



Facultade de Informática

UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
Mención en Nome da Mención

Study and performance comparison of different alternatives for the implementation of the RNA-seq technique

Estudante: Santiago Millán González

Dirección: Diego Andrade Canosa
Emilio José Padrón González

A Coruña, setembro de 2021.

"A cringe man will always think he's based, but a based man is truthful to his cringe self."

-Lao Tzu

Acknowledgements

Me, Myself & I.

Abstract

Genomics is a very interesting and growing field. Specifically, the analysis of RNA-seq data helps researchers to continue to discover new information about all types of organisms and to fight diseases. In this project we have tested different implementations of this technique and have obtained more than remarkable results that provide a clear perception of what is the state of the art of the technique and on which step of the analysis we should focus to accelerate it to the maximum. In addition, knowing these challenges, it has been possible to implement an RNA-seq data pre-processing pipeline for researches to use.



Resumo

A xenómica é un campo moi interesante e que está en aumento. En concreto, a análise de datos de RNA-seq axuda aos investigadores a seguir descubriendo nova información sobre todo tipo de organismos e loitando contra as enfermidades. Neste proxecto probáronse diferentes implementacións desta técnica e obtivérvnse resultados máis que notables que proporcionan unha clara percepción de cal é o estado do arte da técnica e en qué pasos da análise deberíamos centrarnos para acelerala na medida do posible. Ademais, coñecendo estes desafíos, foi posible implementar unha pipeline de pre-procesamento de datos RNA-seq para que a usen diferentes investigadores.

Keywords:

- High-Performance Computing
- RNA-seq
- Genomics
- Bioinformatics
- Pipeline
- Benchmark
- Nextflow
- NVIDIA
- High-throughput

Palabras chave:

- Computación de alto rendemento
- RNA-seq
- Xenómica
- Bioinformática
- Pipeline
- Proba de rendemento
- Nextflow
- NVIDIA
- Alto rendemento

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 1 |
| 1.1 | Justification | 1 |
| 1.2 | Objectives | 2 |
| 1.3 | Structure | 2 |
| 2 | State of the art | 5 |
| 2.1 | Biological background and RNA-seq until today | 5 |
| 2.2 | Data analysis pipeline and tools | 7 |
| 2.2.1 | Universal steps for analysis | 7 |
| 2.2.2 | GATK | 10 |
| 2.2.3 | Input and output files | 10 |
| 2.2.4 | Nvidia Clara Parabricks | 12 |
| 2.3 | Container technologies | 13 |
| 2.3.1 | Docker | 14 |
| 3 | Methodology and planning | 17 |
| 3.1 | Agile methodologies | 17 |
| 3.2 | SCRUM | 18 |
| 3.2.1 | Roles and events | 18 |
| 3.2.2 | Logs | 20 |
| 3.3 | Adaptation to the project | 20 |
| 3.4 | Planning | 21 |
| 3.5 | Costs | 22 |
| 4 | Development | 25 |
| 4.1 | Benchmarking options | 25 |
| 4.1.1 | GEO2RNA-seq R package | 25 |
| 4.1.2 | nfcorn/RNA-seq | 28 |

| | | |
|----------|---|-----------|
| 4.2 | Best practices variant calling workflow | 31 |
| 4.2.1 | Pipeline implementation | 32 |
| 4.2.2 | WDL execution | 35 |
| 4.2.3 | GPU-based workflow | 38 |
| 4.3 | Nextflow pipeline | 39 |
| 4.3.1 | Nextflow script | 40 |
| 4.3.2 | Dockerfile | 43 |
| 5 | Results | 47 |
| 5.1 | Evaluation | 47 |
| 5.1.1 | Cluster environment | 47 |
| 5.1.2 | Input datasets and methodology | 50 |
| 5.2 | Benchmark results | 52 |
| 5.3 | GPU and CPU comparison | 53 |
| 6 | Conclusion | 59 |
| 6.1 | Conclusion | 59 |
| 6.2 | Future lines of work | 60 |
| A | Pre-processing pipeline user guide | 63 |
| A.1 | Requirements | 63 |
| A.2 | Installation | 63 |
| A.3 | Execution | 64 |
| | List of Acronyms | 65 |
| | Glossary | 67 |
| | Bibliography | 69 |

List of Figures

| | | |
|-----|---|----|
| 2.1 | Obtaining RNA-seq data | 6 |
| 2.2 | FASTA format sample | 11 |
| 2.3 | FASTQ format sample | 12 |
| 2.4 | Nvidia Clara Parabricks RNA Pipeline | 14 |
| 2.5 | Minimal steps for creating a Docker container | 16 |
| 3.1 | Gantt diagram | 23 |
| 4.1 | Inclusion of several re-sequenced reads for depth coverage on a samplesheet | 29 |
| 5.1 | GEO2RNA-seq execution times | 52 |
| 5.2 | nfcore/RNA-seq pipeline execution times. | 53 |
| 5.3 | nfcore/RNA-seq pipeline memory consumption. | 54 |
| 5.4 | fq2bam results | 55 |
| 5.5 | Pipeline results | 56 |

List of Tables

| | | |
|-----|--|----|
| 2.1 | Some of the tools | 9 |
| 3.1 | Breakdown of hours dedicated | 24 |
| 3.2 | Project costs | 24 |
| 5.1 | Compute node description within rack 2 | 48 |
| 5.2 | Experiment input data | 52 |
| 5.3 | STAR metrics | 54 |

Introduction

GENOMICS is an interdisciplinary branch of biology that focuses on the structure, function, evolution and mapping of genomes. In contrast to genetics, that aims to study individual genes, the objective of genomics is to study, characterize and quantify the collection of all of an organism's genes. With the emergence of massive parallel next-generation sequencing technologies, tremendous progress has been made in these studies. This technique has revolutionized genomics, allowing labs to produce ultra-high throughput, scalable and quick applications for processing huge batches of data.

This chapter will provide an introduction to the study of the [RNA-sequencing \(RNA-seq\)](#) technology, a sequencing technique used to reveal the presence and quantity of [Ribonucleic acid \(RNA\)](#) in a biological sample. This technique has been key in the understanding of a wide spectrum of new diseases[1] or even muscle disorders[2]. First, a brief look into the gaps that exist in the current topic will be addressed, following with the justification of the research that is being carried out. Besides, the goals and limitations of this project will be stated, along with the structure of the dissertation.

1.1 Justification

RNA-seq is known for being a convoluted amalgam of processes, files and general rules that naturally match a [pipeline](#) structure, that is, a series of processes that are chained, where the output of one process is the input of the next one. Due to the complexity and the need to know both the field of computer science and biology, researches can have a hard time getting acquainted with this technique.

There are several well-established tools and workflows that are very well known among researches, but novices in the field may find themselves overwhelmed by so many options. This is because very little effort has been put in the standardization of these technologies, not counting Broad Institute of MIT and Harvard's endeavours.

In addition, due to the computational burdens that RNA-seq presents, special attention is being paid to improve the efficiency and performance of the workflows. Fine-tuning has to be conducted by the end user when researching, as there is no rule of thumb that applies to every kind of sample data.

1.2 Objectives

The main objective of this project is to assess the performance of several state-of-the-art tools and pipelines to find out, broadly speaking, which are the main **bottlenecks** that may arise in these applications. With this, some of the steps and processing that are done on genomic data will also be presented, thus helping novice users to get a general idea of the RNA-seq data analysis process. Moreover, since both **Central Processing Unit (CPU)** and **Graphical Processing Unit (GPU)** based approaches are being tested, this thesis will serve as a short introduction to the importance of **accelerators** in **High-Performance Computing (HPC)** environments.

Finally, with clear ideas after benchmarking and understanding of the different workflow alternatives, the development of a proprietary pipeline will be carried out, based on the GATK's best practices for data pre-processing. This pipeline will contain all mandatory steps and tools, as well as different optimizations collected throughout the aforementioned tests and quality control and visualization tools for researches.

1.3 Structure

This thesis will start with a dedicated chapter for the state-of-the-art of RNA-seq. As so, a brief look into the biological background is needed to educate the reader regarding biological terms. First, the common methods for library preparation will be addressed, followed with the disclosure of the general steps of the downstream data analysis. A short introduction to the Broad Institute's Genome Analysis Toolkit for CPU-based pipelines will be given, along with its GPU counterpart, NVIDIA Clara Parabricks. Finally, some mentions will be made about the different files that are handled, as well as the common containerization technologies employed.

Chapter 4 will initially present a couple of pipelines subjected to testing. Within this section, all the tools that make up these pipelines will be discussed, along with the configuration requirements and execution process. Following that, an introduction to how both the CPU and GPU based variant calling are composed and executed will be conveyed. Lastly, the technologies employed for the development of the proprietary pipeline and the container used will be presented.

Chapter 5 will introduce the reader to the results, with every application tested against the

same batches of sample data for consistency. Also, it is necessary to describe the execution environment and metrics used for the comparison.

Finally, in the last two chapters, the development methodology used, planning and costs for this thesis will be provided, along with a final conclusion and future lines of work.

State of the art

DUE to the complexity and the need to know the biological field on which the technique is based, it is essential to explain the current state of knowledge. To do so, it is necessary to talk not only about the central tenets of biology but also to know the history behind this technique that aims to classify and quantify, in broad strokes, the amount of RNA in a sample.

In these next sections, a series of topics will be discussed to become further familiar with this technique and gain a full understanding of the matter that will help to reflect on the computational challenges it presents.

2.1 Biological background and RNA-seq until today

Chemically, **Deoxyribonucleic acid (DNA)** is defined as a polymer of nucleotides —with the commonly known nitrogenous bases guanine, thymine, cytosine and adenine—. Usually, it is visualized as this helical chain that everyone is familiar with, containing letters *G-T-C-A* which correspond to the previously mentioned nitrogenous bases. The so-called genes are pieces of this strand of DNA that contain the genetic information and instructions necessary for the development and functioning of the organism of every living being. These are used to manufacture the corresponding protein in a process called *gene expression*. Enzymes that metabolize nutrients or generate the material necessary to prepare for cell division are all proteins.

This is where RNA comes into play. DNA can't act alone and needs an intermediate agent to act as a messenger expressing genes. All of these agents or molecules constitute what is called *transcriptome* and they define the cell itself and its functions within the big scheme. **Messenger RNA (mRNA)** is in charge of carrying the genetic information to the ribosomes where protein synthesis will take place via a process called *transcription*, hence the name. This is one of the reasons why mRNA is referred to as coding RNA, and also why it has been the most widely and frequently studied species. However, RNA is very versatile and some

molecules perform several functions unrelated to gene expression, such as catalysts or simple regulators. This group is known as non-coding RNA and some are being discovered today, expanding the repertoire.

Advances in high-throughput **Next-Generation Sequencing (NGS)** have made it possible to revolutionize transcriptomics by allowing the analysis of both coding and not coding species. To do so, one must transcribe RNA back to complementary DNA, thus enabling the opportunity to research and deep profile the transcriptome. This technique is what is known as RNA-seq[3]. Over the years, the process of obtaining RNA-seq data has been standardized and consists mainly of two universal steps: the **complementary DNA (cDNA)** library preparation and transcriptome sequencing.

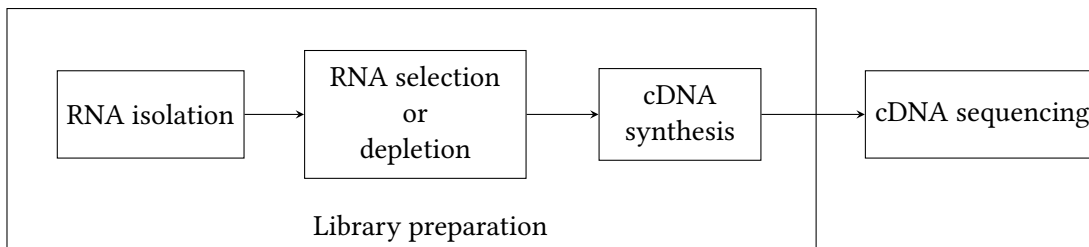


Figure 2.1: Obtaining RNA-seq data

In biology, a library is known as a group of DNA fragments that are meant to be propagated in a population of micro-organisms by inserting them into a cloning vector. Consequently, a series of steps must be taken to ensure its quality and reproducibility in any lab.

1. The experiment should begin by isolating the RNA from the biological sample by mixing it with different chemicals to reduce the amount of genomic DNA. Then, the quality of the remaining RNA after the degradation must be checked to ensure a sufficient measurement obtained by a bioanalyzer – this measurement is known as RNA Integrity Number—. This score must be taken later into consideration during the subsequent steps, as it can alter the results and the ending sequencing data.
2. Previously, emphasis has been placed on the variety of RNA types present in a sample. This must be taken into account when preparing the transcriptome. The next step must be, then, the selection or depletion of specific types of RNA molecules to ensure the signals of interest are being analyzed. Therefore, the biological instance must be manipulated thoroughly to enhance or limitate the detection of certain transcripts.

As instance, the ribosomal RNA account for over 95% of the total cellular pool. If the objective is analyzing **non-coding RNA (ncRNA)** and **ribosomal RNA (rRNA)** is not carefully eliminated from the sample, it will consume the bulk of the sequencing reads, reducing the overall coverage of reads of interest. Nevertheless, there must be a prior

understanding of the quality of the sample and protocol used in this step, as they may not be compatible and could be detrimental to the experiment.

3. Lastly, RNA must be converted into cDNA, as most sequencing technologies require DNA libraries because they leverage modern techniques and are more stable. As a curiosity, this conversion causes the strand orientation to be lost, although this information is particularly valuable in some experiments. Thus, adapters must be ligated in predetermined directions to either RNA or cDNA.

Finally, a sequencing platform for transcriptomics must be chosen to obtain a computer-readable format of the library. There are a lot of decisions to be made regarding cost, time and quality of the sample ratio. The majority of high-throughput sequencing platforms use methods involving sequencing many identical copies of a DNA molecule —ensemble-based—, but there are others that use a single-molecule approach.

Each protocol introduces several quantitative and quality standards, such as the introduction of *spike-ins* to assess coverage, quantification and sensitivity. The goal of this is to lower the variability of the experiment in differential transcriptome profiling studies. Also, the techniques employed in converting short reads into full-length transcripts are dependant of the platform used and can consequently affect the downstream analysis and must be taken into account.

2.2 Data analysis pipeline and tools

Right after getting the RNA-seq data, a series of pre-established steps must be followed to try to avoid inconsistencies in the scientific process. It should be remembered that we are talking about technologies such as [Whole Genome Sequencing \(WGS\)](#) that aim at determining the full genomic sequencing of an organism. Therefore, these analyses result in a high computational burden and require a certain infrastructure and power not achievable in many research centers.

Most of the public tools used in this study process focus on only one step of the entire workflow and do not provide information on a standardize pipeline or golden method. However, many tools have ended up paired with others by working well together and have ended up consolidating within the scientific world as one. Some attempts at standardization by [Genome Analysis Toolkit \(GATK\)](#) will be discussed later in this section.

2.2.1 Universal steps for analysis

The most important step of any RNA-seq data analysis experiment is the alignment and transcript assembly. In fact, it is so important that any other step or tool prior or subsequent to it

is treated as a pre-alignment or post-alignment procedure, respectively.


