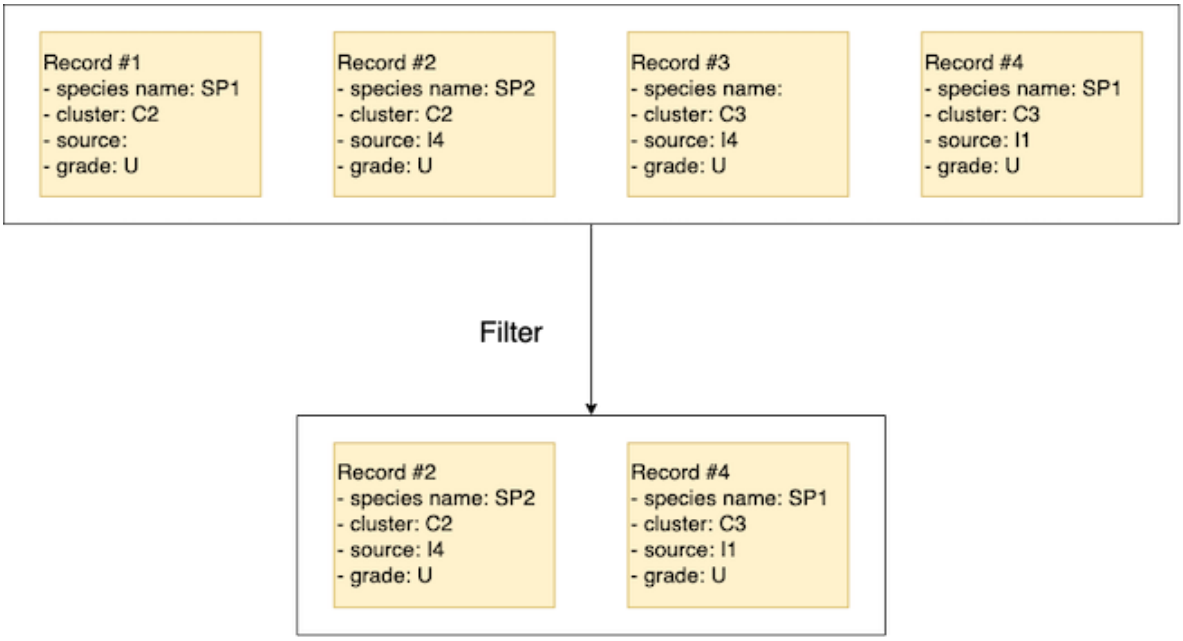


Figure 11: **Filter operation**

Generally, when a filter is applied to a dataset all data points that do not verify a certain condition are removed; a possible implementation is shown in the example below:

```
for(int i = 0; i < N;){
    if(!cond(data[i]))
        remove_item(data, i);
    else
        i++;
}
```

However, when removing one item in the array all remaining items must shift one position left. Instead of accessing each item once, it is possibly to access i times, where i is the position in the array. To optimize the filter operation the tool uses a secondary variable to store the result and then returns it; the condition is also different: it tests if the data point verifies the condition then the data point is appended to the result variable, as shown below.

```
for(int i = 0; i < N; i++){
    if(cond(data[i]))
        append_item(result, data[i]);
}
```

Group

Datasets have a high number of records, and from these, it is only necessary to process the information that refers to the species name, cluster, or storing institution. Different data points may be similar or even equal. Therefore, the dataset may have duplicates, and it is necessary to condense this information to have correct results.

To the auditing task, it is relevant the unique species and respective clusters and storing institutions. The number of records that belong to the same species is also important when choosing grade A or B.

This preprocessing step creates an *std::unordered_map* that maps the species name to *Species* object. These objects are the result of iterating a collection of records. For each record if the respective *Species* do not exist, it is created and then inserted in the *std::unordered_map* otherwise the object is fetched, updated, and then inserted in the *std::unordered_map*.

