Tampere University

Shuang Luo

ENABLING AND PERFORMANCE BENCHMARKING OF A NEXT-GENERATION SEQUENCING DATA ANALYSIS PIPELINE

Medicine and Health Technology

Master of Science Thesis
May 2019

ABSTRACT

Shuang Luo: MS.
Master of Science Thesis
Tampere University
Bioengineering, MSc
May 2019

The development of Next Generation Sequencing (NGS) technology resulted the rapid accumulation of a large amount of sequencing data demanding data mining. Various of variant calling softwares and pipelines came into being. Genome Analysis Toolkit (GATK) and its Best Practices quickly became the industrial gold-standard for variant calling because of its speediness, high accuracy and throughput. GATK has been updated all the time. The latest and strongest version is GATK4 which enabled parallelization and cloud infrastructure optimization via Apache spark. Currently, Broad Institute has cooperated with many cloud providers to deploy GATK Best Practices on cloud platform. However, there is no benchmarking data released for GATK4 and no cooperation with CSC (CSC – IT Center of Science Ltd) cPouta IaaS (Infrastructure as a Service) cloud.

We optimized WDL (workflow description language) script of germline SNPs and Indels short variants discovery workflow from Best Practices and ran it by Cromwell execution engine on a virtual machine of cPouta cloud which featured a 24 cores Intel(R) Xeon(R) CPU E5-2680 v3 with hyper-threading. In addition, we benchmarked pipeline execution time(s) for five seperated pipelines of this workflow with three 30X WGS (Whole Genome Sequencing) datasets: NA12878, NA12891 and NA12892 and explored optimized run-time parameters for GATK4 tools, PairHMM thread scalability in HaplotypeCaller, GATK4 thread scalability for PGC in MarkDuplicates and execution times comparison for GATK4 SortSam vs SortSamSpark and MarkDuplicates vs MarkDuplicatesSpark.

We found the real execution time for similar WGS datasets with different size and features showed consistency and execution time and dataset size were roughly positive correlated. The optimal threads number is 12 for GATK4 HaplotypeCaller in ERC mode, giving rise to 12.4% speed-up. The optimal PGC threads number is 2 for GATK4 MarkDuplicates. And, multi-threading with Spark local runner highly speeded up GATK4 tool execution. SortSamSpark enabled 16 local cores gave rise to a speed-up of 83.6%. MarkDuplicatesSpark enabled 16 local cores gave rise to a speed-up of 22.2% and 37.3%, seperately with and without writing metrics file.

With detailed virtural machine setting up, optimized parameters and GATK4 performance benchmarking data, this thesis is a guide for implementation of GATK4 Best Practices germline SNPs and Indels short variants discovery workflow on CSC cPouta cloud platform.

Keywords: NGS, Variant Calling, GATK, Best Practices, Benchmarking, IaaS Cloud, Docker

PREFACE

I would first like to thank my supervisor at CSC, Dr. Pekka Manninen, and my supervisor at TAU, Prof.Dr. Ville Santala, for offering me such a great project and constantly guided me, supported me and provided valuable comments to improve the thesis.

I would also like to thank the experts and colleagues for their kind help, great discussion and all happy times: Saren Ari-Matti, Mattila Kimmo, Lehtivaara Maria, Laxmana Yetukuri, Eduardo Gonzalez and also other members in the team.

To my closer friends, I would like to express my gratitude for their loyal friendship and support all the time.

Last but not least, I would like to thank my parents for their unconditional support and love throughout my life.

Espoo, 22 May 2019

Shuang Luo

CONTENTS

APPENDIX A: Arguments for Main Tools

LIST OF SYMBOLS AND ABBREVIATIONS

BAM	Binary Alignment Map
BP	Base Pair
BQSR	Base Quality Score Recalibration
CNV	Copy Number Variations
CSC	CSC – IT Center for Science Ltd
CWL	Common Workflow Language
ENA	European Nucleotide Archive
FIMM	Institute of Molecular Medicine Finland
GATK	Genome Analysis Toolkit
GKL	Genomics Kernel Library
IaaS	Infrastructure as a Service
ICT	Information and communications technology
Indel	Insertion/Deletion
MPLS	Multiprotocol Label Switching
NGS	Next Generation Sequencing
PGC	Parallel Garbage Collection
SAM	Sequence Alignment Map
SDMS	Sensitive Data Management System
SNP	Single Nucleotide Polymorphism
SRA	Short Read Archive
SV	Structure Variation
VCF	Variant Call Format
VQSR	Variant Quality Score Recalibration
VM	Virtual Machine
WDL	Workflow Description Language
WGS	Whole-genome Sequencing

.

1. INTRODUCTION

Whole-genome sequencing (WGS) means completing the sequencing of a whole genome at one time. Thus, WGS data contains all information of that genome and can be used to discovery almost any variants from an organism. WGS technology provides various of benefits in both scientific research and clinical diagnostics [1–4]. With the purpose of more extensive applications of WGS, fast sequencing technology and affordable sequencing price are needed [5]. The advent of Next-Generation Sequencing (NGS) technique [6, 7] revolutionized the field of WGS [8] and greatly expanded its range of applications, from research laboratory to clinic [9]. Since 2005, NGS techniques developed and started dominating the sequencing market. It provided higher sensitivity, larger throughput, improved speed and much lower price. The wide adoption of NGS has generated and accumulated lots of sequencing data, which demands deeper and wider data mining and analysis capabilities. A lot of genetic data analysis pipelines and tools were developed to call variants from NGS data. The Genome Analysis Toolkit (GATK) and its Best Practices [10, 11] by Broad Institute are the most outstanding representatives. GATK contains lots of genetic analysis tools and specially focus on variants discovery and genotyping from Illumina human WGS and whole-exome sequencing (WES) data. GATK is compatible to multi-platform and takes advantage of Docker container technology [12] to reduce or even remove the environment configuration issues. There are different versions of GATK. The newest one is the GATK4, which is faster, more accurate and uses Apache Spark [13] for parallel processing and cloud infrastructure utilization. To ensure the high repeatability and accuracy of variant calling process, Broad Institute designed and promoted a series of variant calling workflows named Best Practices which provides step by step guidelines from a DNA library preparation to final variant callset collecting. The most widely used Best Practices workflow is the one for germline SNPs and Indels variants discovery [14] in DNA sequencing data, which is also the workflow discussed and tested in this thesis. Broad Institute offered a new pipelining solution which is capable of parallel computing and user-friendly, it contains a new workflow description language called WDL and its execution engine Cromwell, which can execute WDL script in a local platform and on a cloud computing platform.

The Institute of Molecular Medicine Finland (FIMM) and CSC – IT Center for Science Ltd (CSC), the company where I wrote this thesis, decided to start a pilot project which aims to connect FIMM´s biomedical data-producing devices directly to CSC´s computing platform and enable pipeline implementation and performance benchmarking of GATK4 Best Practices with Docker container on CSC´s Pouta open shell IaaS cloud.

1.1 Whole Genome Sequencing (WGS) and Next Generation Sequencing (NGS)

DNA sequencing is the process of determining the order of four different nucleotides in a specific DNA fragment [15]. They are adenine (A), thymine (T), guanine (G) and cytosine (C). WGS refers to sequencing the entire genome of an organism, not just this organism's chromosomal DNA, but also mitochondria DNA or chloroplast DNA for plants. Thus, this technique can almost identify any type of genetic mutations for an organism. The value of WGS is enormous. As WGS data encompasses the intrinsic relevance of all genes and correlative life features, it helps to identify the new biomarkers and drug targets, adds the information of human complex disease,

and has great guiding significance in animal and plant economic traits and breeding research, species origin, domestication, group history dynamics, etc. On the other hand, it requires more data interpretation, better storage and management solutions and higher technical challenges, such as much faster sequencing speed, bigger data storage, faster and easy data transfer and lower sequencing cost per genome.

In 1973, Gilbert and Maxam published the 24 base-pairs long nucleotide sequence of the lac operator using “wandering-spot analysis” method [16], which took 2 years and 1 month to determine per bp. In 1977, Sanger published the 5375 base-pairs DNA sequence of the genome of bacteriophage φX174 using a simple but faster “plus and minus” method [17]. In 1979, whole genome shotgun sequencing was deployed to sequence small genomes which contains several thousands of nucleotides [18]. Eight years later, Smith, Hood and Applied Biosystems developed fluorescence-based Sanger sequencing machines named ABI Prism 370A, which was able to sequence around 1000 bases per day [19–21]. It exponentially speeded up DNA sequencing. However, these techniques were still time-consuming, labor-intensive and quite expensive for WGS. But bioscientists were confident that whole genome sequencing of species will provide great support to various aspects of life science research, such as: disease analysis, breeding and evolution science [22]. The Human Genome Project (HGP) with a budget of 3 billion US dollars is one of the manifestations of scientists' confidence. It was firstly proposed by American scientists in 1985, then was jointly initiated globally by scientists in the United States, Britain, France, Germany, Japan and China in 1990 [23]. The contradiction of scientists' hunger for the large-scale sequencing of species and the limitations of existing sequencing technologies has always existed. Until 2005, this contradiction was drastically mitigated. A company 454 Life Science introduced the revolutionary pyrophosphate sequencing-based ultra-high-throughput genome sequencing system. The Genome Sequencer 20 System, which was reported by Nature magazine as a milestone [7]. It pioneered “sequencing-by-synthesis” method and became the first pioneer of NGS. It sequenced 20,000,000 bases per day. NGS is also known as high-throughput sequencing or massive parallel sequencing. NGS is marked by the ability of sequence hundreds of thousands to hundreds of millions of DNA molecules in parallel with the using of a short read length. NGS gets its name after the first-generation Sanger sequencing, which mainly includes three modern sequencing technologies (systems/platforms): Illumina´s Genome Analyzer, HiSeq 2000 and MiSeq; Roche´s GS FLX Titanium and GS Junior; Life Sciences’ Ion Torrent PGM and SOLiD sequencing [24]. NGS offers rich information, higher sensitivity to detect low frequency variants, higher throughput of sequencing and lower sequencing cost.

1.2 FIMM, CSC Pouta IaaS Cloud and Pilot Project

Biomedical data-producing devices can generate large amounts of data every day. For example, a modern genome sequencer can generate 3 terabytes of data per day [25]. This data usually needs to be pre-processed and annotated with appropriate metadata before being specifically analyzed. And the environment/platform for data generation and storage is different from the computing platform used for the data analysis, which means that large-scale data transfer is essential. Previously, the transfer of large amounts of data was labor-intensive work, requiring not only uninterrupted supervision but also a lot of manual operations, such as up to 50 checks. Moreover, biomedical data, especially human data, is sensitive. Therefore, the storage and transfer of biological data require a higher level of security.

FIMM, Institute of Molecular Medicine Finland, concentrates on human genetic research and personalized medicine. FIMM has tons of biological data that requires a large amount of safe storage space to store the data and a high-speed computing platform to analysis the data. CSC is a Finnish supercomputer center with absolute security, strong computing power and massive

storage space and it is free for academic use in Finland. It is undoubtedly the best choice for FIMM.