Generally, in this step, reads are mapped —i.e. aligned— to a reference genome for its later reconstruction into transcripts. Mapping RNA reads is way more challenging than mapping DNA, due to the nature and variability of the aforementioned. For this exact reason, alignment tools must have the ability to detect reads that map to non-contiguous portions of the reference genome. In order to help these tools, an index of the genome can be supplied, similar to the index of a book. Therefore, analogous to the way we find a chapter of a book faster by looking at the table of contents than by looking through every page of it, indices help the aligner by narrowing down the potential match of the sequence, saving not only time, but also memory. Additionally and in combination of these indexes, these tools make use of dynamic programming in order to find the optimal alignment. Currently and thanks to the efforts of institutions such as the [National Center for Biotechnology Information \(NCBI\)](https://www.ncbi.nlm.nih.gov/)¹, the current human genome reference sequence is readily available online for anyone to use in their experiments.

However, what happens if there are no reference genomes available due to the novelty of the experiment, the ignorance of the organism at issue or because the source material is completely altered from the reference model? To answer this question, a new technique called *de novo* was born. This approach does not need a reference genome, as its objective is to determine which reads go well together and should be considered as a consensus region (contigs). To do so, the main algorithms employed are overlap graphs —which identifies all associations between reads pair-wisely— and Bruijn graphs, that make use of k-mers (sub-sequences of length k) paired with hash tables to do the job. As this technique uses these fragments to infer transcriptome models, if the data is composed primarily of short-read sequences, it may be not sufficiently reliable. A hybrid solution for this would be to use a long-read-sequence-sample to mirror it and use as skeleton. Computationally, these approaches achieve an increase of speed at the cost of a small loss in sensitivity.

The most common following issue is the gene expression or transcriptome quantification. Depending on the method previously chosen for the alignment, the techniques vary from simply counting the k-mers to generating a matrix containing the number of reads of the transcript mapped to each part of the reference genome. Also, the aforementioned spike-ins that would have been carefully introduced in the library preparation step would now help in the absolute quantification, since RNA fragments concentrations would be known and could be used to infer the ratio of reads that mapped to each gene.

From this point onwards, there are several paths that can be taken depending on what has been designed for the experiment in particular. For starters, one of the most straight forward uses after accounting for gene expression quantification is differential expression. That is,

¹<https://www.ncbi.nlm.nih.gov/>



| | | |
|----------------|---------------------|---|
| Pre-alignment | FastQC ² | Quality control tool for high-throughput sequencing data |
| | Adapter Removal | Removes adapter sequences and trims low quality bases. It can also merge overlapping adapter sequences. |
| | Trimmomatic | For trimming raw reads of Illumina NGS data. |
| | FLASH | Fast Length Adjustment of SHort reads is used for merging paired-end data. |
| Alignment | Bowtie 2 | Memory-efficient tool for aligning sequencing reads to long reference sequences. |
| | HISAT2 | HISAT2 allows to map reads to both DNA and RNA. |
| | Salmon | Salmon is able to quantify the expression of transcripts and map using different techniques. |
| | STAR | STAR is an ultrafast universal RNA-seq aligner. |
| | TopHat | TopHat can only align reads to mammalian-sized genomes. |
| Post-alignment | BamTools | This tool helps handling BAM files. |
| | Picard | Picard is a suite that enables users to manipulate high-throughput sequencing data. |
| | SamTools | Samtools covers a variety of post-processing techniques for sequencing data. |
| | VCFTools | VCFTools is able to manage and process VCF files. |

Table 2.1: Some of the tools

to detect transcripts revealing differences in gene expression across two or more conditions —e.g., treated vs not treated—. To do so, a series of statistical models have been developed. Historical approaches such as the Poisson or normal distributions have been discarded due to biological variability not being correctly accounted by these assumptions. As instance, a highly expressed gene can "get all the attention" and be matched with most of the reads, reducing statistical variability. This bias and the common underestimation of sampling error often mislead these analysis. Choosing a tool that works well with the data in advance is hint a of good experimental design.

Another common objective of this experiments is variant discovery. RNA-seq data is used to discover and process variations in DNA helical stranding. The exchange of a single nucleotide at a specific spot in the genome may greatly affect an individual's susceptibility to a certain disease. If this substitution is present in a considerable fraction of the population it is considered a single-nucleotide *polymorphism*. If, on the contrary, this substitution is due to a somatic mutation caused by a disease, it is considered a single-nucleotide *variation* —e.g., a viral DNA sample obtained by means of a **polymerase chain reaction (PCR)** test may contain several **single-nucleotide variation (SNV)**—. Additionally, **variant calling also accounts** for

²To avoid the bulk of references to every bioinformatic tool used in this project, the interested reader can check this listing of several tools: https://en.wikipedia.org/wiki/List_of_RNA-Seq_bioinformatics_tools

insertions and deletions of bases in the genome, known as *indels*.

Lastly, it is worth mentioning that RNA-seq is a convoluted multistep process that generates tons of bias and corruption in data, as it involves thousands of small manipulations. Thus, quality control and assessment throughout the workflow is key to achieve good results and to avoid cumulative errors in the downstream analysis.

2.2.2 GATK

This last process is what [GATK](#)³ is trying to standardize. The Genome Analysis Toolkit developed by the Broad Institute of MIT and Harvard aims to be the industry standard for variant calling —i.e., identifying [single-nucleotide polymorphism \(SNP\)](#) and indels in germline RNA-seq data—. Additionally to continuing to expand and broaden the spectrum of variant calling applications, GATK also includes several quality control and processing of [high-throughput sequencing data](#), often generated by the [Illumina](#)⁴ protocol. Although conceived initially for human genomics, these utilities also apply to the genomic data of any organism.

GATK in its current version provides best practices workflow recommendations, having in mind maximum computational efficiency and results accuracy. Although workflows are tailored to particular applications, their general structure and analysis phases are:

1. **Pre-processing.** It involves the pre-processing of raw data in order to produce analysis ready [Binary Alignment Map \(BAM\)](#) files. It is usually comprised of the alignment step followed by any other data cleanup operation needed to account for technical biases.
2. **Variant discovery.** From analysis ready BAM files to [Variant Call Format \(VCF\)](#) files, this step involves identifying genomic variation.
3. **Other.** Additional steps are required depending on the application.

Workflow scripts are also provided as reference implementations. These are written in [Workflow Description Language \(WDL\)](#) and are executed by an execution engine, such as Cromwell. The preferred way to execute these is on Terra, the Broad Institute's Cloud platform.

2.2.3 Input and output files

Since there are many file types that are being worked with and [traverse](#) the pipeline, it is best to talk about some of the best known ones. In the following, a brief explanation on what they are used for, their format and a few examples are going to be given.

³<https://gatk.broadinstitute.org/hc/en-us>

⁴<https://emea.support.illumina.com/s>


```
>NR_036753 1
CCATGTGCCATATGTGAGAAATCCTTCTGTC
TTATCAGAGAATCCACACAGAGGGAAAATCC
TGGCAGCAGTCAGCACTCAGGGT
>NR_125392 1
ATGTTTATCTGGCAGAAGAAATGTTATGATC
AGCAAGGCTTAGCAGTTTTACTGTGGATGTG
CGAATGTGGGAATCT
```



Figure 2.2: FASTA format sample

1. **FASTA.** This type of file contains genomic sequences and optionally information about them, each nucleotide being represented by a single-letter code. Although simple, this format requires some specific rules that must be followed. This makes it easier for text-processing tools such as R or parsing languages such as Python to manipulate the data. A FASTA file must always begin with a greater-than symbol (“>”) containing a summary description of the sequence. Besides, any line preceded by a semicolon (“;”) will be treated as a comment. After the first line, the sequence must be written without spaces, tabs, etc..., and usually, not mandatory, finish with an asterisk. Since these files are usually downloaded from the NCBI database, this format also supports a series of identifiers used to define additional metadata like access identifiers or patents. In the RNA-seq data analysis pipeline, the reference genome used for alignment usually comes in this format.
2. **GTF.** Used in conjunction with a FASTA file, the general transfer format is used to store gene descriptions and other DNA or RNA features. This opens the possibility to validate and verify that the data is correct and there are no errors. This file is comprised of 9 fields tab delimited by line.
3. **FASTQ.** The only difference with a FASTA file is that it contains the additional corresponding quality score of each sequence letter. The two first lines of the format are also similar: the greater-than symbol is substituted by a (“@”) character followed by the sequence identifier or description and continues with the raw sequence letters. The line following the sequence contains a “+” sign. Finally, the last line encodes the quality score from the above sequence and, of course, matches its length. The quality value corresponds to a byte going from 0x21 to 0x7e —i.e., from “!” to “~” in [ASCII](#)—. Also, some of the identifiers proposed by the NCBI are also supported. These files are used to store the RNA-seq reads generated by the Illumina protocol.
4. **SAM/BAM.** This file format is used to save information about reads mapped against the reference genome. It’s composed of a header section and the alignment information

```

@r0
GAACGATACCCACCCAACTATCGCCATTCCAGCAT
+
EDCCCBAAAA@@@>===<;9:99987776554
@r1
CCGAACTGGATGTCTCATGGGATAAAAATCATCCG
+
EDCCCBAAAA@@@>===<;9:99987776554

```

Figure 2.3: FASTQ format sample

section as tab delimited fields for each read. There are eleven mandatory fields and as many extra optional fields as necessary.

5. **VCF.** This file is pretty much exclusively used as the output of the variant calling step of the pipeline. As such, it stores gene sequence variations at single-nucleotide level. It has a header containing metadata. These lines start with a single (“#”) symbol or two if it contains a keyword —e.g., filedate or source—. It is followed by eight mandatory columns describing the variation information and as many dedicated ones as samples are included.
6. **TAB.** These are general purpose tab delimited text files. They are commonly used for parsers to manipulate and represent data.

2.2.4 Nvidia Clara Parabricks

Nvidia Clara Parabricks⁵ is a computational suite for genomics. Borrowing from *Artificial Intelligence* (AI), HPC and Nvidia *Compute Unified Device Architecture* (CUDA)⁶ stacks, it can be used to address many needs of genomic labs with its GPU accelerated libraries and workflows[4].

But why are GPUs computational power the most attractive option when it comes to genomic analysis? And not only that, but for many other high performance applications that manipulate heavy workloads? In recent years, there has been an exponential growth in demand for graphic processing units due to them having specific characteristics that make them perfect for these types of loads previously mentioned.

A CPU and a GPU differ greatly in architecture and it’s not the latter being intrinsically better. This component is characterized by having many small processing cores and high memory bandwidth. Fundamentally, they are perfectly ready to run single instruction multiple data **stream** workloads. A GPU basically runs a single program **many** times over a dataset,

⁵<https://developer.nvidia.com/clara-parabricks>

⁶<https://developer.nvidia.com/cuda-zone>

having the possibility to execute those in parallel. If there is a need to run a single program over vast ranges of the same input and they can be treated independently, a GPU-based approach will work multiple times faster than a CPU can thanks to its parallel nature. It's all about getting dozens of operations all running at once on each thread —i.e., CUDA cores—. Also, GPUs often surpass the processor clock speed.

These are the reasons why they are primarily used for graphics processing. Programs called *shaders* are executed on each pixel. A GPU processing a high demanding application such as a game run at 1920x1080 and 60 frames per second would have to update the millions of pixels in less than a 1/60th of a second. Ultimately, a CPU couldn't even compete against that dedicated hardware.

This high computer power is also interesting for non-graphical, general purpose applications. As instance, with the cryptocurrency mining boom, another use was found, as GPUs can complete "blocks" of verified transactions on the blockchain at lightning-speeds and very efficiently. Also, the emergence of new *deep-learning* algorithms and huge data warehouses have made it necessary to leverage the performance and power of these components.

With Parabricks, NVidia has succeeded in developing GPU-accelerated GATK pipelines and some other third-party tools such as Google's DeepVariant⁷, a deep learning-based variant caller. This framework was designed not only to optimize acceleration, but also to maintain a high degree of quality and accuracy in the results. Besides, although centered around GATK best practices, it allows full configuration of the pipeline and parameters, in addition to choosing which software versions to run.

Although this framework supports a lot of different genomic applications, there are mainly two workflows specific for RNA-seq data analysis. The FQ2BAM application is considered as a standalone tool that has the conversion of reads into aligned sequences as objective. To do so, it employs the STAR⁸ tool to map reads to a reference genome and also uses the SortSam and MarkDuplicates programs included in the Picard⁹ suite. The other major application is the GPU-accelerated GATK workflow for RNA-seq short read variant discovery (SNPs + Indels). As shown in [figure 2.4](#), the three first steps make up the FQ2BAM tool. Additionally, a couple of quality control and data make-up tools in SplitNCigar and BQSR follow the alignment. Finally, the Haplotype Caller is run to account for variant calling.

2.3 Container technologies

As shown, there's plenty of applications and software involved in the process of analyzing RNA-seq data. As such, and to make life easier for researchers that are not that acquainted

⁷<https://github.com/google/deepvariant>

⁸<https://github.com/alexdobin/STAR>

⁹<https://broadinstitute.github.io/picard/>

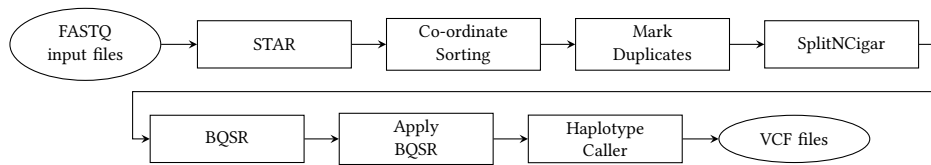


Figure 2.4: Nvidia Clara Parabricks RNA Pipeline

with the use of such technologies, we find ourselves in the need of using something to help us package that software and its dependencies. That way, this methodology can be standardized and homogeneous results can be achieved with no further complications or variations. That's where container technologies can come in handy.

These containers are instances that package the application and host all the dependencies it needs for its correct operation. The name is a perfect analogy to what happens on a cargo ship. Goods are placed into –usually– steel containers instead of being shipped in a specific way. This method not only standardizes keeping the goods together so that they can be transported from one ship to another, but also reduces costs. Containers are the ideal solution for moving software between machines or different development environments, such as a real production environment in a company or a test server. The fact that these machines can have different operative systems, libraries or network topologies is not a problem, since the application is perfectly isolated and cannot behave in an anomalous manner.

A distinction must be made between a virtual machine and a container: the first virtualizes an entire operating system and the application –or several, due to the dimension–, while the latter runs on the same host operating system in isolation without needing their own **Operative System (OS)**, since they share the same **Kernel**. This is the exact reason why a container is lighter. A physical server cannot run but a few virtual machines with their own OS, since there is a lot of overhead on the server as resources get used. However, a server could also run a single operating system with containers sharing its Kernel and resources, which should be treated as read-only to avoid interferences. Because of these advantages, many companies have opted for the use of containers accompanied by an orchestration software such as Kubernetes¹⁰ or Docker Swarm.

2.3.1 Docker

Although there are several alternatives such as RKT¹¹ or Linux Containers¹², Docker¹³ is the de facto choice when it comes to running these pipelines.