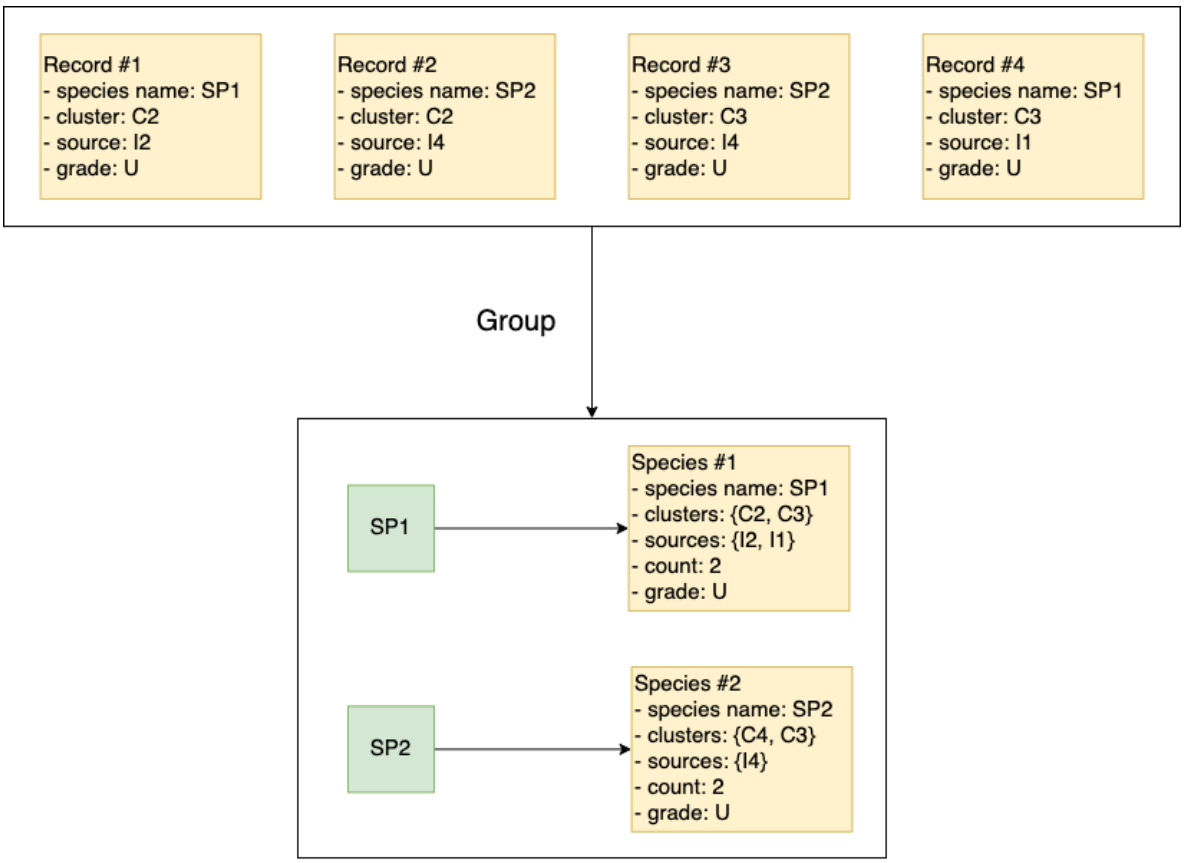


Figure 12: **Grouping operation**

Finally, this process allows the minimisation of computation in the auditing algorithm by using the *Species* object to read the necessary information for its execution.

4.3.4 *Developing the parallel application*

One of the goals of this project is to develop an algorithm that efficiently uses the underlying platforms' resources. The target platforms are servers and desktops, and to have efficient use of resources, it is necessary to study the shared and distributed memory approaches.

Both approaches are parallel applications, and their development uses the same methodology, which consists of creating partitions of the data to be processed in tasks, identifying opportunities for execution in parallel. The next step consists of determining the necessary communication between the various tasks, defining the appropriate data structures and algorithms. Then it is essential to condense the tasks and data structures defined in the first two phases, to improve performance or reduce development costs. Finally, it is necessary to map the tasks in the computing nodes maximizing the use of CPU devices and minimizing communication costs.

4.3.5 *Partition*

This step aims to decompose the data or phases of an algorithm into fine-grained tasks to identify parallel opportunities. The decomposition has two possible classifications: the functional decomposition and the data domain decomposition.

In this case of study, the data domain is preprocessed, a map where the keys are strings, and the values are species. The audit of a species does not depend on any other species. Therefore, the data domain decomposition creates many tasks equal to the number of species.

The algorithm is a decision tree algorithm and does not have phases that can be executed in parallel. Therefore, it is not possible to apply a functional decomposition to it. However, there is a step where it is necessary to verify if all neighbours of a species are at less than a predefined distance, which involves performing data requests over the network. The number of requests varies with the number of clusters that a species has. Therefore, it is necessary to adjust the decomposition to cope with the variable load of each species.

To address the issue of different workloads of each task, the algorithm suffered some modifications. The first modification is that if the neighbour data is not available, new tasks are scheduled to fetch the necessary data, and the second modification is that the algorithm is divided into three sequential phases. In the 1st phase, the auditing algorithm tries to grade all species. In the 2nd phase, the algorithm fetches all the necessary data. Finally, in the 3rd phase, the species that did not have enough data in the first phase are audited.

4.3.6 *Communication*

The development of parallel applications may have as a target a cluster of computing nodes. Frequently, the application has one or more processes in each node, and they may have to communicate through the network. Therefore, it is necessary to take into account how this will affect the performance of the algorithm.

The communication has several characteristics, such as locality, synchronous/asynchronous, scheduling, and structure. Evaluating communication characteristics is also necessary to know if the tasks cannot execute independently.

The possible tasks that the auditing algorithm issues are always independent. Therefore, it is unnecessary to have any communication when executing the algorithm, aside from the initial splitting of data and retrieval of results. At the begining of the execution it is not possible to know each task's load and the tasks themselves, the division of initial auditing tasks is static, and each process receives the same number.

Finally, the 4SpecID has two CLI implementations and uses different memory models, shared and hybrid, on each, and to implement the latter it the *Message Passing Interface (MPI)* library is used since it is a structured and scalable message-passing standard designed to run on a wide range of parallel computing architectures by a community of researchers from academia and industry.

4.3.7 *Agglomeration*

After identifying the tasks and necessary communication, it may be possible to improve the algorithm performance by agglomerating workloads or messages.

The agglomeration at the communication level is the aggregation of several messages into one, thus reducing communication costs. However, since the processes do not need to communicate, it is not possible to agglomerate any communication.

Finally, the division of the auditing algorithm's workload is static, as stated before, and it is not necessary to agglomerate tasks.

4.3.8 *Task Mapping*

The final step of developing a parallel application is defining where each task will run to minimize the time of execution. Since each task is independent and similar in workloads, and the auditing algorithm's iterative nature, the mapping of the tasks is regular and static.

5

VALIDATION AND PERFORMANCE ANALYSIS

To validate that the 4SpecID tool produces correct and acceptable results three different datasets were used. These datasets were obtained in the BOLD platform and are publicly available. Therefore, the developed tool is also open-source, allowing the reproducibility of the results and validations of this work.

The tune and performance analysis measured the execution times of the auditing procedure and were performed on two different platform environments: a laptop and a dual-socket compute server. The performance results are discussed in subsequent sections and address efficiency and scalability issues in both multicore platforms.

5.1 DATASETS

The validation of the 4SpecID tool aimed to guarantee functionality correctness with 3 different datasets. To select the datasets, we considered the following features: the availability of the dataset, the number of species, the number of clusters, and the number of records.

The datasets have different workloads, but they do not depend directly on the number of records. In previous sections, some examples used graphs as the conceptual representation of data and the complexity of the problem was also more easily described by the number of edges of the graph.