CSC – IT Center for Science is a Finnish center of expertise in information technology for science, education and culture. It is a non-profit Finnish state enterprise, 70% share is hold by Finnish state while the rest of shares are hold by higher education institutions, such as universities and universities of applied science. CSC aims at offering internationally high quality digital service, high-performance computing, expert ICT (Information and communications technology) solutions, manages data and software. It provides both public commodity and private senstive data processing IaaS cloud services. They are cPouta and ePouta, seperately. IaaS stands for Infrastructure as a Service. Consumers can get services from the infrastructure via the Internet, containing controling of memory, computing resource and storage and so on. Both Pouta IaaS clouds provide a programmable API and a web interface. Users can easily generate their online virtual machine (VM) instance via the web interface, control the VM, network and storage as well. VMs are run on various sets of compute node hardware. CSC currently provides two kinds of compute nodes, one for High-performance computing (HPC) load which connected with 40 Gb/s Ethernet and another for genetic computing load (such as web servers or software developing servers) which connected with 10 Gb/s Ethernet. User can online connect their virtual machine via given external IP address as well. Compared with other IaaS platforms, CSC cPouta IaaS Cloud HPC nodes are exclusively designed for HPC, thus virtual machine scheduling does not oversubscribe the resources. It avoids contention and supposed to provide more stable and predictable performance characteristics which is optimal for testing and benchmarking work.

As discussed in introduction, FIMM and CSC was targeting on directly connecting sequencer to computing platform and achieving complete automation of NGS data analysis workflows.

An example process is shown in Figure 1 and is described below.

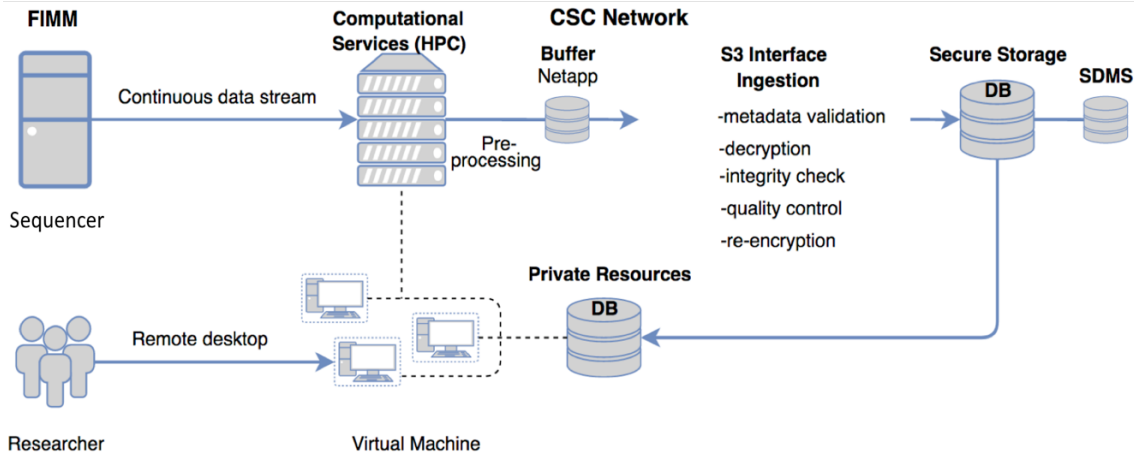

Figure 1. Example process of the CSC and FIMM cooperated pilot project.

- First, FIMM genome sequencer write a new genome as a uBAM file or FASTQ file (introduced later) on a server provided by CSC. This server encrypts the data file using CSC public key. File is sent to CSC via private network that connects FIMM intranet domain to CSC network using MPLS (Multiprotocol Label Switching, for continuous data stream). MPLS is a internet protocol routing technique in telecommunications networks which uses connection-oriented short path label to direct data tansfer between nodes, thus speeding up traffic flow [26].
- Second, submitted file is stored to a buffer. Before this file will be processed, metadata requirements will be checked and must be fulfiled (pre-processing).

- Third, standard workflow to decrypt the data, validate integrity, quality control (QC) or other type of standard processing steps. Based on the results of this analysis and research target, the orginal file is delected or re-encrypted for others to use (ingestion).
- Fourth, the re-encrypted original file and result file will be passed to and archived in a secure storage database (Sensitive Data Management System, SDMS).
- Finally, the data stored in the secure storage database also can be selectively passed to private resources database. Meanwhile, researchers can control and manage their data and analysis workflows via remote desktop connection which connecting their virtual machines of HPC server and private resources database.

This thesis considers the implementing and benchmarking an NGS data analysis workflow, which relate to the ingestion part of the Figure 1. Figure 2 shows the detailed example process of it and is described below.

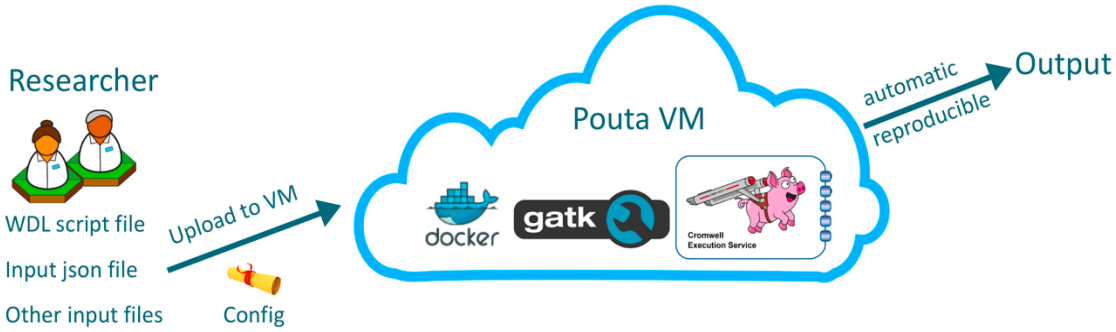

Figure 2. NGS data analysis workflow implementation on Pouta VM.

NGS data analysis workflow was constructed as an WDL script. Researcher uploaded WDL script file, input json file, configuration file if needed, and other input files, such as reference genome file and known sites file to CSC Pouta VM. Script was run on its execution engine Cromwell in Pouta VM. More information is descripted in Section 2 and Section 3.

2. ABOUT NGS METHODOLOGY AND APPLICATION

A main NGS methodology and application is variant calling. In this section, variant calling stages, different types of variants, variant calling workflow involved file formats, such as FASTA, FASTQ, SRA, SAM, BAM, CRAM, VCF, GVCF and the relative index and nonstandard files are presented. The gold-standard variant caller Genome Analysis Toolkits (GATK) and its Best Practices workflows, a new and powerful workflow description language WDL and its execution engine Cromwell and container technology, especially Docker containers are introduced as well.

2.1 Variants and Variant Calling

Since the NGS method has been widely used worldwide, the total amount of sequence data has increased in an unexpected speed. The huge amount of sequence data makes demand on deeper and various of analysis. Many new bioinformatics pipelines and related tools have been developed rapidly to call variants from NGS data. “Variant calling” refers to the process of identifying variants from sequence data.

Variants can be mainly classified into several kinds: Single Nucleotide Polymorphism (SNP), Insertion/Deletion (Indel), Structure Variation (SV) and Copy Number Variation (CNV).

SNP is one nucleotide difference at a given position compared to its reference genome. It is the most common variation occur in human beings with a frequency around 0.05% to 0.1% [27].

Indel means a short DNA fragment is inserted or deleted in the given position. The length of an Indel is supposed to be less than 1 k base pair (bp) [28]. In fact, in human genome, the most common length of Indel is much shorter. The mean length of insertion is only 8 bp and deletion is only 5 bp [29].

SV is the variation in structure of an organism's chromosome. SV is a major source of genomic variation and contains various kinds of variations, such as duplications, Copy Number Variations, inversions, translocations and large insertions and deletions (>1 kb) [30]. Strictly speaking, CNV belongs to SV. But CNV is one of the most important pathogenic factors of human disease and its probability of occurrence is relatively high, it is usually discussed separately [31, 32]. Research shows that there are around two thirds of human genome is constituted from repeats [33]. CNV is a type of duplication or deletion event that affects lots of base pairs.

For the whole-genome sequence and whole-exome sequence, there are usually three steps to deploy variant calling: sequencing, read mapping/alignment and variant identification. First, sequencing the whole genome of an individual or group to generate the WGS or WES data, usually in the FASTQ format [34] or unmapped BAM (uBAM) format [35]. Second, aligning these reads to a reference genome to form BAM or CRAM [36] files. Finally, distinguishing the difference between sequencing reads and reference genome, generating a VCF file which contains all the discovered variants information. The variant calling performance and time-consumed are largely depend on the caller applied and calling strategies. In addition, the sequencing depth, coverage, quality and aligner involved have an impact.

2.2 On File Formats

File format, also known as file type, refers to a special encoding method of information, which enables the information being stored and identified in a computer file. Different file formats are

needed by different targets and fields. There are various file formats are created for bioinformatics, such as FASTA, FASTQ, BAM, BED, VCF, BCF and so on. They are created originally by certain specific bio software or workflow pipelines, and developed into commonly deployed standard format. Unification standards of file formats highly benefit the development of bioinformatics tools and analysis pipelines. No matter what kind of tools, workflows or platforms are used by upstream sample preparation, people always can use the same downstream analysis workflow if they want due to the unification standards of file formats. In this section, I discuss various kinds of file formats which are most relevant to variant calling.

2.2.1 FASTA format

The FASTA format is a text-based format used to record and present nucleotide or peptide sequences, in which each letter code refers to each nucleotide or amino acid. It also contains the sequence name and necessary comments. The FASTA format was created by a software package named FASTA. It has become a standard sequence file format because it occupies small size and is easily processed with any text-processing tools.

A typical FASTA file starts with a one-line description, followed by lines of sequence data. The single-line description starts with a greater-than (>) symbol which is obligatory and closely followed by the identifier and description of sequence which are nonessential.

Example: (Neisseria gonorrhoeae plasmid pCmGFP)

```
>NC_011521.1:4419-5135 Neisseria gonorrhoeae plasmid pCmGFP, complete se-
quence
ATGAGTAAAGGAGAAGAACTTTTCACTGGAGTTGTCCCAATTCTTGTTGAATTAGATGGT
…
CTTCTTGAGTTTGTAACAGCTGCTGGGATTACACATGGCATGGATGAACTATACAAATAA
```

In a variant calling workflow, the reference genome sequence usually in the FASTA format. Typical file extensions are .fasta, .fa or .fa.gz (gzip compressed).

2.2.2 FASTQ and SRA formats

The FASTQ format can be viewed as an extension of the FASTA format. It contains in addition correlative sequence quality information after its sequence data [34]. The quality information is presented as a numeric score, which is encoded as ASCII character to keep one-to-one mapping between sequence and its quality. Like the FASTA format, the FASTQ is text-based file format. It was created by Sanger institute, and has become the default standard for storing high-throughput sequencing raw reads directly from sequencer. Typical file extensions are .fastq, .fq or .fq.gz (gzip compressed).