¹⁰ <https://kubernetes.io>

¹¹ <https://cloud.redhat.com/learn/topics/rkt>

¹² <https://linuxcontainers.org/>

¹³ <https://www.docker.com/>

The background service that builds, runs and distributes the Docker containers is called "dockerd" (Docker daemon). It communicates with the Docker client through a Rest [Application Programming Interfaces \(API\)](#) on top of a [UNIX socket](#) if they are running on the same system or through a network interface in the case of connecting remotely. A Docker client can communicate with one or more daemons. There is also the possibility that a Docker daemon communicates with other daemons to manage different services. Thus, a call from the client to `docker pull` would notify the daemon and it would transparently perform the necessary operations to access a registry containing the requested image.

As shown in figure 2.5, there are three main steps to be taken before obtaining a functional Docker container.

1. A Dockerfile is not but a text file containing a set of instructions needed to build an image. This way, a simple call to `docker build` would be enough to start executing those instructions sequentially. The format of this file is mainly composed by a series of instructions of the form `<keyword, arguments>`. A series of best practices are suggested in order to standardize the creation of efficient and as light as possible images, and that are easily maintainable in terms of layer creation.
2. A Docker Image is defined as a read-only template on which the container is to be created. Typically, it is based on a base image with an additional customization layer. Each instruction in the Dockerfile is considered as a layer. When an alteration is made in that template, only the affected layers are reconstructed when it comes to rebuilding that image. This is exactly what makes Docker so lightweight and fast compared to other virtualization technologies.

An image can be pulled directly from an image repository. By default, the daemon will always look for it on Docker Hub¹⁴, unless another registry is specified in the configuration files.

3. A container is the running instance of an image. In the specific case of Docker, the latter becomes a container when it is run on Docker Engine. This way, containerized software will always run the same, regardless of infrastructure.

However, the fact that Docker needs root privilege can be a threat^[5] in shared environments. Alternative implementations such as uDocker¹⁵ may be attractive for this kind of situations. The other container software that solves this problem is Singularity¹⁶. It is one of the most widely used systems for high-performance computing (HPC), as it offers close to

¹⁴ <https://hub.docker.com/>

¹⁵ <https://github.com/indigo-dc/udocker>

¹⁶ <https://sylabs.io/singularity/>

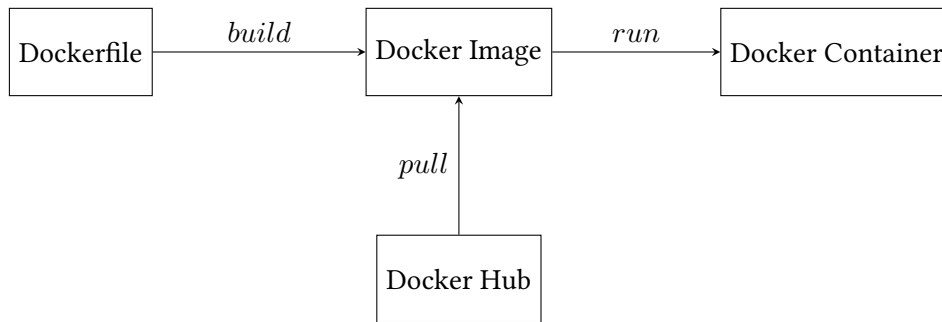


Figure 2.5: Minimal steps for creating a Docker container

bare-metal performance while being secure. It also can be used with Docker Images. This integration is due to the fact that developers and scientists are still eager to using Docker and have already put much resources into creating those images.

Methodology and planning

FOR this project, it was decided to use an agile methodology, adapting on of its strands: the Scrum framework.

In this chapter, the characteristics and advantages of agile methodologies will be presented, along with some basic notions of Scrum. Of course, this methodology had to be adapted to this project. Finally, clear reference will be made to the final planning of the entire thesis, including costs.

3.1 Agile methodologies

There are many methodologies designed for application in software development projects. In fact, the study of this methodologies is a fully consolidated field of software engineering. One of the most widespread in recent years is the agile methodology. It allows to adapt the way of working to the project conditions, providing both flexibility and efficiency to the planning, thus often obtaining a better final product.

The agile methodology emerges as an alternative to traditional work methods, usually too structured and unchanging. It mainly focuses on obtaining a final working product for each iterative life cycle. That is, with this clear divisions and iterations, fundamental requirements can be met sooner to focus later on extra functionalities or refinements of a feature.

All the characteristics of this methodology are gathered in the *Agile manifesto*[6], elaborated as a result of several meetings of many of the most recognized software developers. Some of its key points are:

1. Individuals and their interactions over tools and processes. This allows the focus to be placed on people, achieving a more pleasant working environment and better theoretical results. Regarding the interactions, the aim is to promote the responsibility of the individual within a team, providing greater autonomy and transparency to the rest of

the members of said team. Fluid team communication, equal participation and collaborative work are key to ensure that all parts are focused towards a common objective.

2. Total collaboration with the client. Quick and accurate feedback helps building a better communication model, closer to the client. The client feels like a member of the team, thus achieving a better relationship and satisfaction for both parties. In addition, the proximity to the client provides differential value that would not be achieved otherwise, including knowledge about the customer's vision of the product and avoiding deviations in the planning.
3. Quick response to changes. Since this continuous work environment is promoted and hierarchies are avoided, the operation in the face of changes is much faster: efficiency and optimization are sought. This review rhythm allows to better resize projects, minimizing risks, time, and costs.

3.2 SCRUM

SCRUM[7] is a framework based on partial and regular deliveries of the product, prioritizing the most important functionalities and always seeking the greatest benefit for the client. Its use is recommended primarily in environments where results are needed soon and specifications are subject to many changes or poorly defined initially. Needless to say, SCRUM inherits most of the characteristics of agile methodologies: greater productivity[8], continuous communication among individuals and clients, flexibility to changes... The two main components of the SCRUM working model are presented below.

3.2.1 Roles and events

Each individual must assume one of the following roles:

1. **Product Owner.** He/she is in charge of optimizing and maximizing the product's value. The product owner is also in charge of managing the value flow of the product through the Product Backlog. His/her work as an interlocutor with stakeholders and project sponsors is essential, along with his/her role as a loudspeaker for client requests and requirements. In each Sprint, the Product Owner must make an investment in development that has to constantly increase the value of the product. It is essential to give the necessary value to this role so that any decision that ultimately affects the project can be made without issues and prior consultations.
2. **SCRUM Master.** At the same time, the Scrum Master has to main roles, in addition to being responsible for mentoring and training, organizing meetings and events:

- (a) Managing the Scrum process. He/she is in charge of ensuring that the SCRUM process is carried out correctly, as well as facilitating its execution and mechanics. The SCRUM Master has to always keep in mind that the methodology must be a source of value generation.
 - (b) Remove obstacles. The integrity of this methodology must always be kept, so emerging impediments should not affect its ability to deliver value. Thus, the transmission of its benefits to the organization and ease of implementation is always present.
3. **Development team.** Usually consists of three to nine professionals who are responsible for developing the product, managing to deliver a software increment at the end of the development cycle. All members of the team must know their role. How they decide to manage themselves internally is their own responsibility and they will be accountable for it as one. All external individuals should avoid intervening in the development team dynamic.

Product development is organized in iterations consisting of the following events:

1. **Sprint.** It is the basic SCRUM time unit, and it covers all other events. With each Sprint, the development team must achieve a product increment, that is, a partial delivery consisting of the current version of the product. Each increment must be functional and an improvement over the previous one. Each Sprint usually lasts between two and four weeks, although it is variable.
2. **Sprint planning.** A reunion where the objectives for the Sprint are discussed. Requirements for the product are presented by the Product Owner and the development team is responsible of selecting which requirements are up to be included in the current Sprint. The tasks to be done in the iteration will be distributed among the team members and they are in charge of self-regulating and self-managing themselves to achieve and efficient development.
3. **Sprint review.** The last reunion of the Sprint where the Product Owner presents the software increment to the client and the development team shows its functionalities. Of course, the client may propose some changes, and the Product Owner is in charge of taking them into account and add them to the Product Backlog. Additionally, an internal reunion can be arranged to discuss how good of an implementation of the methodology that Sprint has been.
4. **Daily Meetings.** A small meeting is held everyday to organize the team and have a general talk about the progress of the Sprint. The SCRUM Master must participate on this meetings.

3.2.2 Logs

Finally, the work team builds some assets proposed by the SCRUM framework that aim to offer maximum cohesion and transparency among its members:

1. **Product Backlog.** All tasks necessary to complete the final product are gathered here. The Product Owner manages this log.
2. **Sprint Backlog.** It is a subset of the Product Backlog, containing all tasks that are to be completed within the current Sprint.

3.3 Adaptation to the project

As stated before, minor modifications have had to be made to the SCRUM framework to adapt it to the development of an undergraduate thesis.

The first key point to address is the distribution of roles. The methodology is designed for medium sized teams of 4-10 people. However, as there are only three people available —i.e., the student and both the tutors—, roles have had to be modified. On one hand, the student has had to take on the roles of SCRUM Master and development team. On the other hand, the tutors have been assigned the role of Product Owner.

Sprints have been used as a temporal division for the project, with an average duration of each one of about 3 weeks. Nevertheless, this is a rough estimate, since the student has not always been able to dedicate himself exclusively to the development of the thesis. Unforeseen events and periods of partial abandonment of the project arose, during which the student has had to focus on other subjects of the degree.

The format of the Sprints is as follows: at the beginning of each sprint, meetings were held covering both the Sprint planning and the Sprint review. That is, in each iteration the Product Owner defined the tasks and objectives to be accomplished for the sprint —i.e., Sprint Backlog—. Additionally, these meetings served to review what was done in the previous Sprint, observing the result of the development iteration and proposing improvements or resolving doubts.

The only exception to this rule was the first Sprint meeting, where the Product Owner built the Product Backlog and defined all the priorities of the tasks, so no Sprint review was needed as there was no previous iteration. Moreover, the Product Backlog received several modifications throughout the development, adapting it to the remaining time of the school year and depending on the progress that had been made up to that point.

It is key to highlight the flexibility that this type of methodology provides, since a research work always raises difficulties as one goes deeper into a topic. Being the first serious research work of the student, this flexibility helped him to organize himself and always maintain a

common objective with the tutors, as a development team would do in normal circumstances under this development framework.

3.4 Planning

In this section, an explanation of all the work that has been done in each Sprint will be presented:

- (a) **Sprint 0.** In this Sprint, the student carefully read a lot of documentation and several papers related to the RNA-seq technique, as well as searching for tools and implementation alternatives. A general analysis was also made of the project planning. The appearance of difficulties and not knowing the limits of the subject made this Sprint last longer than the 3 weeks initially proposed per iteration. Regarding SCRUM events, a reunion was held to prepare the Product Backlog and Sprint Backlog for this iteration.
- (b) **Sprint 1.** This Sprint focused mainly on learning Docker, as it is a powerful tool that allows every implementation to achieve good reproducibility of the process and same conditions for every user. This Sprint's tasks were not planned initially, and they had to be introduced in the Product Backlog in this Sprint's reunion. Also, this Sprint shares some of its duration with the previous one, as more insight was needed regarding RNA-seq.
- (c) **Sprint 2.** Once sufficient knowledge of the subject was obtained, the student tried a couple of the best-known general purpose RNA-seq data analysis implementations and a use-case scenario was prepared for each of them. Again, a reunion was held at the beginning of this iteration to prepare the Sprint Backlog.
- (d) **Sprint 3.** In the Sprint planning for this iteration, the team chose to learn about different alternatives to the classic pipeline focused on differential expression. It was decided to investigate the best practices proposed by GATK for variant calling.
- (e) **Sprint 4.** This Sprint followed the previous one very similarly, with a careful review of Parabricks' GPU-accelerated solution.
- (f) **Sprint 5.** Knowing enough about the different implementations and having already achieved several use cases, testing on the CITIC's Pluton `cluster` was initiated. This Sprint began in the middle of the previous one, as 3 weeks seemed a bit excessive to learn about Nvidia's application and this tool was the last one needed in order to start testing.

- (g) **Sprint 6.** Parallel to the previous iteration, the development of the proprietary pipeline started, leveraging the different optimizations achieved by testing. This Sprint and the previous one lasted almost twice as long as the marked Sprint time due to the difficulty of these tasks.
- (h) **Sprint 7.** Using the Product Backlog, notes and documents profiled during all these Sprints, the dissertation of this thesis was written. In the end it added up to most of the time, also due to the coincidence with the final exams and the loss of concentration on the project.

In Figure 3.1 the reader can see a high-level Gantt Diagram, where the relationships between Sprints have been graphically represented. Although most of the relations are *Finish-to-Start* there are a couple key differences.

The first two Sprints share a *Start-to-Start* relation due to the fact that the student couldn't start researching Docker until some insight on the topic was already collected. Also, both iterations had to be finished before starting to play around with the different implementations.

Additionally, the Sprints 4 and 5 share a *Start-to-Start* relationship because the student could start testing all the tools while still learning a bit more about the most different of them all: the GPU-based approach.

Finally, Sprints 5 and 6 share a *Finish-to-Finish* relation, because the development of the pipeline couldn't finish until all the tests finished and the results were obtained, as they were used to optimize the final pipeline.

3.5 Costs

The total cost of the project would be that corresponding to the individuals that were part of it, as the only tool that costs money was used under a free trial license thanks to NVIDIA allowing researchers to use it freely due to the current situation of the pandemic.

In Table 3.1 shows the numbers of hours dedicated to the project for each Sprint by each of the members.

To calculate the final human cost of the project, it is enough to consult public average salaries that are available online and multiply them by the total hours that each one has dedicated to the project. In Table 3.2 the reader can see the final cost for this project.

Regarding materials and physical resources, all necessary hardware was provided by the [Centro de Investigacion en TIC \(CITIC\)](#). However, should this not be the case, we must account for it. Using the [Amazon Web Service \(AWS\) pricing calculator](#)¹, we can find out that the nodes used for testing would cost about 3.47€/h. If we multiply that by the number of

¹<https://calculator.aws/#/createCalculator/EC2>

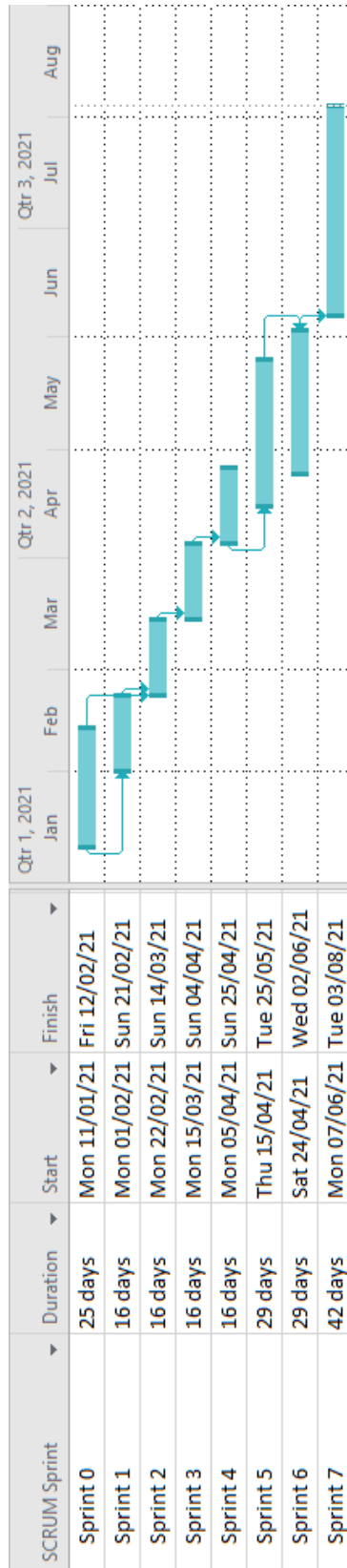


Figure 3.1: Gantt diagram

| Sprint | Student | Director 1 | Director 2 |
|--------------|---------|------------|------------|
| 0 | 42 | 4 | 4 |
| 1 | 12 | 2 | 2 |
| 2 | 35 | 2 | 2 |
| 3 | 27 | 2 | 2 |
| 4 | 11 | 2 | 2 |
| 5 | 55 | 4 | 4 |
| 6 | 35 | 2 | 2 |
| 7 | 80 | 2 | 2 |
| Total | 297 | 20 | 20 |

Table 3.1: Breakdown of hours dedicated

| Resource | Time (hours) | Cost (€/hour) | Total cost (€) |
|-------------------|--------------|---------------|----------------|
| Director 1 | 20 | 45 | 900 |
| Director 2 | 20 | 45 | 900 |
| Student | 297 | 20 | 5940 |
| Final cost | | | 7740 |

Table 3.2: Project costs

hours spent testing plus the computation time of all the execution alternatives, we get a rough estimate of 329.65€ that we would have to sum to the final cost of the project.