	#Records	**#Clusters**	**#Institutions**	**#Species**	**#Edges**
Canidae	933	21	61	22	27
Aves	29286	1088	74	1183	1604
Culicidae	36115	5287	126	5220	5952

Table 3: **Datasets Numerical Description**

Canidae

This dataset describes specimens from the biological family of dog-like carnivorans. It is a relatively small dataset with a low number of records and edges. In Section 5.3, this dataset

is used to evaluate the performance of the auditing algorithm, but its primary use is to validate against the Matlab version.

Aves

The *Aves* dataset characterizes birds as a group of warm-blooded vertebrates constituting the class *Aves*. This dataset has more than 1500 edges and 29000 records, comparatively to the *Canidae* dataset, this dataset has 59 times more edges and 31 times more records.

Culicidae

The mosquitoes are a family of small, midge-like flies: the *Culicidae*. The number of records and edges is higher than previous datasets being 36115 and 5952. This is the most significant dataset, it is the most important to assess the grading algorithm's performance.

5.2 PLATFORMS

As stated in preceding chapters, this work uses two platforms, a laptop and a compute server, to test and analyze performance results; their key features are in Table 4. The laptop platform usage is to verify that the tool can be viable in terms of execution time. The server platform assesses how the application performs in a common environment in HPC.

	Laptop	**Compute Server**
Operating System	MacOS 10.14	CentOS 6.3
CPU Manufacturer	Intel	Intel
Architecture	Broadwell	Ivy Bridge
Model	Core i5-5257U	E5-2695
#Cores / #Threads	2/4	12/24
#Sockets	1	2
Clock Frequency	2.7 GHz	2.4 GHz
Vector Extensions	Intel SSE4.1, Intel SSE4.2, Intel AVX2	Intel SSE4.1, Intel SSE4.2, Intel AVX
FP Performance Peak	86.4 GFlops	460.8 GFlops
L1 Cache (I+D)	64 KiB	64 KiB
L2 Cache (per core)	512 KiB	256 KiB
L3 Cache (shared)	3 MiB	30 MiB
Associativity L1/l2/L3	8/8/12 way set associative	8/8/20 way set associative
RAM Memory	8 GiB	64 GiB
RAM Latency	13.77 ns	14.16 ns
Memory Bandwidth Peak	30 GiB/s	47 GiB/s
#Memory Channels/Socket	2	4

Table 4: **Target Hardware Platforms**

To conduct the performance tests in these platforms, CLI of the 4SpecID and scripts, written in shell and Python, were used to automate, obtaining the results and diminishing

external interferences. Finally, the performance tests always used a version compiled with Clang 10.0.1 and GCC 7.2.0 in the desktop and server platforms, respectively.

5.3 PERFORMANCE ANALYSIS

The evaluation of the 4SpecID performance tool is displayed and discussed throughout the next sections. This evaluation contemplates all three datasets performance results when executed with the versions of the different algorithms. We use the 4SpecID CLI tool to accelerate the process of obtaining results.

Before using any parallelization technique, the sequential version was tested and improved. Since one this project has as objective the development of a high-performance portable software tool, we developed versions of the auditing algorithm using different parallelism approaches. These versions used both a shared-memory model and a hybrid model, and different work distribution approaches were tested and presented in subsequent sections.

5.3.1 *Sequential Approach*

After the Matlab version port to C++, the algorithm needed to be improved since it had a high execution time and to be possible to create parallel versions.

The first C++ version was a faithful port of the Matlab version. Therefore, it was possible to apply primary optimization techniques to improve its performance. These optimizations added local variables when iterating on data, removing unnecessary memory accesses or improving the operations asymptotic complexity. The algorithm performed all the necessary steps to audit a species, instead of applying each step on all species, one by one.

The approach of loading the dataset information to memory was to load all features of each record. This approach loaded and parsed unnecessary data stored by a vector container, where each item is a vector of strings representing each feature. Therefore, to optimize this approach, the 4SpecID tool only loads and parses the species name, cluster, and storing institutions and stores these features in Record objects. Then, the Record objects are the input to the group operation, which agglomerates all Record objects to Species objects and stores them in a map container. These optimizations lead to lower memory consumption and a more efficient data structure to store the necessary information.

Finally, the algorithm needed to retrieve and parse the data requests issued when it was necessary to fetch data from the neighbours of a cluster. The algorithm also checks if the cluster and its neighbours form a connected component. These steps of the algorithm were developed using the standard library of C++, which is portable for almost all operating systems. The Boost library, however, offers support for different tasks and structures. The libraries cover use cases such as linear algebra, generation of pseudorandom numbers,

multithreading, image processing, regular expressions, and unit testing, and a significant effort was made to port the library to as many compilers as possible. Therefore, the last optimization was to use the graph and regular expressions library from Boost.

Table 5 compares the execution times of the different optimization techniques on the compute server.

	Canidae	**Aves**	**Culicidae**
Initial version	10.6	673	14474
Improved data traversal	10	162	2417
Use of a map data structure	10.2	172	2352
Use of the Boost Graph Library	10.7	170	2308

Table 5: **Execution times(s) in seconds of different sequential versions using the Compute Server**

5.3.2 *Shared Memory Approach*

The development of this version is necessary to use all the available resources in a computer system. The target platforms are laptops and compute servers and as testbed the platforms presented in Table 4 were used. As stated before, each platform has several cores and the shared memory approach takes advantage of these cores to improve the overall execution time.

The execution of parallel algorithms requires that, at some point, threads are created, assigned with work, and joined on terminating the program execution. Therefore, to address this thread management problem, different libraries like OpenMP, IntelTBB, and Boost were considered, since they provide the necessary mechanisms. However, OpenMP and IntelTBB were discarded because both use the fork/join model that requires the creation of threads at the beginning of each parallel region, which would affect the algorithm's performance when executed more than once during the lifetime of the program. IntelTBB is hardware dependent and only Intel products can use it. Therefore, the shared-memory implementation of the algorithm uses the thread pool class from the Boost library.