A typical FASTQ file contains 4 lines: The first line starts with the character “@”, closely followed by a sequence identifier and a nonessential sequence description. The second line presents the sequence data. The third line starts with symbol plus (+) and optionally followed by the sequence identifier, which is the same as the one in the first line. The last line shows the sequence base quality encoded as an ASCII character.

Example: Paired-end FASTQ file of NA12878

```
@ERR194147.1 HSQ1004:134:C0D8DACXX:1:1104:3874:86238/1
GGTTCCTACTTCAGGGTCATAAAGCCTAAATAGCCCACACGTTCCCCTTAAATAAGA-
CATCACGATGGATCACAGGTCTATCACCCTATTAACCACTCACG
+
CC@FFFFFHHHHHJJJFHIIJJJJJJIHJIIJJJJJJJJIIGIJJIJJJIJJJIJIJJJJJJJJJ-
JIJHHHHFFFDEEEEEEEEDDDCDDEEDDDDDDDDD
```

The base quality score is used to describe the possibility of an inaccurately called base. If the base sequencing error rate is P_{error}, the value of base quality is $Q = -10\log_{10}{}^{*} P_{error}$. Thus, the Q

value is proportional to base quality. Because of the development of sequencing technology in different companies, there are three different kinds of FASTQ formats. They are separately deployed by Sanger, Solexa and Illumina sequence platforms. Table 1 shows the comparison among them.

Table 1. *FASTQ formats comparison.*

Sequencing platform	ASCII character range	Lower limit	Quality score type	Quality score range	Current situation of application
Sanger, Illumina (1.8 and later version)	33-126	33	Phred quality score	0-93	Still using
Solexa, Illumina (1.3 and earlier version)	59-126	64	Solex quality score	5-62	Except for already sequenced data, no longer been used
Illumina (1.3 to 1.7 version)	64-126	64	Phred quality score	0-62	Except for already sequenced data, no longer been used

The NCBI SRA format (short read archive, currently named as sequence read archive) can be viewed as a variant of FASTQ format but contains more description information. Reads stored in the SRA format can be easily converted into FASTQ format via SRA toolkit.

2.2.3 SAM, BAM and CRAM formats

The SAM (Sequence Alignment Map) format is a file format used to store aligned/mapped high-throughput sequencing data, using TAB as separator [35]. The BAM (Binary Alignment Map) format and the CRAM format both are compressed versions of SAM. BAM is by definition the lossless compression, while CRAM can range from lossless to lossy compression depending on how much compression rate is wanted. Generally, CRAM shows significantly better lossless compression than BAM and is highly compatible with BAM. One can easily convert BAM files to CRAM.

A typical SAM file contains two sections, an annotation information (header section) and an alignment result (alignment section). The header section is optional. Each line starts with the character “@”, followed by a two-letter header record type code. They are: @HD used to illustrate the format version, sorting order of alignment, grouping of alignments and sub-sorting order of alignment; @SQ for reference sequence dictionary; @RG for read group; @PG for program information used; And @CO for one-line text comment. The alignment section, one alignment line presents per read information, includes 11 mandatory fields and an optional field, which are separated by tabs. These fields are presented in a fixed order. When certain field information is not available, according to the field definition, it can be “0” or “*”. These mandatory fields´ types and descriptions are presented in following Table 2:

Table 2. *Overview of mandatory fields in SAM.*

Order	Field	Type	Regexp/Range	Description

1	QNAME	String	[!-?A-~]{1,254}	Query template NAME
2	FLAG	Int	[0, 2^{16} − 1]	bitwise FLAG: The number of the template mapping case, each number represents a comparison, where the value is the sum of the numbers that match the situation
3	RNAME	String	*|[!-()+-<>-~][!-~]*	Reference sequence name: If the SQ-SN is defined in the comment, it must be consistent with it, and for unmapped segment without coordinate, marked as "*"
4	POS	Int	[0, 2^{31} − 1]	1-based leftmost mapping position: First base in a reference sequence coordinated as 1, 0 means an unmapped read
5	MAPQ	Int	[0, 2^{8} − 1]	Mapping Quality = −10 log10* Probability when mapping position is wrong, rounded to the nearest integer.
6	CIGAR	String	*|([0-9]+[MIDNSHPX=])+	CIGAR string: Compact Idiosyncratic Gapped Alignment Report, which is based on a reference sequence and uses numbers with letters to indicate alignment results
7	RNEXT	String	*|=|[!-()+-<>-~][!-~]*	Ref. name of the mate/next read
8	PNEXT	Int	[0, 2^{31} − 1]	Position of the mate/next read
9	TLEN	Int	[-2^{31} + 1, 2^{31} − 1]	observed template length: the leftmost is positive, the rightmost is negative, the middle is not defined positive or negative, the segmentation is not divided (single-segment), or when it is not available, here is 0
10	SEQ	String	*|[A-Za-z=.]+	segment sequence
11	QUAL	String	[!-~]+	ASCII of Phred-scaled base quality+33

Optional fields (format: TAG: TYPE: VALUE), where TAG has two uppercase letters, each TAG represents a type of information, one TAG can only appear once in one line. The TYPE represents the TAG corresponding a value, which can be a printable character, signed integer, single-precision floating number, printable string including space, byte array in the Hex format and integer or numeric array.

Fundamentally, SAM, BAM and CRAM files, although are different formats, contain the same information and can be replaced with each other. However, according to their respective characteristics, they are used for different purposes in practice.

BAM files are used directly in GATK Best Practices for processing and analysis. The CRAM file is mainly used for archiving because it has the highest compression ratio and may cause performance issues if directly used like BAM. Both SAM files and CRAM files can be easily converted to BAM files by Picard [37]. In GATK4, Picard is bundled.

In GATK Best Practices, SAM, BAM, CRAM files need to meet additional requirements:

- Must contain right file extension: .sam, .bam and .cram.

- Must passed with their index files. If there is not yet, users can easily generate the index files by Picard BuildBamIndex.
- Must successfully past the validation of Picard ValidateSamFile.
- The header section must contain @RG with sample name.
- Every read in BAM file must belong to a read group listed in header.
- The reads must be sorted in coordinate order.

The easiest way to check whether a file meets the GATK requirements is to use SAMtools [35] to check the BAM header: `Samtools view -H data.bam`.

2.2.4 VCF and GVCF formats

VCF (Variant Call Format) is a plain-text file format, which is used to store genetic variants. VCF is the favorite format and only well-supported variant call format by GATK, which can be generate by GATK HaplotypeCaller [38] in normal mode or by GenotypeGVCFs [39]. This format originally developed by 1000 Genome Project [40]. However, now the Genomic Data Toolkit team of the Global Alliance for Genomics and Health has the responsibility of continuous development and updates. VCF not just stores the variants themselves (SNPs, Indels, Structural variants and so on) and locations but also other metadata, such as the dataset ownership, sample statistics and a quality score [41]. VCF is usually stored in its compressed version, a .gz file or a BCF depending on two different compression methods used. VCF is a text file format, thus it can be opened and edited by any text editors but as discussed before, it contains abundant information, and hence VCF file is usually big. Editing the VCF file with a text editor is not a good choice, as it can even cause editor to crash. Also, to ensure the right file format, users are forbidden to edit it with any word processors. Opening or editing part of VCF with a special tool is the best way.

The basic structure of VCF can be divided into two parts: header and variant call records. The following figure shows a typical VCF format structure.

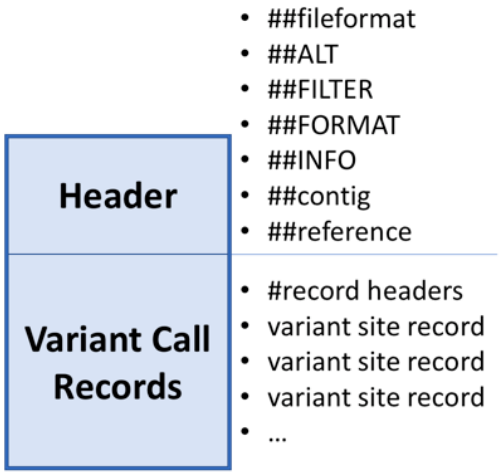

Figure 3. Basic structure of VCF.

There is one kind of VCFs which is called the "sites-only" VCF having a different structure. It contains only 8 columns, without FORMAT and sample-specific information.

If GATK HaplotypeCaller is called in the -ERC GVCF mode, not in the normal mode, a GVCF file will be produced instead of a VCF file. GVCF (Genomic VCF) is a specific kind of VCF file. Its basic format structure is the same as normal VCF but it contains extra information. No matter where there is a variant call or not, the GVCF contains records for all sites, which is essential for later joint calling cohort analysis. There is another kind of GVCF produced by HaplotypeCaller in the -ERC BP_RESOLUTION mode. I didn´t use it in this thesis. However, the difference between the two GVCFs is: later kind of GVCF has an individual record at every site while the ERC GVCF

has an individual record only at variant sites but instead has a non-variant block records for all non-variant sites.

2.2.5 Index file and nonstandard file formats

GATK Best Practices is mainly related to three kinds of file formats which can be indexed: BAM, VCF and FASTA (sometimes, FASTQ). For a big file, the index file works as its external table of contents which allow fast random access to this file without reading through the whole big file. The index file extension follows a similar rule. For BAM, it is .bai, for FASTA, it is .fai, for VCF, it is .idx and for VCF.gz, it is .tbi. Users can easily index their files. There are various ways to do it. For example, SAMtools is an easy option to index BAM, FASTA and FASTQ file with command: `samtools index *.bam`, `samtools faidx *.fasta` and `samtools fqidx *.fastq`. IGV [42, 43] and GATK are good at indexing VCF file.

Besides these standard file formats mentioned above, many nonstandard file formats have been created by programs. For instance, GATK (see section 2.3) creates lots of intermediate files.

2.3 Genome Analysis Toolkit (GATK) and Best Practices

Genome Analysis Toolkit (GATK)

Because of the increase in demand, many variant callers appear on the market. Four most popular among them are SAMtools, Genome Analysis Toolkit (GATK), glftools (http://csg.sph.umich.edu/abecasis/glfTools/) and Atlas2 [10, 11, 35, 44]. There are also a number of articles that compare the performance of various callers [45]. Generally speaking, GATK demonstrates faster speed, bigger throughput and higher accuracy. As time progressed, GATK developed by the Broad Institute has gradually became the gold-standard for the variant calling process with its outstanding performance. The toolkit offers a wide variety of tools and it focuses on variant discovery and genotyping. It is an industry standard for identifying SNPs and Indels in germline DNA and RNAseq, especially for human whole-genome sequencing and whole-exome sequencing data generated by Illumine sequencing technology [45]. Besides the variant caller itself, GATK contains lots of utilities to perform related tasks such as quality control and assessment of high-throughput sequencing data, and has the popular Picard toolkits bundled. In other words, GATK features a rich professional ecosystem: users can not only complete a single simple work such as data diagnosis, but also a complex series of work, such as from reads to detected variants. All these tasks can be accomplished by the harmonized GATK command syntax and a user guide. At the heart of GATK is an industrial-grade infrastructure and engine that handles data access, transformation and traversal, and high-performance computing. In particular, GATK4 uses Apache spark [13] for parallelization and cloud infrastructure optimization.