Development

AFTER this brief introduction, this chapter provides information of the actual development bulk of the project, including some of the most well known pipelines.

4.1 Benchmarking options

To assess the computational scope and importance of each tool, the next two applications were investigated as an entry point for the RNA-seq data analysis. Within this section, some of the information necessary and fundamental for the characterization of the two approaches will be found. This includes not only ~~not only~~ the tweaks that have been made in order to run them properly, but some of the code and insight of the two applications.

4.1.1 GEO2RNA-seq R package

To get acquainted with this topic, a quick look online may be enough to realize that most of the attention in the RNA-seq data analysis pipeline has been focused on differential expression analysis. Two pipelines for processing of RNA-seq data exist and are widely used. The Total RNA-Seq Analysis Package for R¹ is a partial RNA-seq pipeline implemented in R. However, does not include mapping and counting, and uses counts per genes as input. Given counts, TRAPR performs all following steps necessary for the detection of **Differentially Expressed Genes (DEG)**, but the statistical analysis is limited to DESeq2 and edgeR. EasyRNASeq² is a package available through Bioconductor. Again, it can only be used after initial mapping was performed. As well the statistical analysis is limited to DESeq2 and edgeR.

Consequently, a more powerful library that comprised all the steps of the pipeline was needed as an initial contact. The GEO2RNA-SEQ³ package turned out to be the best R imple-

¹<http://www.snubi.org/software/trapr/>

²<https://bioconductor.org/packages/release/bioc/html/easyRNASeq.html>

³<https://anaconda.org/xentrics/r-geo2rnaseq>

mentation for this[9], as it has a modular programming approach and alternative workflow options. Being a R package, it's also lightweight and easy to install. One of the most notorious features is the fact that this implementation takes advantage of all the amount of metadata that often comes with the raw reads after sequencing. This includes important wet-lab metadata such as the temperature at which the experiment as been conducted and more direct dry-lab data like the version number of the reference genome. This is of particular importance for reproducibility and helps to decide how to compare two different datasets. Providing specific metadata for the analysis and incorporating it is key not only to comply with a scientific rigor but also for interpretation.

Although Conda is the preferred method of installation, the Docker alternative ended up being more appropriate for the shared environment where the tests were run. Pulling and executing its image opens a R-studio server, where the client can have easy access to all the tools and pipeline scripts.

Under the *pipelines* folder, a couple of files can be found that contain example applications, as well as some annotations. These files are completely configurable and customizable to the experimental needs. Due to its modularity, the user can then skip some of the steps, add new samples without the need of rerunning some of the most time-consuming or add external software. Also, the pipeline offers the option to be run with multiple cores/threads by just changing one variable ("MAX_CPUS"). The pre-processing steps of native R computation can get some speedup thanks to its Rmpi⁴ package support. This is specially attractive for cluster computing.

Another suitable feature is that you can directly download the raw data from [Gene Expression Omnibus \(GEO\)](#)⁵ given an its specific accession number. It also automatically extracts its metadata, retrieving information like applied protocols in the reads preparation or sequencing methods. However, this data has to be converted from its [Sequence Read Archive \(SRA\)](#) format to the FASTQ format. This can be done with the NCBI's SRA toolkit, also included in the Docker image. Besides, users can just input FASTQ files locally if there's no need to download them. The next steps and corresponding tools chosen to execute for this thesis were:

1. **FastQC.** For the actual data pre-processing, a quality check of raw read data is executed.

To run FastQC, only the paths to the FastQ files must be supplied to its wrapper function.

```

1 fq <- system.file("extdata", "my.fastq.gz", package = "Geo2RNAseq")
2
3 rawQualDir <- file.path(outDir, "quality", "raw")
4

```

⁴<https://cran.r-project.org/web/packages/Rmpi/index.html>

⁵<https://www.ncbi.nlm.nih.gov/geo>


```
5 fq_raw_res <- run_FastQC(files = fq, outDir = rawQualDir, cpus = 2,
  workers = 2)
```

2. **Trimmomatic.** In this steps, the removal of adapter or low quality sequences is performed. Leading and trailing quality bases of the read are removed. Also, we are given the option of computing the average quality of a sliding window. The whole window sequence is cut if the quality falls under a given threshold. Also, minimal sequences of under 30 nucleotides are completely removed to avoid possible bias.

```
1 trimDir <- file.path(outDir, "fastq")
2 trim_res <- run_Trimmomatic(fq, outDir = trimDir, is.paired =
  FALSE, compress = TRUE, cpus = 2, workers = 2)
```

3. **SortMeRNA.** This tool is needed in order to filter rRNA fragments, abundant throughout the sample. Its wrapper function only requires the reference file that matches to the sets of input reads and optional paralelization parameters.
4. **TopHat2** The mapping step is critical for the analysis. Therefore, GEO2RNA-seq allows the user to include several cutting-edge tools such as STAR. However, it was decided to use one of the officially supported tools included in the Docker image for simplicity. The reference genome FASTA file of the organism of interest is needed. Besides, it is recommended to include additional annotation files or the index to ease the process. If the latter can't be found, this package contains a wrapper function to create one.

```
1 tophat_index <- make_Tophat_index(genomeFile = genome)
2 topDir <- file.path(outDir, "mapping", "tophat")
3 map_tophat_res <- run_Tophat(
4   files      = fq,
5   index      = map_index,
6   outDir     = topDir,
7   is.paired  = FALSE,
8   cpus       = 2,
9   workers   = 2
10  )
```

5. **FeatureCounts** For quantifying the number of reads assigned to each feature in the reference genome, the [Gene Transfer Format \(GTF\)](#) file containing the annotations must be supplied. This file must imperatively contain the `gene_type` and `feature_type` columns.

```
1 bamFiles <- map_hisat_res$files
2 gene_type <- "ID"
3 feature_type <- "sequence_feature"
4 count_res <- run_featureCounts(
```

```

5   files      = bamFiles,
6   annotation = file.path(outDir, "generatedGTF.gtf"),
7   outDir     = file.path(outDir, "counting"),
8   featureType = feature_type,
9   IDtype     = gene_type,
10  isPairedEnd = FALSE,
11  cpus       = 2,
12  workers    = 2
13 )

```

6. **MultiQC**. Finally, this tool is included to generate report files. MultiQC supports several of the best known RNA-seq analysis applications. It searches their output directory and subdirectories for their respective report files to combine them into a single [HyperText Markup Language \(HTML\)](#) interactive report.

To ensure that execution times are kept to a minimum, GEO2RNAseq takes advantage of parallelization —i.e., [multi-threading](#) and concurrent execution—. Although native high-throughput sequencing data functions are optimized in R, the most critical steps are handled by third-party software, so it's up to them to achieve low run-times and accuracy in the results. These critical steps are parallelized with the execution of any intermediate R code to reduce total runtime drastically.

Subsequent steps of the downstream analysis such as differential expression or visualization of the results were not run, as they are mainly focused for researchers and are natively integrated in R due to its statistical nature and plotting capabilities.

4.1.2 nfcore/RNA-seq

nf-core/RNA-seq⁶ shows itself to be yet another very interesting and valid option for differential expression analysis. This bioinformatics pipeline is implemented using Nextflow⁷, a workflow tool based on the dataflow programming model which ultimately simplifies writing complex distributed pipelines[10]. It uses Docker or Singularity containers for its portability. Also, thanks to its Amazon Web Service⁸ support, the pipeline is ready to run real-world datasets on cloud infrastructure and provides persistent storage for results to be easily compared with other workflow results from other pipelines.

The pipeline itself contains mainly the same steps and tools used in the GEO2RNA-SEQ package, with some additional inclusions. Unlike the previous option, this application hardly allows you to include external software or make major modifications.

⁶<https://nf-co.re/rnaseq>

⁷<https://www.nextflow.io/>

⁸<https://aws.amazon.com/>

```
sample,fastq_1,fastq_2,strandedness
TFG_TEST1,GSM2072350_R1_001.fastq.gz,GSM2072350_R2_001.fastq.gz,unstranded
TFG_TEST1,GSM2072351_R1_001.fastq.gz,GSM2072351_R2_001.fastq.gz,unstranded
```

Figure 4.1: Inclusion of several re-sequenced reads for depth coverage on a samplesheet

Contrary to how the previous example was able to download input data and its metadata given an identifier, all files must be findable and accessible locally. For the tool to use them, a samplesheet must be created with information about them. This samplesheet must contain the sample name, both FASTQ files if they are paired-end data and the strandedness. In the case of having several re-sequenced samples just to increase sequencing depth, all entries must share the same sample identifier. This tool will automatically concatenate them before continuing with the analysis.

This time, the pre-processing stage consists of:

1. **FastQC and SortMeRNA.** Much like in the GEO2RNA-seq package, quality control and removal of ribosomal RNA is carried out by the same tools.
2. **Trim Galore.** For adapter and quality trimming, a different tool is used. Trim Galore is known for its better reads pair-awareness.
3. **UMI-TOOLS.** An extra tool is included to deal with Unique Molecular Identifiers — i.e., extra indices added to sequencing libraries that enable better quantification and removal of duplicates—.

Several options are available for the alignment and quantification steps. By default, STAR is used to map the reads to the reference genome, followed by a BAM-level quantification with Salmon or RSEM. HISAT2 is also available for alignment, preferably in low-memory environments, but no quantification is supported for that tool. Besides, Salmon can be run at the same time as the standard alignment process for a pseudo-alignment and quantification workflow, as it provides extra [quality control \(QC\)](#) metrics. The minimal reference genome files needed for this step are a FASTA and its corresponding annotation file. Every other file needed, such as the index, can be generated from those. However, it is preferable to have them already stored locally, as it would improve execution-times drastically.

Post-processing is comprised of several steps. Most of these are also available for the GEO2RNA-SEQ bundle but, given the possibility of avoiding them, it was chosen not to execute them.

1. **SAMTOOLS.** This is the go-to application for reading, writing, editing or indexing [Sequence Alignment Map \(SAM\)](#) and BAM format files. In this specific step, it is used

to sort and index the alignments. This process is not mandatory for the analysis, but it is recommended as it leverages on the fact that it is not very time consuming and helps the following tools to run faster.

2. **UMI-TOOLS**. Similar to how FastQC can be run after any other tool to ensure the quality of the reads after processing, this software is run again to accommodate for UMI-based deduplication.
3. **MarkDuplicates**. Included in the Picard suite, this is one of the most well known tools in today's analysis. It locates and tags reads duplicated reads that originated from a single fragment of RNA.
4. **StringTie**. Used for transcript assembly and quantification. Although a quantification step is performed by Salmon, it is not executed in the HISAT workflow due to incompatibility.
5. **BEDtools and bedGraphToBigWig**. They are used for generating bigWig coverage files. These come in handy for visualizing continuous data.
6. **Quality Control**. Several quality control tools are executed, including plots and specific data used for the downstream scientific interpretation.
7. **MultiQC**. Again, MultiQC is used to group all reporting output data of the previous tools into a HTML file.

For its execution, Nextflow allows the user to choose different configuration profiles. These profiles instruct the pipeline to use their respective container technology to bundle the software —e.g., Docker, Singularity—. Besides, Nextflow also loads different configurations depending on whether we are executing it from an institutional cluster or not. These configurations are dynamically loaded from a repository at runtime if they are available for that system. When running the pipeline under any profile, the code will be pulled from GitHub and will be cached. If available, this cached version will be used even if it is outdated.

By default, each step has a predefined set of requirements such as number of cores, memory and maximum time but these can easily produce errors if they are not properly met due to limits in infrastructure. The pipeline execution will stop if any of the jobs fail and it's up to the user to customize the requirements. Custom configurations can be introduced in the pipeline to account for that. These configurations not only apply to the processes but they can also be used to input non mandatory tool-specific parameters.

4.2 Best practices variant calling workflow

To further delve into some of the cutting-edge workflow designs, research on GATK's best practices was carried out. As previously discussed, these approaches focus on the RNA-seq data analysis for variant calling. This results in a change of subject, as previously discussed methods regard differential expression analysis and have no native support for the detection of indels and SNPs. GATK's RNA-seq short variant discovery best practices workflow provides the minimum steps necessary to achieve not only accurate results, but to ensure maximum computational efficiency and reproducibility.

Regarding the pipeline and unlike previous approaches, no quality control steps are required prior to the alignment. This is mainly due to the pipeline only accepting unmapped BAM (uBAM) format files as input. This is a variation of normal BAM files, where raw reads contain no mapping information. It is considered an "off-label" use, as gathering alignment information is the whole point of these files. However, there are many benefits when it comes to storing additional metadata of the reads and using this information to help subsequent tools work optimally.

For further context, some of the major shortcomings of the FASTQ files should be discussed. The main problems regard metadata conventions for paired end reads. Also, there is this situation where some tools want a single FASTQ file containing both the forward and reversed reads interleaved, similar to how the nfc core tool concatenated the samplesheet files. Nevertheless, other tools prefer them to be completely separated. Finally, there are many ways to manage the inherent quality scores of the reads, as some protocols handle these as they fit. GATK tends to use this format, even when it results in slightly more complex workflows, but the documentation reflects the use of FASTQ files in order to please novice researchers.

Continuing with the workflow, STAR is the recommended option for alignment, as it maintains a special sensitivity for indels, beneficial in the downstream analysis. This first processing step is designed to map read pairs individually. This isolation is computationally beneficial, as it can be massively parallelized to increase throughput.

The next step involves Picard's MarkDuplicates tool. Here, read pairs that are likely duplicated due to previous processes are all tagged but one. This read pair becomes the representative, and all other are ignored during the variant calling process. Sorting of the alignments is also performed in this step. Additionally, the SplitNCigar tool is run to help reformat the alignments that would cause problems in the variant calling step. This also entails the reassignment of mapping quality scores to match DNA conventions.