Figure 13 compares the execution times of two different approaches of work distribution for the 3 input datasets: one that only issues auditing tasks to the thread pool and one that issues auditing and data access. It is clear that the second approach is better than the first one and this happens because, as stated before, auditing tasks may require to obtain data through requests over the network. The number of requests varies with the number of clusters that a species has; hence, the first approach may issue more sequential data access requests in the context of an auditing task.

The 4SpecID uses all the cores of the underlying system to perform the grading of a dataset. From a computational point of view, the algorithm performs two computational

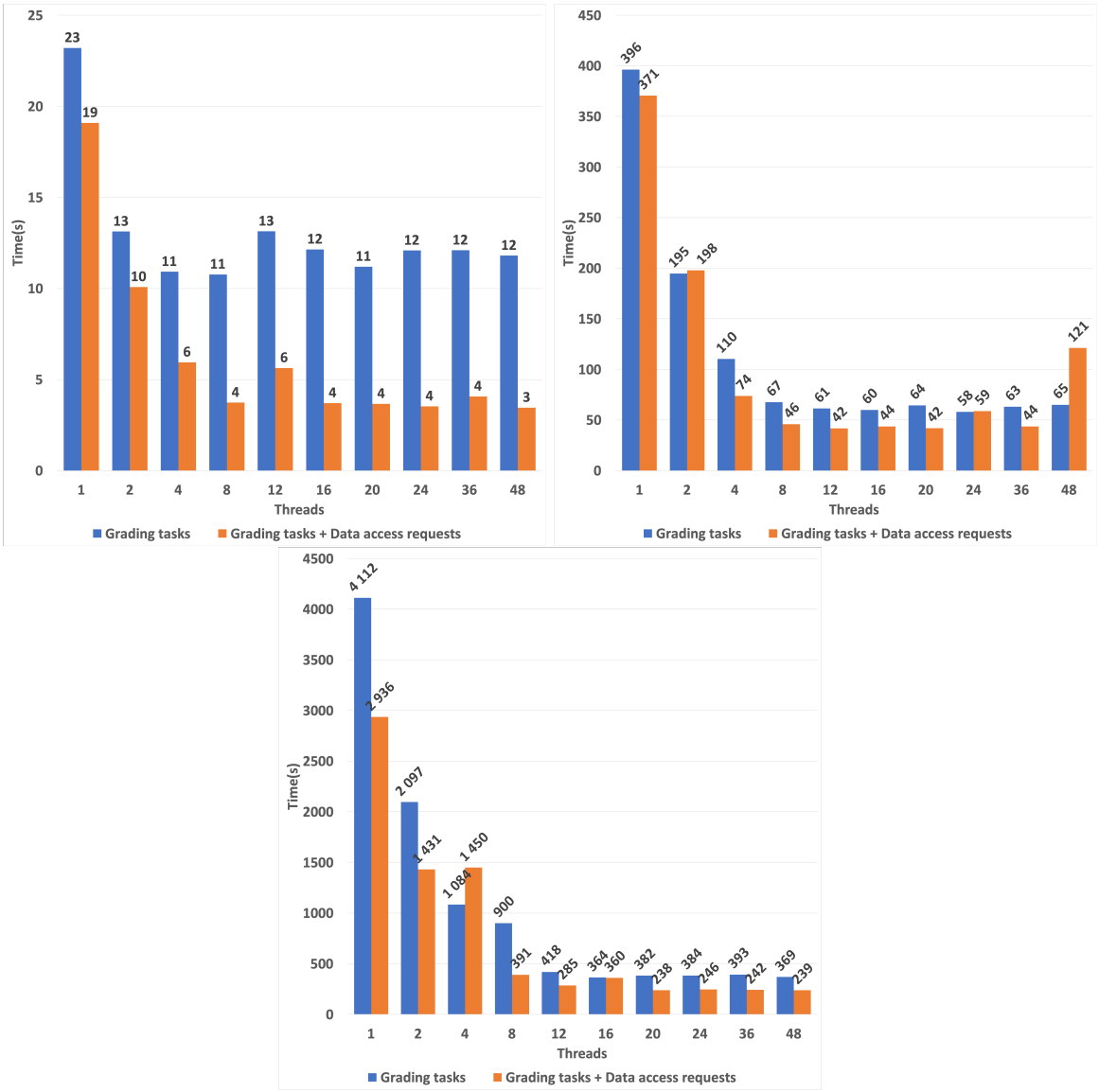


Figure 13: **Compute server execution times on datasets: Canidae (top left), Aves (top right) and Culicidae (bottom).**

tasks: the grading and the data access requests tasks. These tasks may be executed in parallel, if there are available cores, therefore decreasing the overall execution time. However, the tool cannot perform better when more than 20 cores are used, achieving a speedup of 12.

Figure 14 presents the execution times of the second approach on a laptop platform only using the approach of execution data access and grading tasks in parallel. The application shows similar performance to the server platform. The speedups for both target platforms are displayed in Figures 15 and 16.

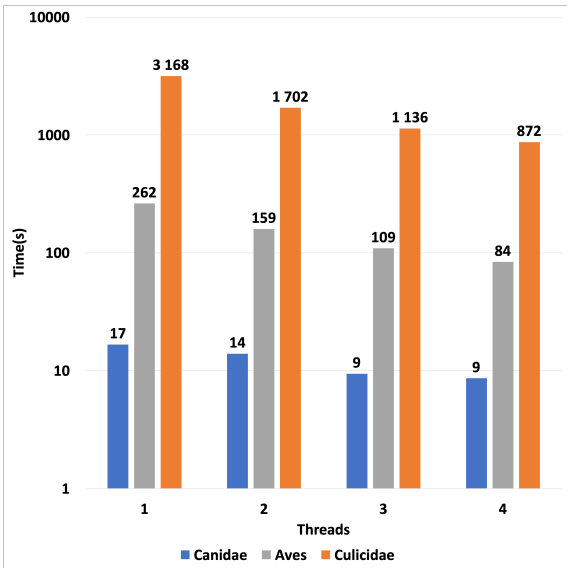


Figure 14: **Laptop execution times with all datasets**

An analysis of these results reveals that the use of more than sixteen threads with small datasets does not bring any performance improvement; this may happen due to the overhead of load distribution of small datasets.