GATK supports different POSIX-compatible platforms such as Linux, Unix and MacOSX but does not support Microsoft Windows. The major system requirement is Java 1.8 (Oracle Java and OpenJDK are both officially supported) and some tools need Python and R. Docker containers are the recommend way to run GATK without worrying about environment configuration and details about Docker will be discussed in a subsequent section. Users can either download GATK Docker images from Dockerhub [46] (broadinstitute/gatk) or the GATK package source code directly from Broad Institute GATK download page [47].

GATK Best Practices

As discussed before, the variant discovery begins from the DNA library preparation, go through sequencing, to variant calling. The GATK team believes that each task should be a step in a well-documented protocol instead of isolated and disconnected task. Differences in experimental design or differences in sample preparation and processing will both result in a change in the final

variation test results. In order to achieve high repeatability and high accuracy of the test results, a complete set of workflows is crucial.

The industry standard GATK Best Practices are the variant discovery workflows for high-throughput sequencing data used at the Broad Institute. It offers step-by-step protocols for detecting different variants and in different type of input data. There are six main workflows, they are: Data Pre-processing, Germline SNPs and Indels, Somatic SNVs and Indels, RNAseq SNPs and Indels, Germline CNVs and Somatic CNVs.

The analysis phase of the workflow mainly consists of two or three stages: 1) Data pre-processing, which is a general step, starting from raw data (in FASTQ or uBAM format) to analysis-ready BAM file. This phase involves alignment to the reference genome, sorting, marking duplication and Recalibrate base quality scores. 2) Variant discovery, which starts from analysis-ready BAM file. This involves detecting variants from one or more individual genomes and filtering to reduce false positives. The output variants are generally represented in VCF format, unless variants such as CNV that cannot be represented by VCF, will be presented by other structured text-based formats. 3) Filtering and annotation, which is an optional step that typically includes filtering and annotation to generate a call set for downstream analysis. This typically involves using known variants, truth sets and other metadata resources to evaluate and improve the accuracy of the results and provide additional information.

2.4 WDL and Cromwell

An efficient genetic analysis workflow requires not only a complex set of processing and analysis chains, but also parallelism, dependencies between input and output, and intelligent recovery from interrupted state. In the past, people often used Perl to write analysis scripts, and Scala [48] implemented parallel computing. But this is not an efficient nor simple solution for biologists, that is because they require a lot of computer knowledge and programming skills. But biologists prefer to focus on the genetic analysis work itself, not on the learning of tools. Broad Institute now offers a new pipelining solution, which involves a new workflow description language, WDL and an execution engine that can execute it both on the cloud and locally, Cromwell [49].

WDL

WDL is an open source workflow description language written in human readable and easy to write syntax. It defines the various tasks in a workflow, links them in an orderly manner and has built-in advanced functions for implementing parallel computing (scatter-gather). Almost everyone can understand its usage and can run it on the cloud and locally. WDL is a cross-user and platform language. In order to improve the repeatability and expand the scope of GATK best practices workflows, Broad Institute's production pipelines scripts of GATK4 Best Practices are written in WDL.

The core components of WDL scripts are: workflow, task, call, command and output. There are additional optional components: runtime parameters, meta-information and parameter-meta descriptions of inputs and outputs.

Cromwell

Cromwell is an open source workflow management system for scientific workflows. It can run WDL and Common workflow language (CWL) both. It supports multiple public clouds (Google Cloud, Alibaba Cloud, Intel BIGstack and Amazon Web Services) and HPC schedules natively via pluggable backend. It is flexible in scale can either work as a server or just in a standalone mode.

2.5 Virtual Machines and Containers

One problem in software development and running is the environment configuration. To successfully run a certain software, users must ensure that they deployed the right operating system settings and installed various libraries and components. It is really common that a software works very well on some user´s computer but totally crashes on others. It asks for extra work for developers and good computer knowledge for users as well. Here, I discuss two popular solutions: virtual machine and containers.

2.5.1 Virtual Machine

One solution is a virtual machine. It can run another operating system in one operating system, such as running a Linux system on a Windows system. The application is not aware of this because the virtual machine looks exactly the same as the real system, and for the underlying system, the virtual machine is a normal file, deleted without it, and has no effect on other parts.

Although the user can restore the original environment of the software through the virtual machine. However, this approach has several drawbacks [50].

(1) More resources

The virtual machine monopolizes part of the memory and hard disk space. Other programs cannot use these resources while it is running. Even if the application inside the virtual machine, the actual memory used is only 1 MB, the virtual machine still needs several hundred MB of memory to run.

(2) More redundant steps

A virtual machine is a complete operating system. Some system-level operating steps cannot be skipped, such as user login.

(3) Slow start

The time it cost to start the virtual machine is exactly it cost to start the operating system. It may take a few minutes for the app to actually run.

2.5.2 Linux Container

Due to these shortcomings of virtual machines, another virtualization technology has been developed: Linux Containers (LXC).

Instead of emulating a complete operating system, the Linux container isolates processes. In other words, a protective layer is placed outside the normal process. For the process inside the container, the various resources it touches are virtual, thus achieving isolation from the underlying system.

Since containers are process-level, there are many advantages over virtual machines.

(1) Fast start-up

The application inside the container is directly a process of the underlying system, not a process inside the virtual machine. Therefore, starting the container is equivalent to starting a process, rather than starting an operating system, the speed is much faster.

(2) Less resource occupation

The container only occupies the required resources. Because the virtual machine is a complete operating system, it is inevitable to occupy all resources. In addition, multiple containers can share resources, and virtual machines are exclusive resources.

(3) small size

The container only needs to contain the components used, and the virtual machine is packaged by the entire operating system, so the container file is much smaller than the virtual machine file.

(4) Better application performance

Strictly speaking, whether a container or a virtual machine provides better application performance is depending on the type of workload [51–53]. However, containers usually offer better application performance because of no hypervisor, especially when the application talks to the I/O devices [54].

In short, the container is like a lightweight virtual machine that provides a virtualized environment, but at a much lower overhead.

2.5.3 Architecture comparison virtual machine vs. container

Architecture comparison between a virtual machine and a container is presented in Figure 4.

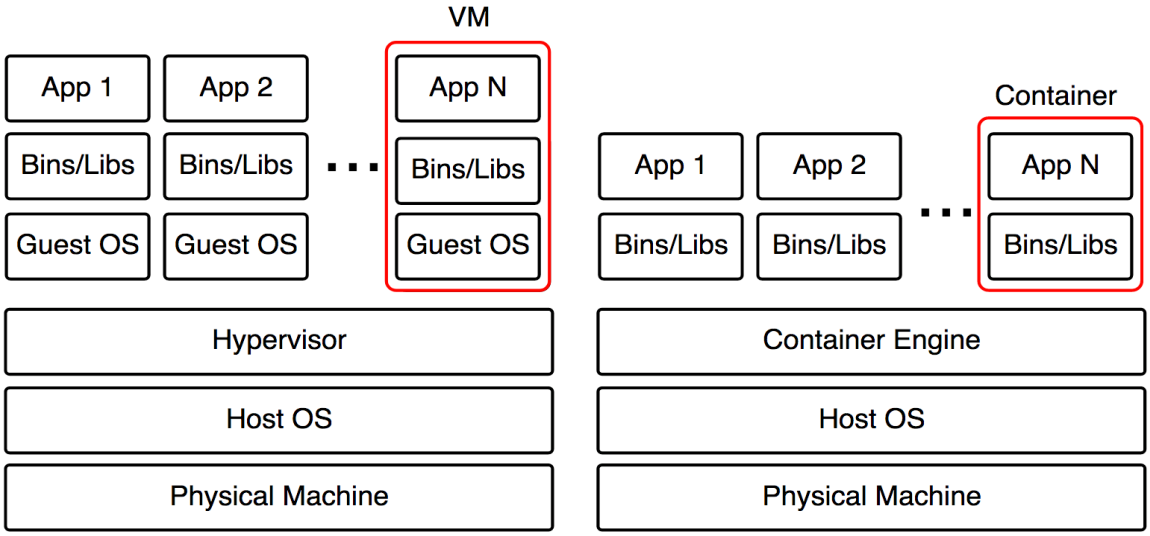

Figure 4. Architecture comparison between virtual machine and container.

There are three major differences between them: 1) On the same physical machine, different virtual machines own their separate guest operating systems while different containers share the same operating system (Host OS). Thus, a container is a more lightweight solution compared with a virtual machine via getting rid of these separate guest operating systems. 2) The hypervisor layer, such as VMware ESXi [55], VirtualBox [56], Xen [57] and KVM [58], is essential to a virtual machine while is not needed by a container. It also called virtual machine monitor (VMM), is a platform virtualization software runs on a physical host machine to enable multiple guest operating systems running concurrently. 3) Each virtual machine has its own image file and these image files are isolated from each other, while containers image files are created layer by layer which means containers may share some of their image files.

2.5.4 Docker and Docker container image

Docker [12] is currently the most popular Linux container that provides an easy-to-use container usage interface. Docker packages the application's dependencies with the program in a single file. Running this file will generate a virtual container. The program runs in this virtual container as if it were running on a real physical machine. With Docker, you don't have to worry about environmental issues. Overall, Docker's interface is fairly simple, and users can easily create and use containers and put their own applications into containers. Containers can also be versioned, copied, shared, and modified just like managing common code.

Docker packages application and its dependencies in an image file. Only through this file can the Docker container be generated. The image file can be thought of as a template for the container. Docker generates an instance of the container from the image file. The same image file can generate multiple container instances running at the same time.

Image is a binary file. In actual development, an image file is often generated by inheriting another image file, plus some personalization settings. For example, you can add an Apache server to your Ubuntu image to form your image. The image file is generic. If the image file of one machine is copied to another machine, it can still be used. In order to facilitate sharing, after the image file was created, it can be uploaded to the online warehouse. Docker's official repository Docker Hub is the most important and most common image repository.

2.5.5 Main uses of Docker

The main uses of Docker are currently in three categories.

(1) Provide a one-time environment. For example, testing other people's software locally, providing unit testing and build environment for continuous integration.

(2) Provide flexible cloud services. Because the Docker container can be opened and closed, it is suitable for dynamic expansion and shrinkage.

(3) Forming a microservices architecture. With multiple containers, one machine can run multiple services, so the microservices architecture can be simulated on this machine.

2.5.6 Docker in GATK Best Practices

In latest GATK Best Practices, GATK4, Picard, SAMtools and Python2.7 are called as docker images which are declared on each pipeline's input json files. Therefore, users do not need to configure the environment required by each tool, only need to configure the environment for Cromwell. This greatly simplifies the environment configuration of running scripts.