Systematic errors in the base quality scores are probably present at this point in the analysis. To correct them, a base quality recalibration is performed using machine learning al-

gorithms to detect patterns for these errors. A statistical model from the collection of biases from all base calls of the dataset is built and adjustments are performed based on it. The recollection of statistics can be parallelized by spreading the resources throughout different coordinates of the genome. After this, you get a file ready for analysis. The variant calling is then performed by GATK's HaplotypeCaller.

4.2.1 Pipeline implementation

But how does GATK implement its workflows? The preferred pipelining solution involves the creation of WDL⁹ files and Cromwell¹⁰ as its execution engine. Repositories containing these workflows are available to the user and are maintained by its git organization. Intel also contributes creating optimized versions with the objective of increasing performance and achieving low execution times.

WDL is an open source scripting language whose objective is to help the user create human-readable genomic processing workflows. The main components forming the structure of the script are **workflow**, **task**, **call**, **command** and **output**. Additional components also exist and are optional. They are usually employed to specify environment conditions, runtime parameters and scripting metadata.

The top level is composed by the workflow component. One can look at it as the *main* function of a program written in C. It houses *call* statements, as well as workflow-level input variables —i.e., call to functions and global variables—. These call components specify that a particular task is to be executed. The simplest call just needs a task name, but the user can also add a code block specifying input variables. In addition, there is the possibility of calling an *alias*, that is, run the same task multiple times within the same workflow but with a different input setup to reuse code and help maintenance. Here's an example code:

```
1 workflow RNAseq {
2
3   File inputBam
4   String sampleName = basename(inputBam, ".bam")
5
6   File refFasta
7   File refFastaIndex
8   File refDict
9   .
10  .
11  .
12  call StarAlign {
13  input:
```

⁹<https://openwdl.org>

¹⁰<https://cromwell.readthedocs.io/en/stable/>

```

14     star_genome_refs_zipped = starReferences,
15     fastq1 = SamToFastq.fastq1,
16     fastq2 = SamToFastq.fastq2,
17     base_name = sampleName + ".star",
18     read_length = readLength,
19     preemptible_count = preemptible_count,
20     docker = star_docker
21 }
22
23 call SamToFastq {
24 input:
25     unmapped_bam = RevertSam.output_bam,
26     base_name = sampleName,
27     preemptible_count = preemptible_count,
28     docker = gatk4_docker,
29     gatk_path = gatk_path
30 }
31 }

```

One important thing to notice is that the order in which the statements are written does not define by any stretch of the imagination the order in which they are going to be executed. Instead, the executing engine is in charge of evaluating input and output relations to infer and create a dependency graph between tasks. In the example, we can easily check that the *StarAlign* task receives the output of the *SamToFastq* one, implying that the latter should be executed first, even if it appears later in the workflow block.

Outside of the workflow segment, the tasks are defined. They are called from within the workflow block –i.e., the main function–, which is what causes them to be executed. Here’s an example task:

```

1 task SamToFastq {
2
3     File unmapped_bam
4     String base_name
5     String gatk_path
6     String docker
7     Int preemptible_count
8
9     command <<<
10     ${gatk_path} \
11         SamToFastq \
12         --INPUT ${unmapped_bam} \
13         --VALIDATION_STRINGENCY SILENT \
14         --FASTQ ${base_name}.1.fastq.gz \
15         --SECOND_END_FASTQ ${base_name}.2.fastq.gz
16 >>>

```

```

17
18 output {
19     File fastq1 = "${base_name}.1.fastq.gz"
20     File fastq2 = "${base_name}.2.fastq.gz"
21 }
22
23 runtime {
24     docker: docker
25     memory: "4 GB"
26     disks: "local-disk " +
27     sub(((size(unmapped_bam, "GB")+1)*5), "\\..*", "") + " HDD"
28     preemptible: preemptible_count
29 }

```

Tasks can be considered as standard functions, with their respective input and output parameters. Thus, it is the functional component that contains the information of what is going to be done. This information is centered around the command component, which is required. Inside of it, the user can specify the literal command that she/he would run in a terminal shell, with the exception of the variable parts, otherwise substituted by a placeholder.

These placeholders have to be previously defined in its respective input definition segment. Thanks to them, the user can specify these values at runtime without having to modify the script, as hardcoding the variables would be a very inefficient way to deal with it. Checking the above example, a segment with the declaration —i.e., name and type— of the variables can be seen at the top of the task definition. These can now be introduced as placeholders by typing their name within curly braces prefaced by a dollar sign.

Besides, an output component is usually required in the definition of the tasks, similar to how a function would return a certain value. The explicit mention of these outputs, together with the inputs, is what guides the execution engine in the creation of the dependency graph. In the example, two output files will be produced when running this task, with a concatenation of the *base_name* string used as input and the *fastq.gz* file extension as name. Paying attention to detail, these *fastq1* and *fastq2* file variables are the ones used in the *call StarAlign* segment example as input, prefaced by the name of the task.

This chaining is what makes the creation of sophisticated pipelines possible. OpenWDL developers call this “plumbing”. Up until now, with all the information that has been covered, linear or very simple branching and merging workflows can be built, it’s a matter of chaining multiple inputs and outputs. However, in this very same variant calling workflow, several conditional statements are specified, depending on input data. These built-in features allow further sophistication and open paths to new possibilities.

Although not present in the pipeline, one of the most interesting and worth-mentioning

aspects is the scatter-gather parallelism feature, that, although similar, has some key differences with the technique called multi-threading. In the latter, a program can achieve very quick execution times at core and machine level, where a program sends several threads to the processor at the same time to get the work done. However, by running a program with a scatter-gather approach, cluster level parallelism can be achieved. This strategy involves the execution of several copies of the same program, independent among them, over a portion of the input data —i.e., the scattering—. The results are then merged —i.e., the gathering— as if it had just been one single command. In WDL, the scatter part must be explicitly defined with specific keywords, but the gather part is automatic or implicit.

4.2.2 WDL execution

WDL scripts are not executable by themselves, and need an execution engine to do so. Cromwell is an open-source workflow execution engine that supports, among other description languages, WDL. It's written in Java and can be run in a wide variety of platforms, including local machines, shared computer environments regulated by a job scheduler —e.g., SLURM—, or cloud platforms. Different versions of the Cromwell executables are available to the public in the Cromwell GitHub¹¹ repository as pre-compiled jar files. These must support the features of the WDL version specification.

Remember the *runtime* component found within the *task* block in the above examples? It can be used to customize runtime parameters such as memory limitations or number of tries before the workflow stops with an output error. It can also be manipulated to specify the runtime backend of use when executing the script. GATK's WDL files are, by default, customized to run on the Google Cloud Platform¹². This is because the Broad Institute leans more towards cloud-based computing environments. To run locally, the user is in charge of making the necessary changes in the configuration files. Cromwell provides users with a minimal "default" configuration file example that is intended to be edited with their respective needs.

For this specific project, several small changes had to be made in order to run the scripts on a cluster with a job scheduler and uDocker —more on that later—. Regarding the latter, Docker images are tagged with different versions —e.g., *image:latest* is referring to the latest tag of the image—. However and by default, Cromwell uses their hash specifier to pull and run them. This is because this strategy is consistent with Docker's best practices due to hashes being better version specifiers for those images. The problem arises when implementations such as uDocker don't support this feature and thus, has to be deactivated. Additionally, a uDocker command wrapper has to be specified for it to work locally:

¹¹ <https://github.com/broadinstitute/cromwell/>

¹² <https://cloud.google.com/>

```

1 docker {
2   hash-lookup {
3     enabled = false
4   }
5 }
6
7 backend {
8   #default = "Localudocker"
9   providers {
10    Localudocker {
11      config {
12        run-in-background = true
13        runtime-attributes = ""
14        String? docker
15        String? docker_user
16        ""
17        submit = "/usr/bin/env bash ${script}"
18
19        submit-docker = ""
20        udocker run \
21          --rm -i \
22          ${"--user " + docker_user} \
23          --entrypoint ${job_shell} \
24          -v ${cwd}:${docker_cwd} \
25          ${docker} ${docker_script}
26        ""
27      }
28    }
29  }
30 }

```

Therefore, this configuration file must be specified when executing the WDL file, together with a [JavaScript Object Notation \(JSON\)](#) file containing the inputs. This is the best way to customize and specify all the input values that are subject to change between run to run, instead of being hardcoded in the script. Cromwell will execute the workflow with these new batches of data, that will run through the pipeline. To facilitate the creation of this file, WDL comes with a package that parses the script to look for variables and generates a template JSON file containing them, as doing it by hand would be tedious and problematic. In any case, each WDL script found in GATK's repository comes with its respective JSON template file, already specifying testing sample data. These data are found in Google Buckets, so it is up to the user whether to change the inputs to some local ones or leave them as they are. Here is an example of the input file generated from the variant calling workflow WDL script. It not only contains all input variables specified in it, but also the runtime attributes for the

execution engine:

```

1 {
2   "##_COMMENT1": "Input",
3   "RNAseq.inputBam": "path/to/sample/data.bam",
4
5   "##_COMMENT2": "REFERENCE FILES",
6   "RNAseq.refFasta": "path/to/sample/data.fasta",
7   "RNAseq.refFastaIndex": "path/to/sample/data.fasta.fai",
8   "RNAseq.refDict": "path/to/sample/data.dict",
9
10  "##_COMMENT3": "INTERVALS",
11  "RNAseq.wgsCallingIntervallList":
12    "path/to/sample/data.interval_list",
13
14  "##_COMMENT4": "RESOURCE FILES",
15  "RNAseq.dbSnpVcf": "path/to/sample/data.vcf",
16  "RNAseq.dbSnpVcfIndex": "path/to/sample/data.vcf.idx",
17  "RNAseq.knownVcfs": [
18    "path/to/sample/data.sites.vcf",
19    "path/to/sample/data.known_indels.vcf"
20  ],
21  "RNAseq.knownVcfsIndices": [
22    "path/to/sample/data.sites.vcf.idx",
23    "path/to/sample/data.vcf.idx"
24  ],
25  "RNAseq.annotationsGTF": "path/to/sample/data.gtf",
26
27  "##_COMMENT4": "DOCKERS",
28  "#RNAseq.gatk4_docker_override": "String? (optional)",
29  "#RNAseq.star_docker_override": "String? (optional)",
30  "#RNAseq.gitc_docker_override": "String? (optional)",
31
32  "##_COMMENT5": "PATHS",
33  "#RNAseq.gatk_path_override": "path/to/gatk/gatk",
34
35  "##_COMMENT6": "PREEMPTIBLES",
36  "##RNAseq.preemptible_tries": "(optional) Int?",
37
38  "##_COMMENT7": "Misc",
39  "#RNAseq.StarAlign.num_threads": "(optional) Int?",
40  "#RNAseq.StarAlign.star_limitOutSJcollapsed": "(optional) Int?",
41  "RNAseq.StarAlign.additional_disk": "50",
42  "#RNAseq.StarAlign.star_mem_max_gb": "(optional) Int?",
43  "RNAseq.StarGenerateReferences.addtional_disk": 50,
44  "#RNAseq.StarGenerateReferences.num_threads": "(optional) Int?",
45  "#RNAseq.StarGenerateReferences.mem_gb": "(optional) Int?",

```

```

45 "#RNAseq.haplotypeScatterCount": "(optional) Int?",
46 "#RNAseq.use_gatk4_for_all_tools": "(optional) Boolean",
47 "#RNAseq.minConfidenceForVariantCalling": "(optional) Int?",
48 "#RNAseq.zippedStarReferences": "(optional) File?",
49 "#RNAseq.readLength": "(optional) Int?"
50 }

```

Cromwell will print out logs containing information about the current status of the workflow and, once it completes, it will specify the location of all the output files generated by the pipeline.

4.2.3 GPU-based workflow

So far, only workflows that are executed by the CPU have been discussed. As previously mentioned, Nvidia Clara Parabricks offers GPU-accelerated solutions for genomics. It is very easy to start using it, as downloading and installing it only takes minutes. The package uses a Python application wrapper to execute the workflows, so Python and nvidia-drivers supporting CUDA architecture must be up to date. As powerful of a tool as it is, the user must request a trial access, as the suite only works under licence. Parabricks is a containerized software, and it supports Docker and Singularity solutions that are handled automatically by the software and installer. Again, small modifications had to be made in order to wrap all Docker commands with the udocker alternative.

Parabricks can only run on machines that have Nvidia GPUs available. Both the installer and the executing of workflows check for them by launching the command `nvidia-smi`, that lists all GPUs available.

To test not only the GPU-accelerated pipelines, a couple of tests were run regarding its standalone tools. The RNA FQ2BAM is dedicated to transforming FASTQ input files into analysis-ready BAM files that can be used in the subsequent variant calling process. It is not but a sequential execution of the three corresponding CPU-based applications:

```

1 #####
2 # Parabricks fq2bam command #
3 #####
4 $ pbrun rna_fq2bam --in-fq sample_X_1.fq.gz sample_X_2.fq.gz
   --genome-lib-dir HG38 --output-dir sample_X/ --ref ref.fasta
5
6 #####
7 # CPU-based counterpart #
8 #####
9 #STAR Alignment
10 $ ./STAR --genomeDir HG38 --readFilesIn sample_X_1.fq.gz
   sample_X_2.fq.gz --outFileNamePrefix sample_X/ --outSAMtype BAM
   SortedByCoordinate

```

```
11 |
12 | #Coordinate Sorting
13 | $ gatk SortSam --java-options -Xmx30g --MAX_RECORDS_IN_RAM=5000000
    |   -I=Aligned.sortedByCoord.out.bam \ -O=cpu.bam
    |   --SORT_ORDER=coordinate --TMP_DIR=/raid/myrun
14 |
15 | # Mark Duplicates
16 | $ gatk MarkDuplicates --java-options -Xmx30g -I=cpu.bam
    |   -O=mark_dups_cpu.bam -M=metrics.txt --TMP_DIR=/raid/myrun
```

These two commands result in the same exact output. To compare them, the BamUtil diff tool can be used, and no error report should result.

Finally, to run the GATK's best practices variant calling workflow, a simple run of Parabricks's RNA pipeline is enough. Again, all genome reference files, including indices and annotation files, must be included in the execution. However, Nvidia's approach utilizes raw FASTQ files instead of the uBAM files that characterized the GATK's pipeline. Thus, when testing with the same dataset, although redundant, uBAM files must be converted beforehand into FASTQ files or viceversa.

Once again, matching outputs can be compared to check for sensitivity. This time and similar to the BAM comparison, the BQSR report generated should be exactly the same. However, they may be some inconsistencies in the VCF report, since the sensitivity and the specificity is at its maximum in this step.

4.3 Nextflow pipeline

We have talked about the different alternatives that exist today, both for CPU and GPU based workflows. Some of the technologies used in the implementation of these pipelines have also been discussed. In this chapter, the development of a proprietary pipeline will be carried out following GATK's best practices, with some additional quality control tools. This is due to the lack of this kind of applications in previous alternatives, as GATK seems to take for granted the quality of the raw reads, thus avoiding extra computational workloads that could slow down the execution of the pipelines.

Nextflow was chosen as the workflow developing framework for this duty, as it benefits from several key features:

1. **Ease of use.** Nextflow requires no installation, as the user can easily download any executable packages needed. It makes use of a programming domain-specific language that aims to ease the writing of sophisticated pipelines. Besides, Nextflow is centered around the idea of Linux providing simple command-line tools that can be chained together to create complex workflows. This extension of the Unix pipes model provides

extra functionality to the standard one-way flow of data. The dataflow programming paradigm employed allows the user to define complex program interactions and the abstraction of the parallel environment.