The memory architecture of the server platform is *Non-Uniform Memory Access* (NUMA), where each CPU device has its own memory controller, which allows a core to have a much faster access to the local RAM than the RAM on the neighbour CPU device. For optimal performance, cores should access data on the RAM connected to the same device. During the execution it may be necessary to do network accesses and the latency of the data access requests is not under the developer's program control.

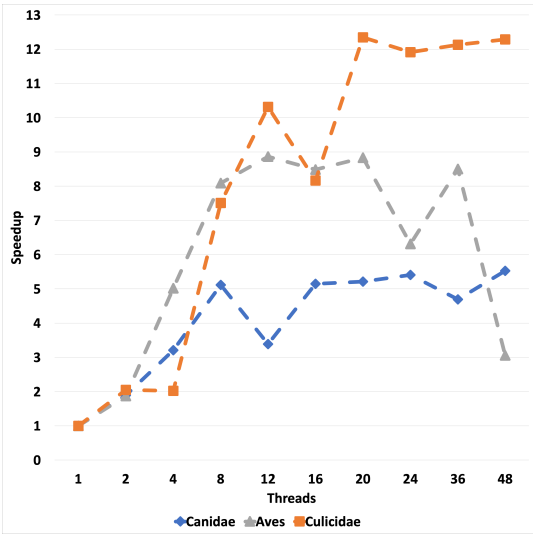


Figure 15: **Server speedup**

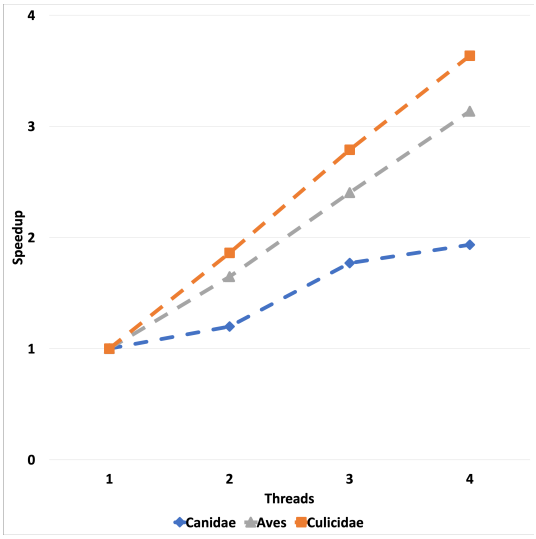


Figure 16: **Laptop speedup**

5.3.3 *Distributed Memory Approach*

Since the target compute server is dual-socket, we also developed a hybrid memory model application aiming to reduce the possible performance degradation due to the NUMA architecture.

The application implementation followed the methodology presented in subsection 4.3.4 and it uses OpenMPI to allow interprocess communication. The OpenMPI library offers different communication primitives and the application uses, whenever it can, group communication primitives which are more efficient [Sinha and Das (2009)]. The communication calls are used by the application to divide the workload and to broadcast the distance matrix if needed.

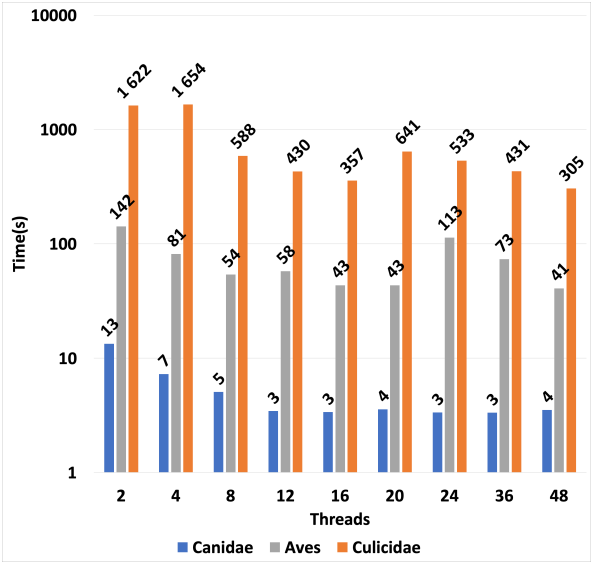


Figure 17: **Server execution times with the hybrid memory model**

The compute server described in the 4 was used to test the hybrid application's. In each test, two processes with the same number of threads were created.

Figure 17 shows execution times on a dual socket platform and how it behaves with different number of threads. It is possible to verify that the distributed memory application, using this hardware configuration, has a similar performance to the shared memory application.

5.3.4 *VTune analysis*

The VTune tool collects measurements from the algorithm to create a report that may have insights about its performance. The following reports are from the shared-memory version of the annotation algorithm. The analysis of the report of VTune is important since it offer

useful insights, such as, confirmation of the algorithm's efficiency using the underlying platform resources, spent time per function, and the threads state.