3. GATK BEST PRACTICES WORKFLOW

The two most important targets of this thesis are: 1) Enabling the pipeline implementation of GATK Best Practices workflow on CSC cPouta IaaS Cloud. 2) Collecting performance benchmarking data. Completed tasks contain automating the adjusted GATK Best Practices for germline short variants (SNPs and Indels) discovery with Docker container in three common widely used human whole genome sequence data (NA12878, NA12891 and NA12892) which sequenced by Illumina Cambridge Ltd. and download from European Nucleotide Archive (ENA) [59] and collecting necessary performance benchmarking data (such as multi-threads control, needed swap space, tools running time, PairHMM scalability in GATK4 HaplotypeCaller and so on). The GATK Best Practices Workflows are constructed as WDL scripts and run on Cromwell. This paper can be used as a guide to implement efficiently GATK Best Practices on CSC cPouta.

3.1 Flow of the single sample calling and joint calling pipelines

The original pipeline provided by Broad Institute for Data Pre-processing of each sample starts from a list of unmapped bam (ubam) files which are separated based on Read Group Tags (which are indicated as the flowcell:lane ID in FASTQ files), and ends with a BAM file which composed with Analysis-Ready reads of that sample. This analysis-ready BAM file will be passed to the following HyplotypCaller pipeline as input, applied GATK4 HaplotypeCaller with -ERC GVCF mode, then generate one GVCF file for each sample. In this thesis, the above steps were performed to each of three human whole-genome sequence samples (NA12878, NA12891 and NA12892) and generated three GVCF files, separately. Finally, I adopted multiple-sample variant-calling strategy to increase the sensitivity of GATK, made cohort analysis with all three GVCF files. I passed them simultaneously to Joint Calling pipeline, eventually got one VCF file which contains all genotypes for three samples at all sites.

Broad Institute prefers uBAM file format over FASTQ. The uBAM files are generated directly from the Illumine basecalls in Broad Institute instead of FASTQ files which are commonly generated by other sequencing providers, in order to reserve all metadata together with sequence reads. Thus, pipelines offered by Broad Institute will not write FASTQ files. However, FASTQ files still were common widely used by data pre-processing workflows provided by company and research institutes except Broad and most of sequencing providers generate FASTQ files with the raw unmapped read sequences.

Here, to ensure these users who only own raw sequence data in paired-end FASTQ format could utilize this workflow for germline short variants discovery as well, I added two pipelines in the beginning. Firstly, Linux Command Line Pipeline was used to divide paired-end FASTQ files into a set of FASTQ files according the flowcell:lane ID (which will be indicated as RG tags in uBAM or BAM files). Secondly, Sequence Format Conversion Pipeline was utilized to transfer FASTQ files to a list of unmapped bam (ubam) files, which can be passed to Data Pre-processing Pipeline. In addition, besides the main outputs mentioned above, each pipeline produce other files as well, such as index files, csv files and so on.

Figure 5 and 6 combined show our adjusted GATK Best Practices Germline Short Variant Discovery workflow which contains 5 pipelines in total.

These 5 pipelines are:

1. Linux Command Line Pipeline
2. Sequence Format Conversion Pipeline
3. Data Pre-processing Pipeline

4. HaplotypeCaller Pipeline
5. Joint Calling Pipeline

Figure 5 shows pipelines from No.1 to No.4 which describes the whole germline short variant single calling workflow, from FASTQ raw sequence data to a GVCF file generated by HaplotypeCaller and Figure 6 shows pipeline No.5, Joint Calling Pipeline which describes the cohort analysis of all three samples. In two figures, the documents of inputs/outputs are colored orange and the processing pipelines are colored light blue and text with bold.

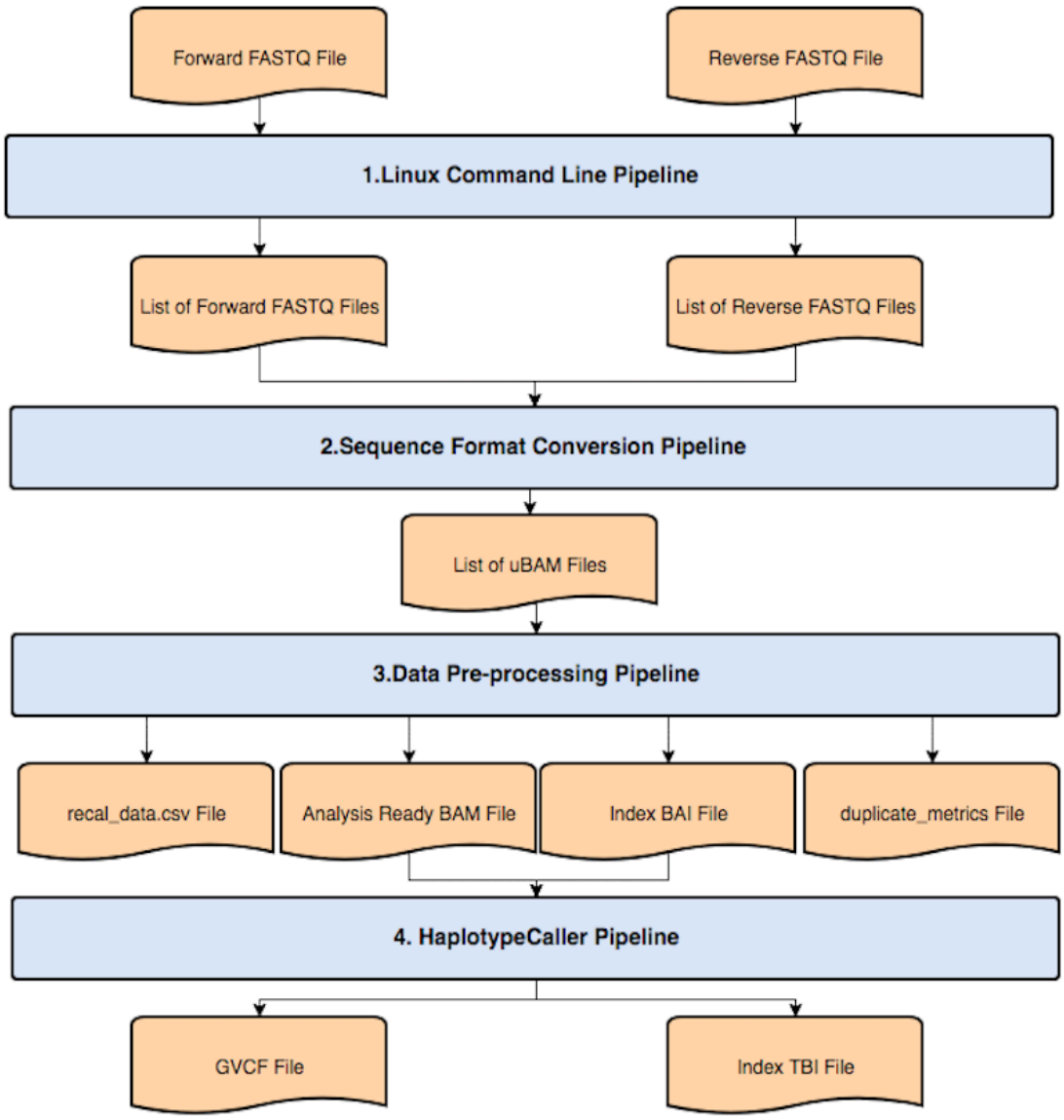

Figure 5. Overview of the single sample calling workflow.

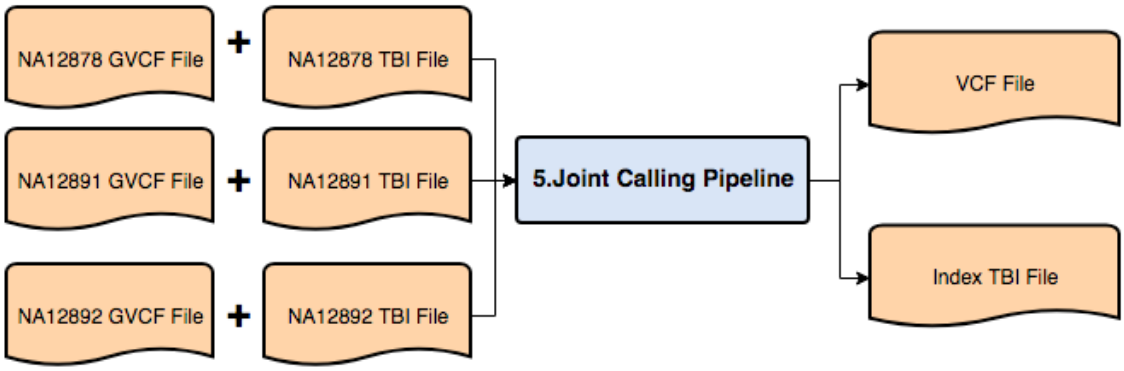

Figure 6. Overview of the Joint Calling Pipeline.

3.2 Arguments, methods and algorithms for the pipelines

This section introduces mainly applied methods, algorithms and tools with its arguments in the adjusted GATK Best Practices of this thesis. Section 3.2.1 describes the major steps for Data Pre-processing pipeline which is the essential pipeline for any variant calling workflows. Its target is producing analysis-ready BAM file for each sample from raw sequencing data (FASTQ or uBAM file). Section 3.2.2 describes HaplotypeCaller Pipeline with ERC GVCF mode. Section 3.2.3 describes GVCFs consolidating process which aims to increase the scalability and speed of later process. Section 3.2.4 describes Joint Calling Pipeline which used for cohort analysis to increase discovery rate. Section 3.2.5 describes variant filter and algorithm used in this thesis. Arguments for main tools involved in described pipelines are listed in Appendix A.

3.2.1 Map to Reference, Mark Duplicates and Base Quality Score Recalibration

Data pre-processing is an obligatory initial phase for any variant calling workflows. It contains three common steps: (1) Mapping reads to reference genome to produce a BAM or SAM file which sorted by coordinate aiming at providing a common coordinate framework for later analysis; (2) Marking duplicates to reduce the biases caused by DNA library preparation step such as PCR amplification; (3) Applying base quality score recalibration to increase sensitivity and specificity, because the GATK variant calling algorithm relies heavily upon each base quality of reads. In addition, at the very first step, the raw sequencing data must has already passed the quality control checks.

There are major 7 steps, in the data pre-processing pipeline of this thesis.

1. I got the BWA version which will be wrote in the PG record of the header of BAM file that produced by next step with GATK MergeBamAlignment tool.
2. The Picard SamToFastq tool combined with bwamem and samtools were used to convert reads from uBAM format to FASTQ format and mapped to the reference genome. Then, GATK MergeBamAlignment tool was used to merge the original uBAM and BWA-aligned BAM files. This step was performed per-read group and the flowcell-level uBAM input files were aligned paralleled via scatter-gather function.
3. GATK MarkDuplicates, SortSam and SetNmAndTags tools were used to mark duplicate reads, sort BAM file by coordinate order and fix tag values for NM and UQ.
4. Python was used to create sets of intervals for later scatter-gather paralleling over chromosomes and output to a stdout where it was parsed into a WDL Array[Array[String]].

5. GATK BaseRecalibrator tool was used to generate the recalibration model and this step was performed paralleled over chromosomes via scatter-gather function based on last step´s output.
6. GATK GatherBQSRReports tool was used to merge multiple recalibration report tables from last step scattered BaseRecalibrator runs. Then, the GATK ApplyBQSR tool was used to apply the Base Quality Score Recalibration model by interval parallelly via scatter-gather function.
7. Finally, the GATK GatherBamFiles tool was used to combine multiple recalibrated BAM files from last step scattered ApplyRecalibration runs.