2. **Container support and pipeline sharing.** Of course, container technologies are supported for analysis reproducibility purposes, specifically Docker and Singularity. Also, Nextflow introduces an abstraction layer that separates the pipeline's logic and the actual execution of the script, fitted to the needs of the platform in use. Besides, thanks to its seamless integration with repository platforms such as GitHub, Nextflow can look for scripts in public repositories. These files are then downloaded and executed just by specifying its *qualified name* formed by the owner's name and repository name separated by a slash ("/") character —e.g., `santiagomillan/tfg`—.
3. **Implicit parallelism.** Parallelization is implicit thanks to its dataflow programming model. Similar to how WDL and the execution engine managed it, it is defined by the input and outputs of the processes defined within the script. Nextflow is then in charge of managing resources and creating the dependency graphs needed to run tasks with inherent parallelism.
4. **Several language support.** The user is not limited to BASH scripts commands within the processes. In the workflow definition, a mix of different scripting languages can be put together, meaning that for each task definition you are free to use the specific language that suits you best or that better fits the needs of the process declaration.

4.3.1 Nextflow script

It's time to talk about the making of the script itself. The Nextflow scripting language syntax is based on Groovy¹³, an object-oriented programming language implemented over the Java platform for the [Java Virtual Machine](#). Thus, Nextflow can make use of any library or piece of Groovy code. This includes the ability to declare variables, lists, maps, apply conditional statements or use regular expressions.

Similar to how WDL used *tasks* to define the most basic processing unit that executes commands, in Nextflow it comes by the hand of *processes*. These must contain the strings that represent the pieces of code to execute within its body. Additionally, the user can define up to four more blocks, containing inputs, outputs, *when* clauses and directives.

Starting with the latter, directives are declarations that provide customization to the execution of the process they appear in. They must be specified at the top of the process block and usually depend on the executor —i.e., the system where the actual pipeline is executed

¹³ <https://www.groovy-lang.org>

and supervised—. Some of the most commonly used are: *CPU*, to define the number of logical cores that are accessible by the process; *containerOptions*, specifying container-supported options of the underlying execution engine; *maxErrors* and *maxRetries*, to declare how many times the process can fail before the execution stops; and *cache*, to allow the storage of results in a local cache to speedup subsequent executions under the same data or to abuse the resume option of the workflow.

The when clause allows the user to sophisticate the pipeline by introducing conditional statements. These make the execution of the process subject to a given condition, enabling or disabling it depending on the state of the instance.

The script block itself is formed by a string that is to be executed by the process. It is essentially what the user would use in a terminal shell or Bash script, limited by the execution environment at stake. Like in the WDL example, one can introduce placeholders within the command to account for variables or even use system environment variables that would not normally be accessible due to the abstraction layer of the executor. Although considered as Bash scripts by default, the commands can be written in any other scripting language, such as Python or R.

Before talking about the input and output blocks, a brief mention should be made about the concept of channels. As processes are isolated from one another and have no shared memory, they must communicate among them sending values through channels. The *sending* operation is asynchronous. That is, the operation is executed without a predictable time and the process does not need to wait for the receiving one to continue its execution. However, the *receiving* operation does stop and can't continue its execution without the arrival of the message. There are two types of channels:

1. **Value channels:** also known as a singleton channel, regulates a single value that can be read any times necessary without consuming its content. The creation of a value channel is implicit due to several cases involving the retrieval of a single piece of data without specifying a queue-managing factory method —e.g., the method *count* retrieves a single value, thus implicitly creating a value channel—. Also, a process creates a value channel as its output when only value channels are defined as input.
2. **Queue channels:** conversely, a factory method such as *from* or *into*, that would retrieve or place values among others, would result in the implicit declaration of queue channels. These are non-blocking unidirectional "first in, first out" queues that can store fleeting values —i.e., can be consumed—.

Having sorted that one out, the comprehension of the input and output blocks is trivial. With them, the user can define what the process is expecting from its peers and what it offers to the other processes. Both the input and output definition start with a qualifier —e.g., *val*,

path, file, stdin— that defines the type of data that is being handled. Next, the name of the variable. Finally, the definition of the channel over which the data is received or sent.

Knowing all of this, a similar pipeline to that of the GATK's best practices for pre-processing workflow can be recreated¹⁴, with the inclusion of MultiQC to retrieve some statistics and FastQC to check for the quality of the raw reads. Also, contrary to the use of uBAM files as input on the GATK's pipeline, a simpler approach was chosen and the input is comprised directly of the FASTQ files. Everything else is exactly the same.

For this pipeline, several channels were created to chain the processes and avoid unnecessary **Input/Output (IO)** operations, achieving better execution times:

```

1 Channel
2   .fromFilePairs( params.reads )
3   .ifEmpty { error "Cannot find any reads matching:
4     ${params.reads}" }
5   .into { read_pairs_ch; read_pairs_fastqc }
6 process fastqc {
7   tag "$pair_id"
8
9   input:
10  set val(name), file(reads) from read_pairs_fastqc
11
12  output:
13  file "*_fastqc.{zip,html}" into fastqc_results
14
15  script:
16  """
17  fastqc -q $reads
18  """
19 }
20
21 process buildIndex {
22   tag "$genome.baseName"
23   publishDir params.outdir, mode: 'copy'
24
25   input:
26   path genome from params.genome
27   path annot from params.annot
28
29   output:
30   path 'g_index' into index_ch
31   file "g_index/*out" into starindex_results

```

¹⁴ This pipeline can be found in this project's repository: https://github.com/santimillang/tfg_pipeline


```

32
33     """
34     mkdir g_index
35     STAR --runThreadN 2 --runMode genomeGenerate
36     --sjdbGTFtagExonParentTranscript Parent --sjdbGTFfile ${annot}
37     --genomeDir g_index --genomeFastaFiles ${genome}
38     """
39 }

```

In this specific example, we can clearly see that two channels are created in the beginning of the workflow to pipe the input FASTQ files directly into the first two processes. As these processes have no relation whatsoever and having two queues avoids the consumption of values, these two tasks can be executed in parallel without issues.

Finally, regarding the execution of the scripts, a configuration file can be supplied to account for any specific settings the user may want to apply at runtime. These include the configuration of the executors. By default, the local one is used. It parallelizes the execution of the pipeline by running multiple threads leveraging on the multi-core architectures the environment may have. In this case, the configuration file was mainly used to specify the container that would store all the tools:

```

1 process {
2   container = 'santiagomillan/tfg_pipeline@sha256:...'
3 }

```

4.3.2 Dockerfile

A brief introduction to how all the tools that are included in this pipeline are containerized¹⁵ will be given. Following best practices for writing Dockerfiles, a series of guidelines were taken into account when creating it.

First, the base image used for the creation of the container was *ubuntu:20.04*. The specification of the tag makes it more stable, as a *latest* tag would end up generating problems due to the appearance of newer versions of the base image. Also, the option of using multi-stage builds was considered, as it reduces the size of the final image by quite the amount. However, for simplicity, the focus was on the reduction of intermediate layers and leveraging on build cache. This is doable by ordering the commands —i.e., layers— from less frequently modified to more frequently changed. This way, it's less probable that a minimal change in the bottom layers can affect the top ones, which would entail the re-build of the entire image. Besides, the installation of "just in case" packages was meticulously avoided to reduce complexity and, mainly, image size.

¹⁵https://hub.docker.com/repository/docker/santiagomillan/tfg_pipeline

One of the most important decisions was to choose whether or not each application of the pipeline should be executed in a completely dedicated container. Similar to how GATK handled its containers, this would mean that, for each step, an image containing only the software needed for that tool to run would be executed. Nextflow supports this feature, allowing the user to specify the image that has to be pulled and executed within the block of the process, thanks to one of the directives. This decoupling of applications is actually encouraged not only by Nextflow, but by the guidelines for writing good Dockerfiles. This approach has the advantage of facilitating the scalability of the tool by adding more machines to the pool of resources –i.e., horizontal scalability– and modularity, both tenets of software development. However, with modularity comes the drawback of having to design a more sophisticated amalgam of containers and dependencies between them, thus causing this project to be more enclined for a simpler Dockerfile that encompasses all tools needed within a single container.

For this specific case, the order of the layers are as followed:

1. The specification of the base image using the *FROM* clause, that pulls the image from a public repository. By default, this repository is *Docker Hub*.
2. The installation of pre-requisites comes by the hand of the *RUN* instruction. With it, all software needed was found in the Ubuntu package repository and installed using the Ubuntu packet manager *apt*. Following best practices, previous to the installation of the tools, an update on the list of available packages and versions has to be carried out. As the image building requires no user interaction, all the prompting that pops up during the installation has to be taken care of including the "assume-yes" option.
3. All the installation steps needed for the tools were carried out using a single *RUN* instruction, including the download, decompression and creation of the symbolic links necessary as life-quality measures.

```
1 FROM ubuntu:20.04
2
3 LABEL maintainer = "Santiago Millan Gonzalez
4     <santiago.millang@udc.es>"
5 #
6 # Install pre-requistes
7 #
8
9 RUN apt-get update --fix-missing && \
10 apt-get install -q -y samtools python && \
11 apt-get install -y wget && \
12 apt-get install -y unzip && \
13 apt-get install -y python3-pip && \
```

```
14 apt-get install -y openjdk-8-jre && \  
15 apt-get install -y fastqc  
16  
17 #  
18 # RNA-Seq tools  
19 #  
20  
21 RUN wget -q https://github.com/alexdobin/STAR/archive/2.7.9a.tar.gz  
    -O- \  
22 | tar -xz -C /opt/ && \  
23 ln -s /opt/STAR-2.7.9a/ /opt/star  
24  
25 RUN wget -q -O gatk.zip  
    https://github.com/broadinstitute/gatk/releases/download/4.2.2.0/  
26 gatk-4.2.2.0.zip && \  
27 unzip gatk.zip -d /opt/ && \  
28 rm gatk.zip  
29  
30 RUN \  
31 pip3 install multiqc  
32  
33 #  
34 # Finalize environment  
35 #  
36  
37 ENV PATH=$PATH:/opt/star/bin/Linux_x86_64:/opt/gatk-4.2.2.0
```


Chapter 5

Results

Having disclosed the different options considered for testing, the results will be presented in this chapter, along with a description of the environment in which the tests were performed and the metrics used in the comparison.

5.1 Evaluation

It is important to make a small exposition about the execution environment and how tests have been carried out before showing the results themselves.

5.1.1 Cluster environment

To carry out these tests and to elaborate on the results, the CITIC's¹ cluster *Pluton*² was used as the computing infrastructure and environment. *Pluton* is a heterogeneous cluster used mainly for High Performance Computing and has been receiving minor upgrades over the years, thus consolidating itself as a no-compromise supercomputer.

The cluster is composed of what is called a *frontend* node and several *backend* nodes. This frontend node functions as the only point accessible from the outside, so that end-users must exclusively connect to it remotely. Additionally, it works as a setup environment, where one can revise and compile his/her code and send a so-called job via scheduler or queue management system for its subsequent execution on the backend nodes. This way, a job can never be run on the frontend node.

As a final feature, this node also works as a [Network Attached Storage \(NAS\)](#) server, so that all user files are stored there physically, functioning as permanent storage. These files are then accessible from the compute nodes via [Network File System \(NFS\)](#) protocol under

¹<https://citic.udc.es/>

²More can be found in Pluton's user guide: <http://pluton.dec.udc.es/guide/user-guide.pdf>

| | compute-2-{0-5} |
|-------------------------|---|
| CPU Model | 2 × Intel Xeon Silver 4216 Cascade Lake-SP |
| CPU Speed/Turbo | 2.1 GHz/3.2 GHz |
| #Cores per CPU | 16 |
| #Threads per core | 2 |
| #Cores/threads per node | 32/64 |
| Cache L1/L2/L3 | 32 KB/1 MB/22 MB |
| RAM Memory | 256 GB DDR4 2933 Mhz |
| Storage | 1 × SDD 240 GB SATA3 1 × HDD 2 TB SATA3 7.2K rpm |
| Accelerators | 2 × NVIDIA Tesla T4 16 GB GDDR6 (2-0) |
| Network interfaces | InfiniBand EDR and Gigabit Ethernet |

Table 5.1: Compute node description within rack 2

the **InfiniBand (IB)** network. Each node has its high-performance interconnection network (low latency and high bandwidth) with their corresponding IB specifications. These specifications determine the maximum bandwidth the network allows, ranging from SDR to HDR. Specifically, backend nodes are equipped with FDR or EDR network interfaces, allowing 56 Gbps bandwidths and 1-2 μ s latencies.

Backend nodes are the ones that host the computational resources, including accelerators such as GPUs. They are grouped logically in computing racks as follows:

1. **Rack 0:** 272 physical cores (544 threads) distributed in 17 compute nodes, 1088 GB of memory, 17 NVIDIA Tesla GPU (Kepler architecture) and 3 manycore processors Intel Xeon Phi.
2. **Rack 1:** 48 physical cores (96 threads) distributed in 17 compute nodes and 256 GB of memory.
3. **Rack 2:** 192 physical cores (384 threads) distributed in 6 compute nodes, 1536 GB of memory and 2 NVIDIA Tesla GPU (Turing architecture).

In summary, the cluster is made up of 25 compute nodes for a total of 512 physical cores (1024 threads), 2.8 TiB of memory, 19 Nvidia Tesla GPU accelerators and 3 Intel Xeon Phi manycore accelerators.

The Rack 0 is arguably the most powerful rack in the cluster and the one that was used for this project. As seen in table 5.1, it contains 2 NVIDIA Tesla T4 GPUs. These NVIDIA series are mainly focused in the field of high performance computing, deep learning and artificial intelligence. They feature configurations that offer higher computational power working with

datasets that are frequent in the scientific field. This means that they offer higher accuracy when managing 64-bit floating point (double precision) operations, have larger amounts of dedicated memory and come with error correction technology (ECC), which is key to perform large simulations and comparative tasks with full validity. They also offer high scalability and, of course, have no video output. These accelerators are a perfect match for the Nvidia Clara Parabricks' requirements, as their large dedicated memory are a must for good pipeline executions with no interruptions.

The software chosen for managing the queue system is Slurm Workload Manager³. It is in charge of distributing the user's jobs and fetching all the resources required for their execution. Thus, when deemed appropriate by the manager (depending on system load, number of users, total number of jobs...), the job will be executed at the compute nodes.

There are two types of jobs the user can execute:

1. **Batch.** These are simply a series of commands specified within a script file that are sent to Slurm for its execution using the command `sbatch`:

```
1 #!/bin/bash
2 #
3 # SBATCH -J GATKtest
4 # SBATCH -o %x_%j.out
5 #
6 date
7 sleep 20
8 date
```

To obtain information about this job status, the command `squeue` can be run. Its output contains, among other data, the job identifier and the current status in queue. The job ID is used in the previous example to name the output file, concatenated with the string "GATKtest", in this case. Batch jobs are also suited for parallel applications—e.g., OpenMP and MPI—.