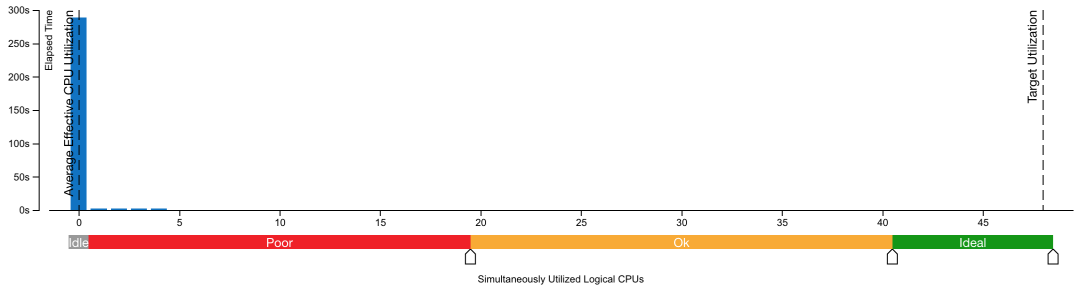


Figure 18: **Concurrency report from VTune**

From the report that VTune produces using 48 threads and the *Culicidae* dataset it possible to see that most of the time the number of CPUs that process simultaneously is 0.

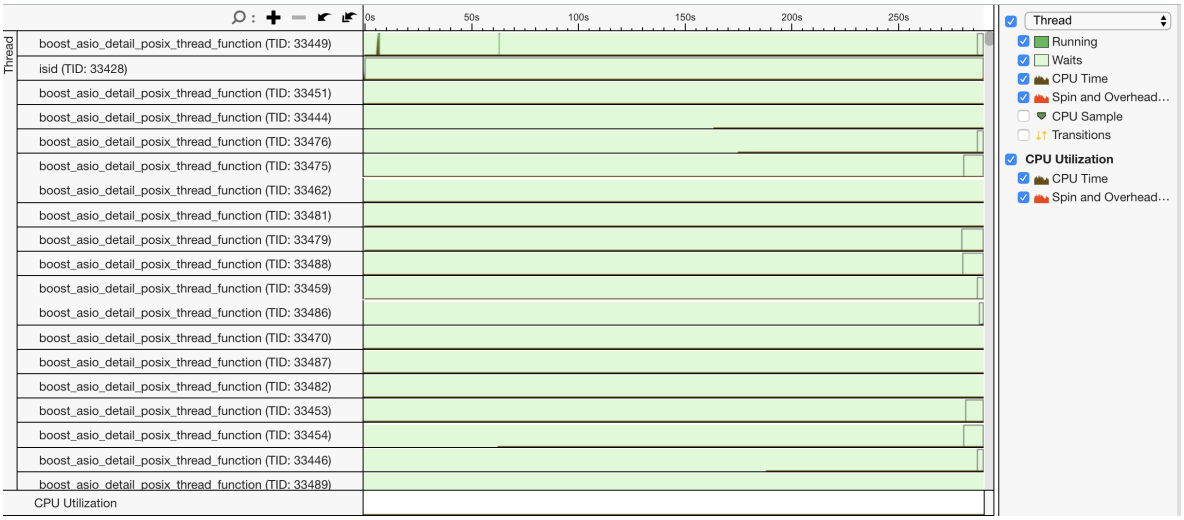


Figure 19: **Thread state report from VTune**

The Figure 19 and 20 show that most of the time the threads are waiting through most of the execution and the function *parse_bold_data* is where most of the computing time is spent on due to the fetch and parse of neighbour data.

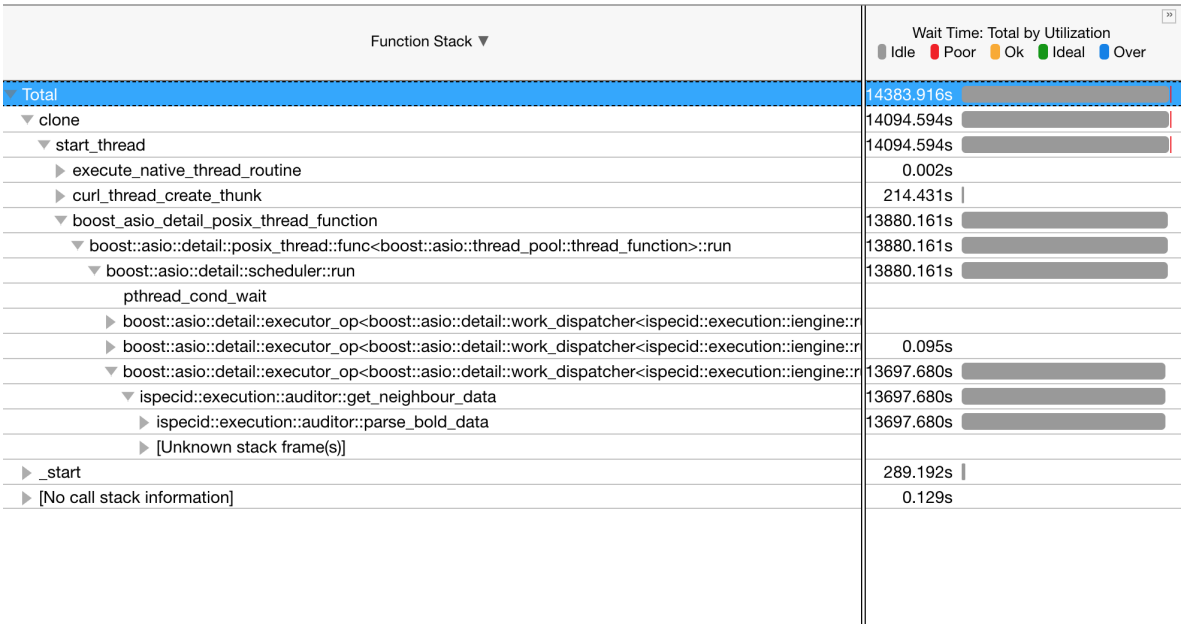


Figure 20: **Top-down tree from VTune**

6

CONCLUSION

This dissertation presented an approach to audit and correct DNA databases, which aimed to be time-efficient and provided the user useful functionalities to perform studies that depend on DNA databases.

This project resulted in the port and optimization of the auditing method specifically to these databases. The algorithm implementation is contained in a new application, the 4SpecID, which offers different known approaches to handle the databases, such as a graph and a record editors. The application also simplified tasks such as data filtering, which allowed the user to remove specific entries, and file handling, which frees the user from this responsibility.