Base Quality Score Recalibration (BQSR) and involved tools: BaseRecalibrator and ApplyBQSR

GATK variant calling algorithm accuracy is heavily dependent on the base quality score. However, since biases are introduced by the sequencing process such as sequencing reaction situations or sequencer itself, the sequencer assigned base quality score is not accurate enough. In order to eventually improve variant calling accuracy, BaseRecalibrator [60] utilizing machine learning methods [61] to build correction model which can intelligently adjust estimated base quality score.

Base Recalibration contains two key steps and one optional step.

1. Build a model of covariation based on the read data and sets of known variants (such as dbSNP, ExAc or GnomAD resource) to generate a recalibration table. If there are no known variants yet, bootstrap will be used [62]. First, the unrecalibrated data will be used to do the first round of SNP calling. Then, the found SNPs with the highest confidence will be used as the database of known SNPs and sent to BaseRecalibrator. Finally, do SNP calling with the recalibrated data. Users can repeat above steps until convergence.
2. Adjust the base quality score based on that model. ApplyBQSR produces recalibrated BAM or CRAM files based on previously generated recalibration table.
3. (optional) Build a second model to visualize this base recalibration process via plotting before and after plots.

It is worth mentioning that read filters such as: MappingQualityZeroFilter, MalformedReadFilter, BadCigarFilter, NotPrimaryAlignmentFilter, FailsVendorQualityCheckFilter, DuplicateReadFilter and MappingQualityUnavailableFilter have been automatically applied to the data by engine before BaseRecalibrator.

3.2.2 Call Variants Per-sample

Firstly (optional), the script checked the input was cram or not, if it was cram file, samtools was used to convert cram to bam. Then, variants were called in parallel over grouped calling intervals via scatter-gather function. The GATK HaplotypeCaller –ERC GVCF mode was used to generate GVCF by interval. Finally, all these per-interval GVCFs were merged by Picard MergeVcfs tool.

HaplotypeCaller Algorithm

HaplotypeCaller works in 4 steps.

- Define active regions.
- Reassemble each active region based on a De Bruijn-like graph to determine the possible haplotypes. Then, realigns these haplotypes against their reference haplotypes based on Smith-Waterman algorithm to identify potential variant sites.
- Determine likelihoods of previously found haplotypes according PairHMM algorithm.
- Finally, assign genotypes for sample via applying Bayes´rule on each potential variant site.

Generally speaking, by introducing local de-nova reassembling of haplotypes in active regions, HaplotypeCaller is able to efficiently and more accurately call SNPs and Indels at the same time. HaplotypeCaller has normal VCF mode and so called GVCF mode. In this thesis, I deployed HaplotypeCaller with –ERC GVCF mode. This mode generated intermediate GVCF instead of normal VCF files. These GVCF files were jointly passed to Joint Calling pipeline for cohort analysis.

3.2.3 Consolidate GVCFs

Before Joint calling process, the GATK GenomicsDBImport tool was used to consolidate the three single-sample GVCF files into a datastore: GenomicsDB datastore. This step can increase the scalability and speed of the upcoming joint genotyping process. Arguments for tool GenomicsDBImport are listed below. All bash variables needed for the command line have been set before.

3.2.4 Joint-Call Cohort

In the Joint Calling Pipeline, the GATK GenotypeGVCFs tool was used to perform joint genotyping on three previous generated GVCF files from sample: NA12878, NA12891 and NA12892 which all stored in a GenomicsDB workspace. Finally, generated a set of SNPs and indels variants waiting for filtering.

3.2.5 Filter Variants by Variant (Quality Score) Recalibration

Joint Calling Pipeline can be generally split into two parts: 1) Joint genotyping with tools GenomicsDBImport and GenotypeGVCFs and 2) Variant Quality Score Recalibration (VQSR) with tools: VariantRecalibrator and ApplyVQSR. When variant recalibration cannot be performed, hard-filtering will be used to instead. Filtering (via VQSR or hard-filtering) is an important and undeletable step for the whole Variant Calling Workflow which can significantly reduce false positives.

Variant Quality Score Recalibration (VQSR) and involved tools: VariantRecalibrator and ApplyVQSR

The target of variant quality score recalibration (VQSR) is to assign a well-calibrated probability, called VQSLOD score, to each variant call inside of a call set. Then, filter variants based on the VQSLOD score of each variant call to significantly reduce the number of false positives. VQSR is a two-step process. The first step is completed by tool VariantRecalibrator and the second step is completed by tool ApplyVQSR. First, VariantRecalibrator uses machine learning method to build a filtering model based on these high-accuracy known variant sites, typically are HapMap sites [63] or polymorphic sites found in human omni 2.5M SNP Chip Array [64]. Then, this model is applied to all input variants to evaluate the probability of each variant is true and generate VQSLOD score for each variant. Eventually, this tool will produce two output files: a recalibration table file which will be used in the second step and a tranches file which shows various metrics of the recalibration callset for slices of the data. Second, ApplyVQSR filters input variants based on the recalibration table and a given target sensitivity value. This filtering is not a simple “pass or fail” process, but assigning different level annotation to each slice of the dataset, called tranche, according its truth sensitivity and “filter” in this case does not means removed but marked as filtered.

3.3 On the Parallelism of the pipeline

This section introduces parallel computing and implementation in GATK Best Practices.

3.3.1 Parallel Computing

Parallel computing refers to using multiple processing units to perform several operations at the same time in parallel, instead of sequentially i.e one after another. The basic idea is to use multiple processors to solve the same problem collaboratively which means the problem or computing task need to be decomposed into several parts that can be done independently. Each part will be calculated in parallel by different processors.

Although parallel computing is an effective way to improve the computing speed and processing power of computer system, it has "overhead" costs as well. Enabling parallel computing will introduce additional work, such as file access management, dividing jobs and collecting results. Thus, it is important to keep the balance between benefits and costs avoiding unnecessary parallelism or too many parallel branches. A parallel computing system can be either a computer with multiple processors/cores shared memory parallelism or a cluster of processing units interconnected distributed memory parallelism.

3.3.2 Parallelizing the GATK

Parallelism can be implemented in GATK via multi-threading or via the scatter-gather function.

Multi-threading and Spark

A **process** is the instance of a computer program which is being excuted by one or multiple threads. A **thread** is a unit of execution and execution scheduling of a process. Threads depend on the existence of processes. Under the process, you can share the memory of the process, and also have a memory space of your own. This memory space is also called the thread stack. It is allocated by the system when the thread is created. It is mainly used to save the data used inside the thread. **Multi-process** means different programs or replicates of the same program running at the same time. **Multi-threading** means that there are multiple threads under one process. Each thread executes any part of this process code, including parts currently being executed by another thread. The comparision between thread and process is showed in Figure 7.

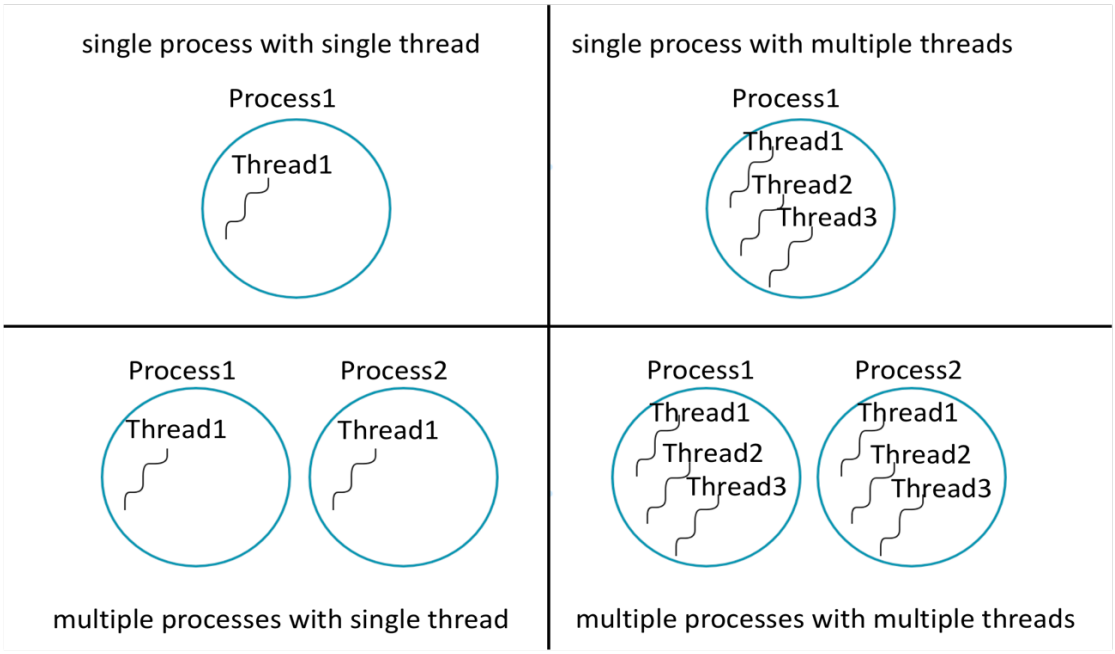

Figure 7. Process vs. Thread.

For some tools of GATK3 with built-in multi-threading option, it can be easily enabled by arguments: `-nt` or `–nct`. “nt” stands for number threads which controls the number of data threads sent to the processor (machine level). “nct” stands for number CPU threads which controls the number of CPU threads allocated to each data thread (core level).

In GATK4, an open-source software library called **Spark** [13] is used to enable multi-threading. Many GATK4 tools are Spark-capable but not all of them and these Spark-capable tools are still in beta version which are not suitable for production yet. Users can use Spark without additional installations because all necessary dependencies needed to use Spark already are bundled by GATK4 itself. Spark-enabled GATK4 tools can be run on both a local multi-core machine and on a Spark cluster. Example command lines are as follow. 1. “`--spark-master local[n]`”. It means run on a local machine with n cores. “n” can be replaced with “*” which means using all cores. 2. “`--spark-master spark://23.195.26.187:7077`”. It means run on a Spark cluster at 23.195.26.187, port 7077. 3. “`--spark-runner GCS --cluster my_cluster`”. It means run on my_cluster in Google Dataproc.

The performance of GATK4 can be improved by involving the Intel Genomics Kernel Library (GKL) [65]. GKL is an open-source collection of optimized components used in genomics applications which was developed by Intel and Broad Institute. In the case of hardware compatibility, it can provide alternatives to the key components of the GATK4 toolkit. Alternatives can significantly improve the speed of operation and optimize algorithms. For example, in HaplotypeCaller tool, alternative code adds Open Multi-Processing (**OpenMP**) [66] support for multi-threaded control. More detail information and testing data is showed in section 4.3. OpenMP is a portable application program interface (API) to direct multi-threaded, shared memory parallelism. It is composed by three basic API components: compiler directives, runtime library routines and environment variables.