2. **Interactive.** Also known as "interactive session", consists of making use of the corresponding terminal of a computational node to launch jobs directly on it. The output is obtained from the terminal itself. To do so, the user must use the `srun` command. This is specially useful for the execution of dockerized applications that make use of GPUs and make sure that the Docker image and the host where it is being executed have matching versions of nvidia-drivers.

Also, there are several parameters the jobs require in order to be processed in the most efficient way by Slurm. Not all parameters are mandatory and they can be introduced both in the script or on the command line when running `sbatch` and `srun`. Some of them are:

³<https://slurm.schedmd.com/documentation.html>

1. **Time.** All users are required to use this parameter when executing their jobs. It is an estimate of the maximum work duration and, if exceeded, the job will be forcibly terminated. Calculating a good estimate can help Slurm plan the execution of these in a more efficient way, prioritizing shorter jobs that require less resources instead of waiting for larger ones.
2. **Number of tasks and cores per task.** Pretty self-explanatory, these parameters allow the user to choose the maximum number of tasks that are going to be executed and how many cores per process are going to be needed. By default, both parameters assign 1.
3. **Memory.** Allows us to specify the memory required per computational node.
4. **Exclusive access.** This way, a backend node can be reserved exclusively for the job, not allowing any other user to interfere. This is specially attractive when benchmarking, since no external meddling can happen that may alter the measurements.
5. **Special resources.** Such as GPUs, allows the user to specify the need for a job to have a GPU available.

Finally, it is worth mentioning that Pluton uses the Lmod⁴ tool, an enhanced version of Environment Modules, to manage almost all software in order to support different versions of the same package/library/application. This way, users can change versions without having to explicitly specify different paths, minimizing dependency management. One of the most interesting features of this tool is the hierarchical management of dependencies approach. This way, when loading a module, only modules that depend on its version can be subsequently loaded, easing the environment management. Some of the most used modules in this project were the Java Runtime Environment and uDocker.

5.1.2 Input datasets and methodology

For these particular experiments, the polyA mRNA RNA-seq data from the continuous cell line K-562⁵ was selected[11]. Samples come from the pleural effusion of a 53-year-old female with chronic myelogenous leukemia in terminal blast crises. These are stranded from independent growths of the aforementioned cell line via the Illumina Hi-Seq RNA-Seq libraries from rRNA-depleted Poly-A+ RNA > 200 nucleotides in size. There are two replicates available for depth coverage, as seen in figure 5.2.

For the reference genome, the Genome Reference Consortium Human Build 38 (GRCh38)⁶ was used in its 13th patch release —i.e., the latest—. Successive versions of these assemblies

⁴<https://lmod.readthedocs.io/en/latest/>

⁵<https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE78557>

⁶https://www.ncbi.nlm.nih.gov/assembly/GCF_000001405.39/

have been published over the last few years, consequently bringing improvements in the quality of the build. The latest version of the human reference genome has meant a major refinement in the representation of the alternate haplotypes, specially interesting for the variant calling workflow accuracy.

Besides the main FASTA file containing the assembly, several supporting files such as the index and genome annotation files were also used.

Regarding the methodology employed for this experiment, several tests were run with the execution of the state-of-the-art tools previously discussed to gain some trivial insight. This also entails a thorough documentary analysis of the tools that are being used. Both example pipelines were run with their default configurations, such as the number of threads and maximum memory cap, as they are already optimized according to their creators. To evaluate their performance, the execution time of each tool and memory consumption was recorded along with the impact of the input/output operations for the applications that are not piped.

Finally, for the GPU vs CPU comparison, a series of test were executed using 1, 2, 4, 8, 16 and 32 CPU cores for the GATK version and up to 2 NVIDIA Tesla T4 accelerators for the GPU counterpart. Speedup, efficiency and scalability data will be presented for these workflows, as well as accuracy results.

But how are we computing these metrics? The speedup can be defined as the division of sequential execution time of some particular software by its parallel execution time, taking into account the number of processing units used for the parallelization:

$$speedup(n) = \frac{T_{base}}{T_{parallel(n)}}, \quad (5.1)$$

being T_{base} the sequential execution time and $T_{parallel(n)}$ the execution time that has benefited from the increase in n resources. Although the speedup typically takes values between 0 and 1, a linear or ideal speedup would take place when $speedup(n) = n$. In addition, a super-linear speedup can be achieved with the help of, among other circumstances, good caching management.

Efficiency is a direct product of the speedup achieved:

$$efficiency(n) = \frac{speedup(n)}{n}, \quad (5.2)$$

where programs that have linear speedup have an efficiency value of 1, whereas many programs that are difficult to parallelize have efficiencies such as $1/\ln(s)$ that approach 0 as the number of processors increases.

This is exactly what scalability means. It represents whether the efficiency remains constant as we increase the number of processing units or not.

| Run Accession | Instrument Model | Stranding | Read Count | Size |
|---------------|------------------------|-----------|------------|--------|
| SRR3192408 | Illumina HiSeq 2000 | PAIRED | 92172367 | 11.5Gb |
| SRR3192409 | Illumina HiSeq 2000 | PAIRED | 113327735 | 14.1Gb |

Table 5.2: Experiment input data

5.2 Benchmark results

The results of the execution of the GEO2RNA-seq pipeline, seen in figure 5.1 coincided with the theory presented above, with the read alignment contributing most of the total run time. In fact, TopHat2 is known for being particularly slow when used outside of the "tuxedo"⁷ suite.

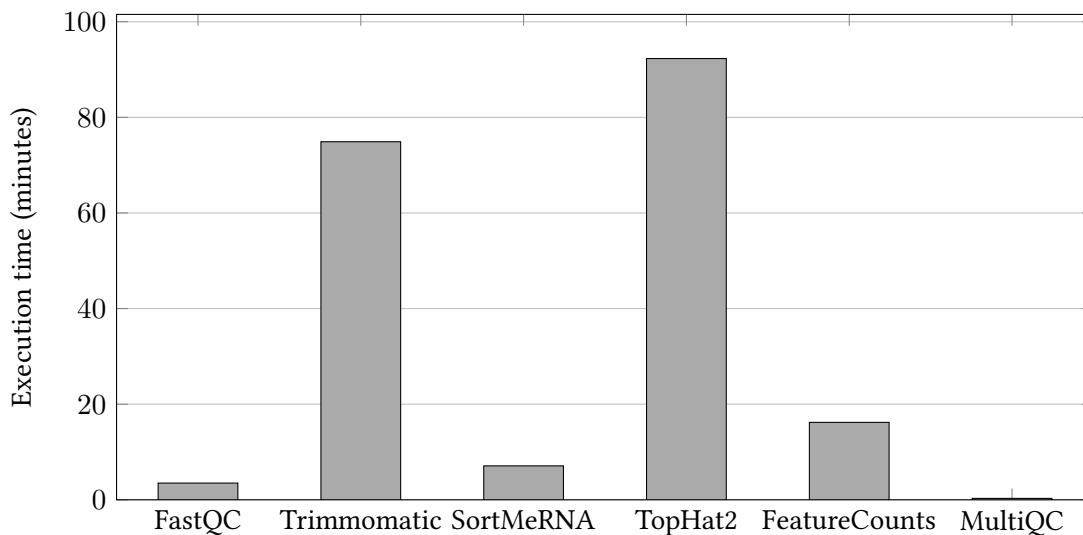


Figure 5.1: GEO2RNA-seq execution times

What was really surprising was how slow the Trimmomatic tool turned out to be. However, this is because this tool is a java application, and further threading only contributes to the Java Virtual Machine garbage collector. Also, the FastQC execution results showed that the input FASTQ files used didn't have the best quality on their bases, so the tool had to do extra work when processing them.

Very similar execution times are presented in the nfcore/RNA-seq pipeline, with STAR being particularly faster when it comes to read alignment. Star is the go-to mapping software

⁷The tuxedo suite is a pipeline composed by BowTie for index building, TopHat2 for mapping and Cufflinks for quantification. They work specially well together.

nowadays, and it is clearly visible why. It is not only faster, but an aftermath comparison between the GEO2RNA-seq's TopHat2 and STAR using the BAMtools suite showed better accuracy and higher quality mappings for the latter. The only downside to this tool is the high memory consumption, as it consumed close to 40Gb of memory during its execution.

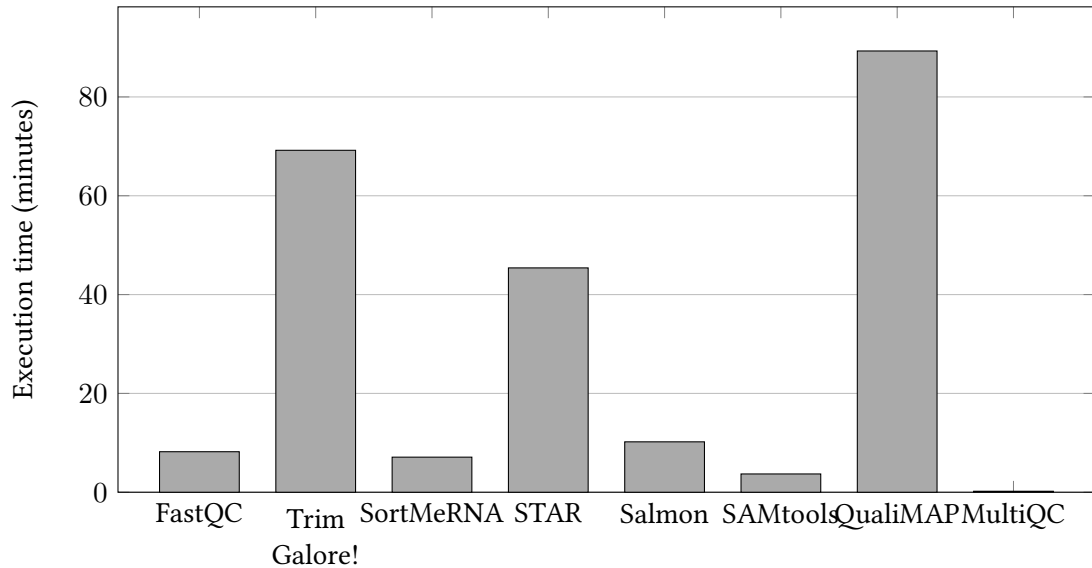


Figure 5.2: nfc core/RNA-seq pipeline execution times.

The other tools and steps that these two pipelines have in common, including FastQC, trimming, SortMeRNA, quantification and MultiQC; showed very similar results. However, with the memory consumption report that Nextflow creates, we can see that a rather odd detail. All tools seem to be in direct proportion to memory consumption, besides Trim Galore!. This is explained when we take a look to the IO operations performed. nfc core/RNA-seq uses channels to pipe inputs and outputs and save on these rather slow operations except for this tool, that needs to write its results on a directory for its subsequent use by MultiQC to generate reports. Also, the fact that the tool is working with no compressed files must be taken into account.

Also, QualiMAP accounted for a good chunk of the execution time. This is because this tool uses a sliding window technique when processing reads. Therefore, a fine tuning of the window size and maximum memory for Java to run with is needed, depending on the testing infrastructure.

5.3 GPU and CPU comparison

Moving on to the variant calling workflow comparison, one can easily appreciate in figure 5.4 that the GPU-based approach was completely demolishing in the standalone Parabricks' tool

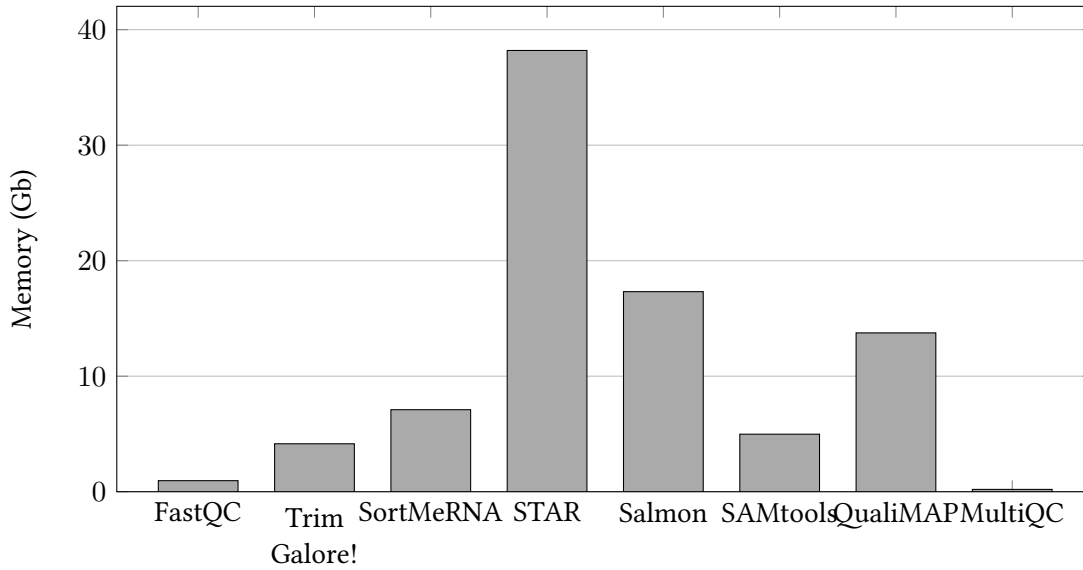


Figure 5.3: nfcore/RNA-seq pipeline memory consumption.

test. With just 2 Nvidia Tesla T4 GPUs, the tool obtained a 29.18 speedup over the single-threaded CPU approach. The tool managed to generate processing-ready BAM files from FASTQ files in under 6 minutes.

Moreover, one detail that has emerged from this experiment is the scalability of STAR. A full disclosure of the metrics can be seen in figure 5.3, where decent execution times were achieved when multi-threading. In fact, this is information that helps to choose the number of threads in the proprietary pipeline for this specific tool, as maximum efficiency and scalability is sought. The sweet-spot looks to be the execution with 8 threads, where there is a balance between speed and 70% efficiency.

| Threads | Speedup | Efficiency[%] |
|---------|---------|---------------|
| 1 | 1 | 100 |
| 2 | 1.59 | 79.80 |
| 4 | 3.13 | 78.49 |
| 8 | 5.52 | 69.04 |
| 16 | 8.44 | 52.75 |
| 32 | 11.85 | 37.04 |

Table 5.3: STAR metrics

Also, similar to what happened to the aforementioned Java applications, both MarkDuplicates and SortSam are part of the GATK's Picard suite, where multiple threads only help the garbage collector. No speedup is achieved whatsoever for these. Besides, tuning the memory

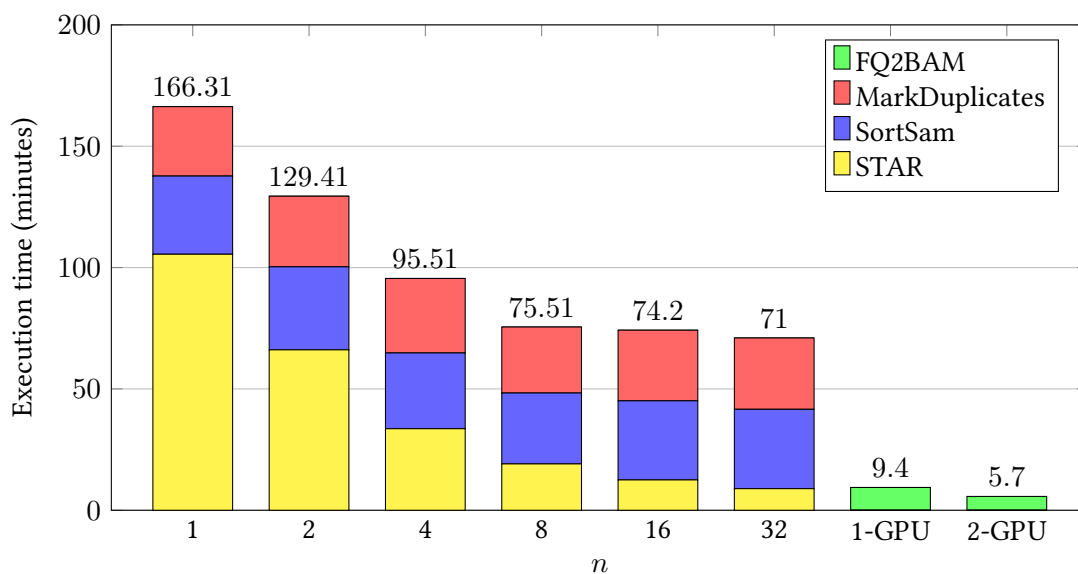


Figure 5.4: fq2bam results

cap available for these tools does not help either, although it should, as the more memory available, the less the garbage collector has to perform.