The C++ version results were compared with the original Matlab version. The validation showed that the new application produces correct results, using the available computational resources properly.

The 4SpecID application has two different interfaces and it is divided into namespaces. The design allows other developers to effortlessly implement new features and test the performance either on a personal computer or on a multi-server platform.

After evaluating the performance results, it possible to state that the application does not always efficiently use the underlying platform's resources. This misuse of resources occurs due to design flaws, such as not allocating the necessary data in the memory closest to the logical processor. However, the application has portable performance and functionality, and it offers a more time-efficient solution than the Matlab version.

The challenges presented in Chapter 1.2, which raised the problems of efficiency and data processing, were not fully overcome for the auditing algorithm. However, this work shows a more performant algorithm and an application that simplifies common tasks when handling DNA datasets. The 4SpecID is a modular application making it a software package where new approaches and functionalities can be easily added, integrated and tested.

BIBLIOGRAPHY

Mark Blaxter, Jenna Mann, Tom Chapman, Fran Thomas, Claire Whitton, Robin Floyd, and Eyualem Abebe. Defining operational taxonomic units using dna barcode data. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 360(1462):1935–1943, 2005. doi: 10.1098/rstb.2005.1725. URL https://royalsocietypublishing.org/doi/abs/10.1098/rstb.2005.1725.

Klemen Candek and Matjaz Kuntner. Dna barcoding gap: reliable species identification over morphological and geographical scales. *Molecular Ecology Resources*, 15(2):268–277, 2015. doi: 10.1111/1755-0998.12304. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/1755-0998.12304.

Peter J. A. Cock, Christopher J. Fields, Naohisa Goto, Michael L. Heuer, and Peter M. Rice. The sanger fastq file format for sequences with quality scores, and the solexa/illumina fastq variants. *Nucleic acids research*, 38(6):1767–1771, Apr 2010. ISSN 1362-4962. doi: 10.1093/nar/gkp1137. URL https://www.ncbi.nlm.nih.gov/pubmed/20015970.

Eduardo Conde-Sousa, Nádia Pinto, and António Amorim. Reference dna databases for forensic species identification: auditing algorithms. *Forensic Science International: Genetics Supplement Series*, 7, 10 2019. doi: 10.1016/j.fsigss.2019.10.091.

NCBI Resource Coordinators. Database resources of the national center for biotechnology information. *Nucleic acids research*, 45(D1):D12–D17, Jan 2017. ISSN 1362-4962. doi: 10.1093/nar/gkw1071. URL https://www.ncbi.nlm.nih.gov/pubmed/27899561.

Simon Creer, Kristy Deiner, Serita Frey, Dorota Porazinska, Pierre Taberlet, W. Kelley Thomas, Caitlin Potter, and Holly M. Bik. The ecologist's field guide to sequence-based identification of biodiversity. *Methods in Ecology and Evolution*, 7(9):1008–1018, 2016. doi: 10.1111/2041-210X.12574. URL https://besjournals.onlinelibrary.wiley.com/doi/abs/10.1111/2041-210X.12574.

Yoann Dufresne, Franck Lejzerowicz, Laure Apotheloz Perret-Gentil, Jan Pawlowski, and Tristan Cordier. Slim: a flexible web application for the reproducible processing of environmental dna metabarcoding data. *BMC Bioinformatics*, 20(1):88, 2019. ISSN 1471-2105. doi: 10.1186/s12859-019-2663-2. URL https://doi.org/10.1186/s12859-019-2663-2.

João Ferreira, João Sobral, and Alberto Proença. Jaskel: A java skeleton-based framework for structured cluster and grid computing. *Sixth IEEE International Symposium on Cluster*

Computing and the Grid (CCGrid 2006), pages 301–304, 01 2006. doi: 10.1109/CCGRID.2006.65.

Bruno Fosso, Monica Santamaria, Marinella Marzano, Daniel Alonso-Alemany, Gabriel Valiente, Giacinto Donvito, Alfonso Monaco, Pasquale Notarangelo, and Graziano Pesole. Biomas: a modular pipeline for bioinformatic analysis of metagenomic amplicons. *BMC Bioinformatics*, 16(1):203, 2015. ISSN 1471-2105. doi: 10.1186/s12859-015-0595-z. URL https://doi.org/10.1186/s12859-015-0595-z.

Mehrdad Hajibabaei, Gregory A.C. Singer, Paul D.N. Hebert, and Donal A. Hickey. Dna barcoding: how it complements taxonomy, molecular phylogenetics and population genetics. *Trends in Genetics*, 23(4):167 – 172, 2007. ISSN 0168-9525. doi: https://doi.org/10.1016/j.tig.2007.02.001. URL http://www.sciencedirect.com/science/article/pii/S0168952507000364.

Paul D. N. Hebert and T. Ryan Gregory. The Promise of DNA Barcoding for Taxonomy. *Systematic Biology*, 54(5):852–859, 10 2005. ISSN 1063-5157. doi: 10.1080/10635150500354886. URL https://doi.org/10.1080/10635150500354886.

Paul D. N. Hebert, Alina Cywinska, Shelley L. Ball, and Jeremy R. deWaard. Biological identifications through dna barcodes. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 270(1512):313–321, 2003. doi: 10.1098/rspb.2002.2218. URL https://royalsocietypublishing.org/doi/abs/10.1098/rspb.2002.2218.

Goo Jun, Mary Kate Wing, Gonçalo R. Abecasis, and Hyun Min Kang. An efficient and scalable analysis framework for variant extraction and refinement from population-scale dna sequence data. *Genome research*, 25(6):918–925, Jun 2015. ISSN 1549-5469. doi: 10.1101/gr.176552.114. URL https://www.ncbi.nlm.nih.gov/pubmed/25883319.