Scatter-Gather

Multi-threading parallelism takes place within one process, the scatter-gather approach involves multiple processes. There are two pipelining methods to enable scatter-gathering in GATK. The GATK-Queue and combination of Cromwell with WDL script. GATK-Queue is a command-line job manager and scripting framework for multi-stage genomic analysis. According Broad Institute document [67], GATK support for Queue was permanently discontinued upon release of GATK 4. The one applied in this thesis is the combination of Cromwell with WDL script. In scatter step of the scatter-gather, Cromwell generates separate command lines based on a given WDL script. These command lines will run that tool on certain portion of the input data and produce their results separately. All the results will be stored in temporary files. In the gather step, Cromwell collects all the results into a final output file. Figure 8 shows a scatter-gather parallelism example and descripted below.

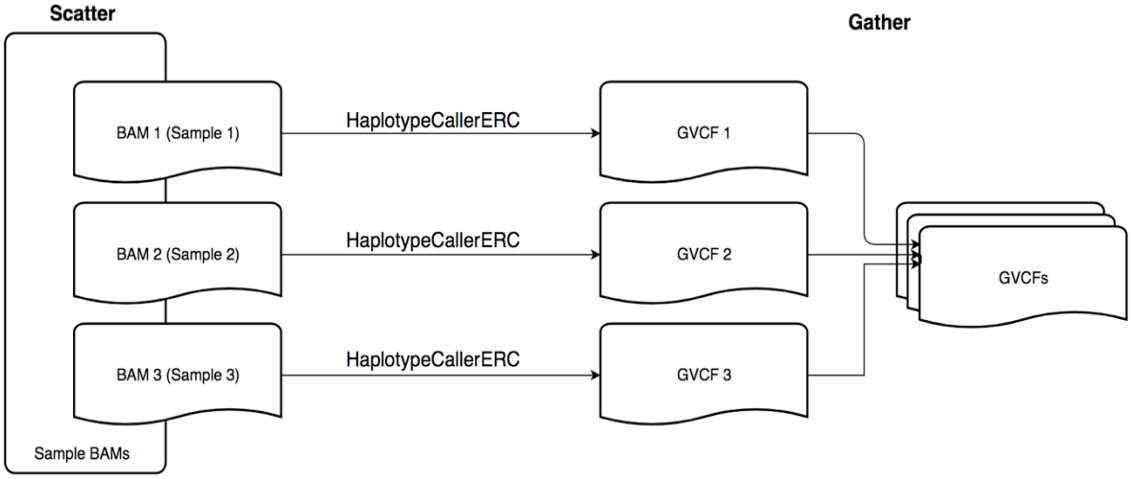

Figure 8. Example of Scatter-Gather.

In this example, I have sets of sample BAM files. Instead of direct passing all of them into tool HaplotypeCaller (ERC mode), I firstly scattered them into 3 separate BAM files (BAM1, BAM2 and BAM3). Each of BAMs went through tool HaplotypeCaller (ERC mode) and produced its GVCF file. Then, all three GVCFs were gathered into one final output file, typically as an array file. It is worth mentioning that while using scatter-gather function, users still can use GATK´s internal multithreading capabilities inside of each scatter branches (nodes) to increase the benefits of parallelism.

3.4 Experimental Setup

Hardware Specifications

All experiments and testing were conducted on cPouta Virtual Machine provided by CSC. The Virtual Machine flavor used in this experiment was hpc-gen2.24core. In this flavour, CPU was Intel Xeon CPU E5-2680 v3, with hyper-threading, VCPUs was 24 cores. RAM is 117.2 GB, nodes ran CentOS 7 system and connected with 40 Gb/s Ethernet. Root Disk was 80 GB with 3 attached volumes: volume t-vol1 attached on /dev/vda, 900 GB. volume t-vol2 attached on /dev/vdb, 4000GB. volume t-vol3 attached on /dev/vdc, 3500 GB. The instance featured a local SATA disk, no RAID. Swap space was 99 GB. According testing results, swap space is essential for running GATK4 Best Practices but the size of swap space can be adjusted according user´s analysis scale.

Software Requirements

Docker, docker-ce-17.06.0.ce-1.el7.centos.x86_64.rpm was downloaded from docker-ce version for centos 7 download page (`https://download.docker.com/linux/centos/7/x86_64/stable/Packages/`). Cromwell, Cromwell-33.1.jar was downloaded from broadinstitute/cromwell GitHub page (`https://github.com/broadinstitute/cromwell/releases`). GATK4, Samtools and Python2.7 called as Docker images which are declared on each pipeline's input json files.

3.5 Genomes Datasets

Single sample-calling workflow starts from a FASTQ file and ends at a generated GVCF file. It contains 4 pipelines: Linux Command Line Pipeline, Sequence Format Conversion Pipeline, Data Pre-processing Pipeline and HaplotypeCaller Pipeline. The Paired-End FASTQ files used by this thesis (sample NA12878, NA12891 and NA12892 [59], WGS, 30X sequencing coverage) were downloaded from European Nucleotide Archive (ENA) [68]. Sample NA12878 came from a Utah woman who had a genetic disease (CYP2D6 mutation) and her parents´ samples are NA12891 (father) and NA12892 (mother). NA12878 is the most commonly used testing data and the three members of CEU trio (NA12878, NA12891 and NA12892) are the recommend benchmarking data for GATK Best Practices by Broad Institute. All detailed information of WGS datasets for Linux Command Line Pipeline and Sequence Conversion Pipeline are mentioned in the Table 3 and Table 4. The uBAM files used for Data Pre-processing Pipeline are listed in Table 5. The datasets used by the HaplotypeCaller Pipeline and the Joint Calling pipeline are listed in Table 6. These three GVCF files used by Joint Calling Pipeline are generated by passing the Paired-End FASTQ files through the single sample calling workflow. The last tool HaplotypeCaller in single sample calling workflow produced one GVCF and its index file for each sample. Table 7 lists reference files and know sites resources for single sample calling workflow and Joint Calling Pipeline.

Table 3. *WGS Dataset for Linux Command Line Pipeline.*

Se-quence type	Sample	Library Name	Plat-form_unit	File Used	Size(GB)	X Cover-age	Read Length
Input WGS Da-taset	NA12878	ERR194147	HSQ1004:134	ERR194147_1.fastq.gz ERR194147_2.fastq.gz	47.9 48.9	30X	100
	NA12891	ERR194160	HSQ1008:175	ERR194160_1.fastq.gz ERR194160_2.fastq.gz	46 48	30X	100
	NA12892	ERR194161	HSQ1008:176	ERR194161_1.fastq.gz ERR194161_2.fastq.gz	51 52	30X	100

Table 4. *WGS Dataset for Sequence Format Conversion Pipeline.*

Sam-ple_Name	Read-Group_Name	Fastq_1	Fastq_2	Li-brary_name	Plat-form_name	Plat-form_unit	Size(GB)
NA12878	ERR194147_C0D8DACXX.1	ERR194147_C0D8DACXX.1_1.fastq	ERR194147_C0D8DACXX.1_2.fastq	ERR194147	Illu-mina	HSQ1004:134	
	ERR194147_C0D8DACXX.2	ERR194147_C0D8DACXX.2_1.fastq	ERR194147_C0D8DACXX.2_2.fastq				
	ERR194147_C0D8DACXX.3	ERR194147_C0D8DACXX.3_1.fastq	ERR194147_C0D8DACXX.3_2.fastq				
	ERR194147_C0D8DACXX.4	ERR194147_C0D8DACXX.4_1.fastq	ERR194147_C0D8DACXX.4_2.fastq				
NA12891	ERR194160_C0JVFACXX.5	ERR194160_C0JVFACXX.5_1.fastq	ERR194160_C0JVFACXX.5_2.fastq	ERR194160	Illu-mina	HSQ1008:175	
	ERR194160_C0JVFACXX.6	ERR194160_C0JVFACXX.6_1.fastq	ERR194160_C0JVFACXX.6_2.fastq				
	ERR194160_C0JVFACXX.7	ERR194160_C0JVFACXX.7_1.fastq	ERR194160_C0JVFACXX.7_2.fastq				
	ERR194160_C0JVFACXX.8	ERR194160_C0JVFACXX.8_1.fastq	ERR194160_C0JVFACXX.8_2.fastq				

NA12891	ERR194161_D0UYCACXX.4 ERR194161_D0UYCACXX.5 ERR194161_D0UYCACXX.6 ERR194161_D0UYCACXX.7	ERR194161_D0UYCACXX.4_1.fastq ERR194161_D0UYCACXX.5_1.fastq ERR194161_D0UYCACXX.6_1.fastq ERR194161_D0UYCACXX.7_1.fastq	ERR194161_D0UYCACXX.4_2.fastq ERR194161_D0UYCACXX.5_2.fastq ERR194161_D0UYCACXX.6_2.fastq ERR194161_D0UYCACXX.7_2.fastq	ERR194161	Illumina	HSQ1008:176	

Table 5. *WGS Dataset for Data-Processing Pipeline.*

File Type	Sample	File Used	Size(GB)
txt	NA12878	NA12878_unmapped_bam.list	312 KB
uBAM		ERR194147_C0D8DACXX.1.unmapped.bam	33
uBAM		ERR194147_C0D8DACXX.2.unmapped.bam	33
uBAM		ERR194147_C0D8DACXX.3.unmapped.bam	33
uBAM		ERR194147_C0D8DACXX.4.unmapped.bam	33
txt	NA12891	NA12891_unmapped_bam.list	312 KB
uBAM		ERR194160_C0JVFACXX.5.unmapped.bam	31
uBAM		ERR194160_C0JVFACXX.6.unmapped.bam	32
uBAM		ERR194160_C0JVFACXX.7.unmapped.bam	32
uBAM		ERR194160_C0JVFACXX.8.unmapped.bam	32
txt	NA12892	NA12892_unmapped_bam.list	312 KB
uBAM		ERR194161_D0UYCACXX.4.unmapped.bam	35
uBAM		ERR194161_D0UYCACXX.5.unmapped.bam	35
uBAM		ERR194161_D0UYCACXX.6.unmapped.bam	34
uBAM		ERR194161_D0UYCACXX.7.unmapped.bam	35

Table 6. *Input Datasets for HaplotypeCaller Pipeline and Joint Calling Pipeline.*

File Type	Sample	File Used	Size(GB)
Analysis-Ready Bam Files	NA12878	NA12878.hg38.bam	66
Analysis-Ready Bam Files	NA12891	NA12891.hg38.bam	63

Analysis-Ready Bam Files	NA12892	NA12892.hg38.bam	67
GVCF Files	NA12878	NA12878.hg38.g.vcf.gz	2.7
GVCF Files	NA12891	NA12891.hg38.g.vcf.gz	3.1
GVCF Files	NA12892	NA12892.hg38.g.vcf.gz	2.1

Table 7. *Reference Files and Known Sites Resources for Single Sample Calling Workflow and Joint Calling Pipeline.*

File Type	File Used	Size(GB)
Reference Genome Fasta	hg38_v0_Homo_sapiens_assembly38.fasta	3.1
dbSNP VCF Files	hg38_v0_Homo_sapiens_assembly38.dbsnp138.vcf	11
Known Indels Sites VCFs	hg38_v0_Homo_sapiens_assembly38.known_indels.vcf.gz	0.06
JointGenotyping.one_thousand_genomes_resource_vcf	hg38_v0_1000G_phase1.snps.high_confidence.hg38.vcf.gz	1.8
JointGenotyping.omni_resource_vcf	hg38_v0_1000G_omni2.5.hg38.vcf.gz	0.05
JointGenotyping.mills_resource_vcf	hg38_v0_Mills_and_1000G_gold_standard.indels.hg38.vcf.gz	0.02
JointGenotyping.axiomPoly_resource_vcf	hg38_v0_Axiom_Exome_Plus.genotypes.all_populations.poly.hg38.vcf.gz	0.003
JointGenotyping.hapmap_resource_vcf	hg38_v0_hapmap_3.3.hg38.vcf.gz	0.06

4. BENCHMARKS

I collected 4 pipelines execution time(s) with 3 different datasets, compared the execution time(s) with Spark local runner enabled or not in two tools: SortSam(Spark) and MarkDuplicates(Spark). In addition, I explored the GATK4 PairHMM threads scalability in tool HyplotypeCaller, and investigated the GATK4 thread scalability for Java parallel garbage collection in tool MarkDuplicates.