These applications are, in fact, historical choke points for this workflow, and GATK proposes a SPARK⁸ version of them. With SPARK, GATK aims to speed the process relative to the serial tools. Nevertheless, not every SPARK version is considered production quality and they are optimized for their Terra platform.

To assess full pipeline turnaround time, both approaches were run with the same inputs and configurations as the previous experiment, since the process of the standalone tool is contained within the bigger pipeline.

As expected, the GPU-based approach ended up being way faster than its counterpart, resulting in a 23.5 speedup over the single-threaded execution alternative. Again, as STAR is the only non-GATK tool, every other application turned out to be choke points in this workflow. As explained before, these are essentially single-threaded tools that require optional extra threads for garbage collection. In this case, special attention was given to the memory limitations of the tools, as most of them require more than 20Gb of available memory for them to work under the 2 threads that are used by default.

One extra thing that was tested was the consequences of using a sorting-by-name approach instead of the sorting-by-coordinate option that is used by default in the SortSam step as both work nicely to ease some computational load for the subsequent analysis. However, sorting by name resulted in a slight slowdown of the execution, not only in the actual execu-

⁸<https://gatk.broadinstitute.org/hc/en-us/articles/360035890591-Spark>

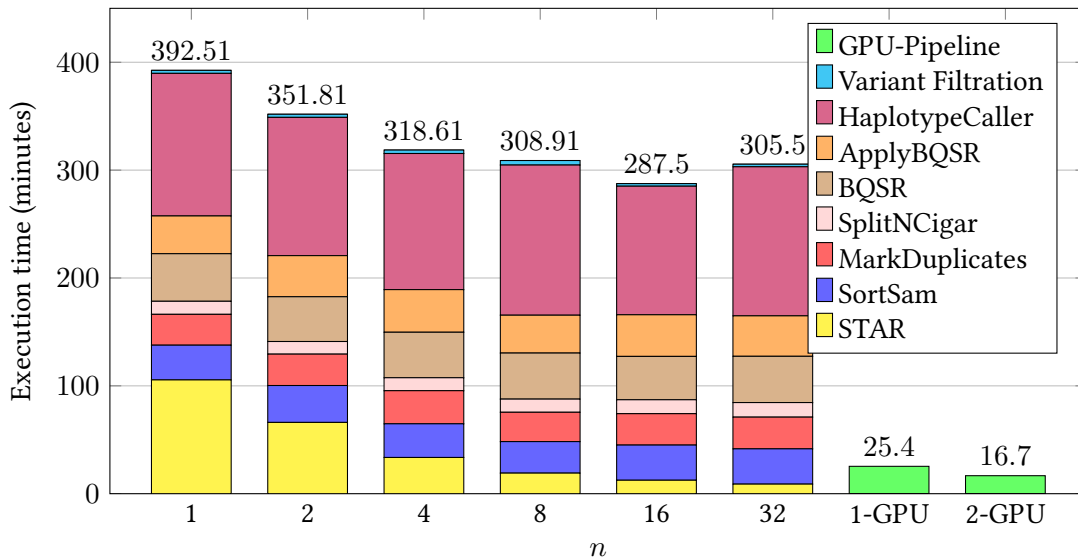


Figure 5.5: Pipeline results

tion of the tool itself, but for the downstream analysis, too.

Besides, predictably, only the output VCF files happened to have some kind of discordance between the CPU and GPU pipeline. Since there are no publicly available reference variant calling files for the sample data used, a quick comparison between both output files was run using VCFtools, reaching a 97.4% coincidence, which implies that both pipelines achieve very similar results. Therefore, Parabricks's pipeline should always be chosen over the CPU one, provided that there are available NVIDIA GPUs, due to both its speed and precision.

With all this information, extra care can be put to fine-tune all the required parameters of the Nextflow pipeline developed, including:

1. No more than 2 threads used for GATK-based applications, as no increase in performance is apparent.
2. Input and output operations can slow down the execution of the workflow massively, as seen in the nfcore pipeline. Thus, it is necessary to chain them and write directly to the processes via the channels that Nextflow implements. If IO operations are strictly needed, it is best to work with compressed files, such as BAM instead of SAM.
3. Sorting by coordinate works specially good for a variant calling workflow.
4. Checking the quality of reads is an extra step that can be useful and gives researchers more insight on the sample data. FastQC looks like a good enough option, as it involves hardly any computational load.

5. Same for MultiQC, as it helps researches to visualize data and interpret results with minimal computational impact.

More insight could have been acquired by using the NVIDIA Visual Profiler⁹. This application is a performance profiling tool part of the CUDA toolkit that helps developers optimize CUDA applications and find bottlenecks. This[?] could[?] have been specially useful for taking a look at how NVIDIA has achieved such speedups and find which drawbacks are causing it to come close to some kind of bottleneck, if they exist.

⁹<https://developer.nvidia.com/nvidia-visual-profiler>

Conclusion

As a closure, this chapter will outline both the conclusions and future lines of work that are opened for revision.

6.1 Conclusion

This research aimed to identify the presence of bottlenecks and the largest computational loads in the pipeline to get an overall view of this process. Based on empirical analysis of the tools that compose the different workflow alternatives, it can be concluded that an original rusty design of experiment that is not adapted to the sample data can be catastrophic. Not only should much emphasis be placed on discerning the really necessary steps of the analysis, but also, chain the different steps carefully in order to avoid extra input/output loads.

There are several paths for RNA-seq data analysis that have arisen over the past few years, as illustrated in Section 2.2.1, but it also raises the question of whether at some point it will be worth the creation of new alternatives for a process that is already sufficiently complex. However, Chapter 5 clearly shows that the common steps of any pipeline—i.e., mapping and assembly—are the most computationally intensive, hence the use of different tools for post-alignment analysis should not be too much of a burden.

Regarding the benchmarking of the best practices variant calling pipelines, although different technologies were used for the workflow descriptions, this testing methodology approach provided new insight into the capabilities of GPU-based techniques. These accelerators are fundamental for any data center infrastructure, as new HPC applications often leverage them to parallelize the processing of many data at the same time by applying the same instruction to them over and over again.

Finally, this thesis has succeeded in bringing together all the general necessary knowledge to introduce new people, such as me, the student, to this field. All the research and papers that are already available on this technique can be very overwhelming. In fact, the study of

the domain is probably the most-time consuming part of this kind of research, where both informaticians and biologists are expected to learn about each other to fully generate comprehensive studies. Once this is clear, the development of applications for bioinformatics can result in a big success, such as the pipeline developed in this project.

6.2 Future lines of work

Although the methodology used for benchmarking was enough to highlight the computational challenges and reasons as to why this technique is rather slow and computationally intensive, in the future it would be interesting to make use of performance analysis tools, that is, profilers. This way, profiling could be used during development and testing as method for debugging and optimizing algorithms. Although this practice is essential for carefully designed applications, the objectivity and veracity should be evaluated by truly specialized personnel and in the right environment. In particular, NVIDIA's profiler would be a nice addition to the GPU-based pipeline testing, to obtain truly interesting insight on how it performs.

Besides, the inclusion of new significative datasets to assess performance would suite the whimsical nature of the tools that were tested, since some work better with a certain type of data and specialize in a specific type of RNA, for example.

Finally, work should be done on the developed pipeline. Although this version focuses on the general pre-processing steps that any batch of data can use, the inclusion of conditional statements to provide the user with more options should be a focus in the future. This way, you could get a very complete pipeline that follows the best practices proposed by GATK but also offers other types of workflows depending on the input data.

Appendices

Pre-processing pipeline user guide

A.1 Requirements

To run the pipeline¹, the following requirements must be met before installation:

1. A POSIX compatible system (Linux, OS X, etc.).
2. Bash 3.2 (or later).
3. Java 8 (or later).
4. Windows Subsystem for Linux (WSL) when working under Windows.

A.2 Installation

Nextflow does not require installation, as the user can directly download the executable package:

```
$ wget -qO- https://get.nextflow.io | bash
```

This will create the `nextflow` main executable file in the current directory. Additionally, the users must make the binary executable:

```
$ chmod +x nextflow
```

Optionally, the user may edit his/her `PATH` for ease of use by modifying their `.bashrc` file and adding:

```
export PATH="/path/to/dir:$PATH"
```

¹https://github.com/santimillang/tfg_pipeline

A.3 Execution

Finally, to execute the pipeline, the user must first pull the corresponding Docker image:

```
$ docker pull santiagomillan/tfg_pipeline:1.0
```

And execute it as follows:

```
$ nextflow run https://github.com/santimillang/tfg_pipeline  
-with-docker
```

Extra arguments can be applied to the execution to automatically generate several kinds of Nextflow pipeline reports:

1. Nextflow can generate an HTML execution report by adding the `-with-report` option.
2. Also, the `-with-trace` option creates an execution tracing file that contains information about each process executed in your pipeline script.
3. A timeline report render is available adding `-with-timeline`.
4. Finally, a direct acyclic graph can be generated using the `-with-dag` option.

The pipeline is prepared to align as many different FASTQ files as needed to the same reference genome. As an example, four FASTQ files are provided as input, with half referring to one batch of data and the other half to another. The input data used as an example is stored in the `data` directory. To change it, the user must create a new directory containing their local data (FASTQ files, FASTA and GFF) to generate analysis ready BAM files.

The output files are stored in the `results` directory, and it includes both the MultiQC final report and the intermediate output of the tools.

List of Acronyms

- AI** Artificial Intelligence. 12
- API** Application Programming Interfaces. 15
- AWS** Amazon Web Service. 22
- BAM** Binary Alignment Map. 10
- cDNA** complementary DNA. 6
- CITIC** Centro de Investigacion en TIC. 22
- CPU** Central Processing Unit. 2
- CUDA** Compute Unified Device Architecture. 12
- DEG** Differentially Expressed Genes. 25
- DNA** Deoxyribonucleic acid. 5
- GATK** Genome Analysis Toolkit. 7
- GEO** Gene Expression Omnibus. 26
- GPU** Graphical Processing Unit. 2
- GTF** Gene Transfer Format. 27
- HPC** High-Performance Computing. 2
- HTML** HyperText Markup Language. 28
- IB** InfiniBand. 48

- IO** Input/Output. 42
- JSON** JavaScript Object Notation. 36
- mRNA** Messenger RNA. 5
- NAS** Network Attached Storage. 47
- NCBI** National Center for Biotechnology Information. 8
- ncRNA** non-coding RNA. 6
- NFS** Network File System. 47
- NGS** Next-Generation Sequencing. 6
- OS** Operative System. 14
- PCR** polymerase chain reaction. 9
- QC** quality control. 29
- RNA** Ribonucleic acid. 1
- RNA-seq** RNA-sequencing. 1
- rRNA** ribosomal RNA. 6
- SAM** Sequence Alignment Map. 29
- SNP** single-nucleotide polymorphism. 10
- SNV** single-nucleotide variation. 9
- SRA** Sequence Read Archive. 26
- VCF** Variant Call Format. 10
- WDL** Workflow Description Language. 10
- WGS** Whole Genome Sequencing. 7

Glossary

- accelerators** Hardware device or software program that enhances the performance of a machine.. 2
- ASCII** Character codification that employs 7 bits.. 11
- bottlenecks** Bottlenecks occurs when there is a restrictive element holding back the performance that could otherwise be achieved.. 2
- cluster** Group of computers linked together usually by a high-speed network and behaving logically as a single server.. 21
- deep-learning** Machine learning algorithms that mimic the human brain.. 13
- high-throughput** The use of many computing resources over long periods of time to accomplish a computational task.. 10
- Java Virtual Machine** Virtual machine that enables a computer to run any Java bytecodes.. 40
- kernel** Also known as core, it is a fundamental part of any operative system that usually requires root access to execute.. 14
- multi-threading** Process of executing multiple threads simultaneously.. 28
- pipeline** A pipeline is a set of data processing steps that are chained together through input-output relations.. 1
- shaders** Computer program that performs graphical calculations.. 13
- UNIX socket** Abstraction that enables two processes to communicate. Used in POSIX operative systems.. 15

Bibliography

- [1] V. Costa, M. Aprile, R. Esposito, and A. Ciccodicola, “RNA-Seq and human complex diseases: recent accomplishments and future perspectives,” *European Journal of Human Genetics*, vol. 21, pp. 134–142, 2013.
- [2] H. Gonorazky, M. Liang, B. Cummings, M. Lek, and J. M. et al., “RNAseq analysis for the diagnosis of muscular dystrophy,” *ANNALS of Clinica and Translational Neurology*, vol. 3, pp. 55–60, 2016.
- [3] Z. Wang, M. Gerstein, and M. Snyder, “RNA-seq: a revolutionary tool for transcriptomics,” *Nature reviews. Genetics*, vol. 10, pp. 57–63, 2009.
- [4] T. Tanjo, Y. Kawai, K. Tokunaga, and et al., “Practical guide for managing large-scale human genome data in research,” *Journal of Human Genetics*, vol. 66, pp. 39–52, 2021.
- [5] B. Thanh, “Analysis of Docker Security,” *arXiv e-prints*, p. arXiv:1501.02967, 2015.
- [6] M. Fowler and J. Highsmith, “The Agile Manifesto,” vol. 9, 2000.
- [7] K. Schwaber, “SCRUM Development Process,” *OOPSLA Conference*, pp. 1–23, 1995.
- [8] C. Verwijs and D. Russo, “A Theory of Scrum Team Effectiveness,” *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 2021.
- [9] B. Seelbinder, T. Wolf, S. Priebe, and S. M. et al., “GEO2RNAseq: An easy-to-use R pipeline for complete pre-processing of RNA-seq data,” *bioRxiv*, 2019.
- [10] P. D. Tomasso, M. Chatzou, E. W. Floden, and P. P. B. et al., “Nextflow enables reproducible computational workflows,” *Nature Biotechnology*, vol. 35, pp. 316–319, 2017.
- [11] T. Barrett, S. E. Wilhite, and P. L. et al., “NCBI GEO: archive for functional genomics data sets—update,” *Nucleic Acids Research*, vol. 41, pp. D991–D995, 2012.