Mehdi Kchouk, Jean-Francois Gibrat, and Mourad Elloumi. Generations of sequencing technologies: From first to next generation. *Biology and Medicine*, 09, 01 2017. doi: 10.4172/0974-8369.1000395.

A. M. Maxam and W. Gilbert. A new method for sequencing dna. *Proceedings of the National Academy of Sciences of the United States of America*, 74(2):560–564, Feb 1977. ISSN 0027-8424. doi: 10.1073/pnas.74.2.560. URL https://www.ncbi.nlm.nih.gov/pubmed/265521.

Alice McCarthy. Third generation dna sequencing: Pacific biosciences' single molecule real time technology. *Chemistry & Biology*, 17(7):675 – 676, 2010. ISSN 1074-5521. doi: https://doi.org/10.1016/j.chembiol.2010.07.004. URL http://www.sciencedirect.com/science/article/pii/S1074552110002474.

Matthew Meyerson, Stacey Gabriel, and Gad Getz. Advances in understanding cancer genomes through second-generation sequencing. *Nature Reviews Genetics*, 11(10):685–696, 2010. ISSN 1471-0064. doi: 10.1038/nrg2841. URL https://doi.org/10.1038/nrg2841.

Stephen B. Montgomery, Micha Sammeth, Maria Gutierrez-Arcelus, Radoslaw P. Lach, Catherine Ingle, James Nisbett, Roderic Guigo, and Emmanouil T. Dermitzakis. Transcriptome genetics using second generation sequencing in a caucasian population. *Nature*, 464(7289):773–777, 2010. ISSN 1476-4687. doi: 10.1038/nature08903. URL https://doi.org/10.1038/nature08903.

L. M. Oliveira, T. Knebelsberger, M. Landi, P. Soares, M. J. Raupach, and F. O. Costa. Assembling and auditing a comprehensive dna barcode reference library for european marine fishes. *Journal of Fish Biology*, 89(6):2741–2754, 2016. doi: 10.1111/jfb.13169. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/jfb.13169.

Annam Pavan, Gireesh Babu, and Wazir Lakra. Dna metabarcoding: A new approach for rapid biodiversity assessment. *Journal of Cell science and Molecular Biology*, 2, 08 2015.

W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences*, 85(8):2444–2448, April 1988. ISSN 0027-8424, 1091-6490. doi: 10.1073/pnas.85.8.2444. URL http://www.pnas.org/cgi/doi/10.1073/pnas.85.8.2444.

Sujeevan Ratnasingham and Paul D. N. Hebert. bold: The barcode of life data system (http://www.barcodinglife.org). *Molecular ecology notes*, 7(3):355–364, May 2007. ISSN 1471-8278. doi: 10.1111/j.1471-8286.2007.01678.x. URL https://www.ncbi.nlm.nih.gov/pubmed/18784790.

Jason A. Reuter, Damek V. Spacek, and Michael P. Snyder. High-throughput sequencing technologies. *Molecular cell*, 58(4):586–597, May 2015. ISSN 1097-4164. doi: 10.1016/j.molcel.2015.05.004. URL https://www.ncbi.nlm.nih.gov/pubmed/26000844. 26000844[pmid].

F. Sanger, S. Nicklen, and A. R. Coulson. Dna sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Sciences of the United States of America*, 74(12):5463–5467, Dec 1977. ISSN 0027-8424. doi: 10.1073/pnas.74.12.5463. URL https://www.ncbi.nlm.nih.gov/pubmed/271968.

Eric E. Schadt, Steve Turner, and Andrew Kasarskis. A window into third-generation sequencing. *Human Molecular Genetics*, 19(R2):R227–R240, 09 2010. ISSN 0964-6906. doi: 10.1093/hmg/ddq416. URL https://doi.org/10.1093/hmg/ddq416.

M. C. Schatz, A. L. Delcher, and S. L. Salzberg. Assembly of large genomes using second-generation sequencing. *Genome Res.*, 20(9):1165–1173, Sep 2010.

Patrick D. Schloss and Sarah L. Westcott. Assessing and improving methods used in operational taxonomic unit-based approaches for 16s rrna gene sequence analysis. *Applied and environmental microbiology*, 77(10):3219–3226, May 2011. ISSN 1098-5336. doi: 10.1128/AEM.02810-10. URL https://www.ncbi.nlm.nih.gov/pubmed/21421784.

Jay Shendure, Shankar Balasubramanian, George M. Church, Walter Gilbert, Jane Rogers, Jeffery A. Schloss, and Robert H. Waterston. Dna sequencing at 40: past, present and future. *Nature*, 550(7676):345–353, 2017. ISSN 1476-4687. doi: 10.1038/nature24286. URL https://doi.org/10.1038/nature24286.

Arnab Sinha and Nabanita Das. A comparative study of the mpi communication primitives on a cluster. 01 2009.

Alexandros Stamatakis. RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics*, 30(9):1312–1313, 01 2014. ISSN 1367-4803. doi: 10.1093/bioinformatics/btu033. URL https://doi.org/10.1093/bioinformatics/btu033.

Jiajie Zhang, Paschalia Kapli, Pavlos Pavlidis, and Alexandros Stamatakis. A general species delimitation method with applications to phylogenetic placements. *Bioinformatics (Oxford, England)*, 29(22):2869–2876, Nov 2013. ISSN 1367-4811. doi: 10.1093/bioinformatics/btt499. URL https://www.ncbi.nlm.nih.gov/pubmed/23990417. 23990417[pmid].

Wei Zheng, Qi Mao, Robert J Genco, Jean Wactawski-Wende, Michael Buck, Yunpeng Cai, and Yijun Sun. A parallel computational framework for ultra-large-scale sequence clustering analysis. *Bioinformatics*, 35(3):380–388, 07 2018. ISSN 1367-4803. doi: 10.1093/bioinformatics/bty617. URL https://doi.org/10.1093/bioinformatics/bty617.

This master thesis was developed with support of SeARCH-ON2 HPC infrastructure.