4.1 Workflow Execution Time(s) for WGS Samples

I tested 4 pipelines (Sequence Format Conversion, Data Pre-processing, HaplotypeCaller and Join Calling pipelines) with three Whole Genome Sequence (WGS) data: NA12878, NA12891 and NA12892 and collected their baseline execution time(s). The bwa mem tool involved in Data Pre-processing pipeline was enabled to execute 16 threads with command line "`bwa mem -K 100000000 -p -v 3 -t 16 -Y $bash_ref_fasta`". The other tools are all deployed with default value which means single-threaded. Testing results are listed in Table 8 and Figure 9. Time(Real) is wall clock time. It is the time from start to finish of the call. Time(User) is the amount of CPU time spent in user-mode code (outside the kernel) within the process. It is the actual CPU time used to execute the process. Time(Sys) is the amount of CPU time spent in kernel within the process.

Comparing the execution time(s) for three different WGS samples with similar characteristics, pipelines execution time(s) showed consistency. The Real Execution Time(s) to complete pipelines in three different datasets were very close, for Sequence Format Conversion pipeline took around 1h23mins (mean), for Data Pre-Processing pipeline took around 34h54mins (mean), for HaplotypeCaller pipeline took around 4h26mins (mean), for Joint Calling pipeline took around 1h26mins and for the whole workflow took around 42h9mins (mean). The real execution time(s) for three datasets were slightly different because of small difference in dataset´s size and features. Generally speaking, total real execution time from FASTQ file to GVCF file (from Sequence Format Conversion to HaplotypeCaller pipeline), the smallest dataset NA12891 took shortest processing time 39h17m24.456s, the largest dataset NA12892 took the longest processing time 42h45m16.765s and mean processing time was around 40h42m48s.

The real execution time for similar WGS datasets with different size and features showed consistency and execution time and dataset size were roughly positive correlated.

***Table 8**. Execution Time(s) for Three WGS Samples.*

Pipeline	Sample	File Size(GB)	Time(Real)	Time(User)	Time(Sys)
Sequence Format Conversion	2xPaired_End_NA12878 Fastq.gz	96.8	1h24m13.258s	1m48.401s	0m14.658s
	2xPaired_End_NA12891 Fastg.gz	94	1h14m31.188s	1m40.021s	0m11.855s
	2xPaired_End_NA12892 Fastq.gz	103	1h30m21.144s	1m49.312s	0m19.364s

Data Pre-pro-cess-ing (16 threads for bwa mem)	List of NA12878 uBAMs	132	34h11m11.160s	42m31.043s	3m43.535s
	List of NA12891 uBAMs	127	33h26m51.594s	40m16.328s	3m30.946s
	List of NA12892 uBAMs	139	37h3m36.916s	43m32.385s	4m9.170s
Haplo-type-Caller	NA12878.hg38.bam	66	4h30m10.983s	13m14.295s	0m35.697s
	NA12891.hg38.bam	63	4h36m1.674s	12m6.269s	0m42.976s
	NA12892.hg38.bam	67	4h11m18.705s	11m41.254s	0m38.704s
Joint Calling	NA12878.bam.hg38.g.vcf.gz	2.7	1h26m15.202s	2m52.504s	0m22.295s
	NA12891.bam.hg38.g.vcf.gz	3.1			
	NA12892.bam.hg38.g.vcf.gz	2.1			

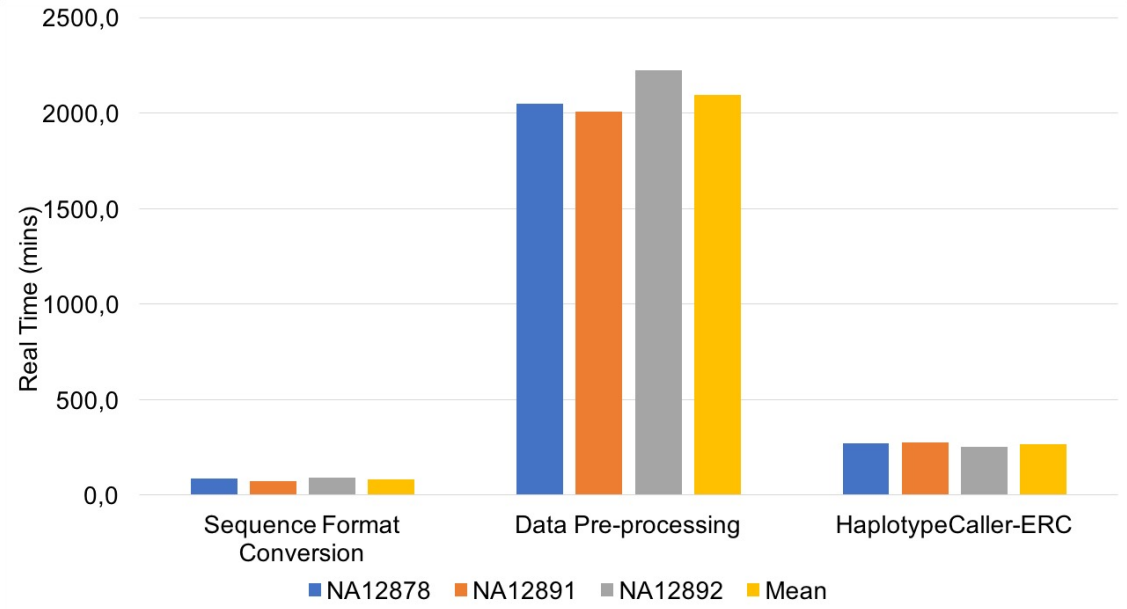

Figure 9. Real execution time(s) for NA12878, NA12891 and NA12892 WGS datasets in single calling pipelines (Sequence Format Coversion pipeline, Data Pre-processing pipeline and HaplotypeCaller in ERC mode pipeline). Mean real time stands for the mean real execution time of processing above three WGS datasets for each pipeline.

4.2 Single-Thread vs Parallelized Run

Unlike GATK3 which can enable multithreading on a multicore CPU via the –nt and –nct arguments, GATK4 was originally designed single-threaded. However, there are built-in multithreading

control arguments in the HaplotypeCaller tool. Its enabling method and testing result are showed in section 4.3. As discussed in section 3.3.2, some GATK4 tools have two versions: Spark-capable and non-Spark capable versions, such as: MarkDuplicates and MarkDuplicatesSpark. These Spark-capable tools with the “Spark” suffix and users better to invoke GATK using its wrapper script rather than directly calling the jar file, because the wrapper can automatically choose the right jar file and set appropriate parameters. Some tools only exist in spark versions, thus, there is no Spark suffix in their name. No matter which one, GATK Spark-capable tools can both run on local or with Spark Cluster. Once you have a machine with multiple CPU cores, you can easily run Spark-enabled GATK tools with arguments: `-- --spark-runner LOCAL --spark-master local[n]`.

Here, I locally enabled two Spark-capable tools in GATK4: SortSamSpark and MarkDuplicatesSpark and compared the real execution time(s) with their non-Spark version. I tested GATK 4.0.4.0 SortSam and SortSamSpark with 8 cores and 16 cores of local runner and GATK 4.1.0.0 MarkDuplicates and MarkDuplicatesSpark with 16 cores of local runner with and without writing metrics file (enabled and disabled by argument "`-M`"). The results are showed in Figure 10 and Figure 11.

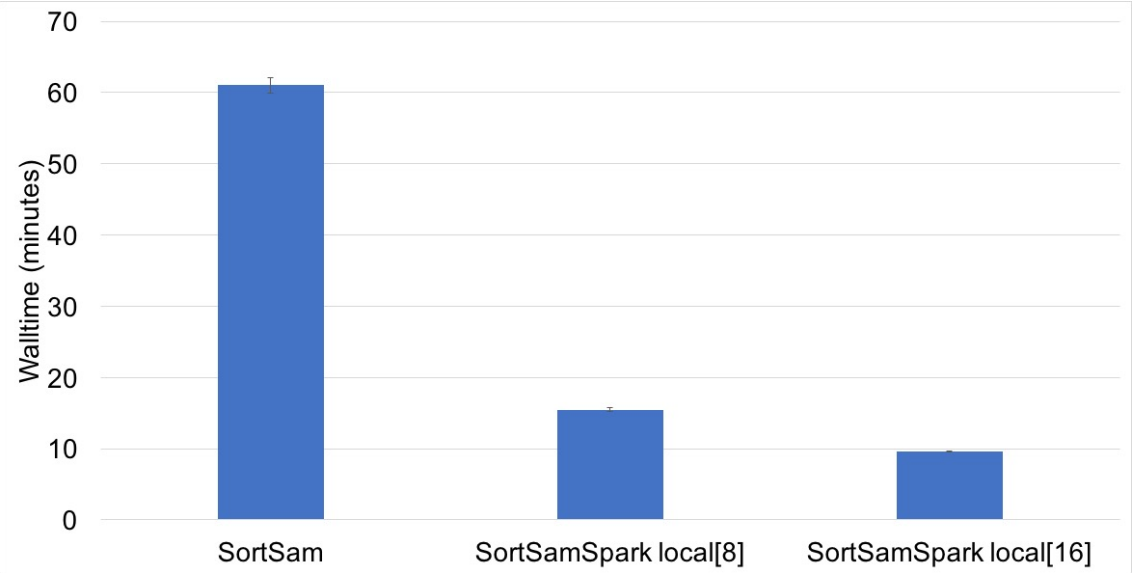

Figure 10. Real execution time(s) for GATK 4.0.4.0 SortSam and SortSamSpark enabled 8 cores and 16 cores Spark local runner. Testing sample for SortSam and SortSamSpark is ERR194147_C0D8DACXX.1.unmapped.aligned.unsorted.bam file which is the ERR194147_C0D8DACXX.1 unmapped bam file (listed in Table 5) aligned to reference genome via Best Practices workflows. Error bars denote one standard error around the mean of three runs.

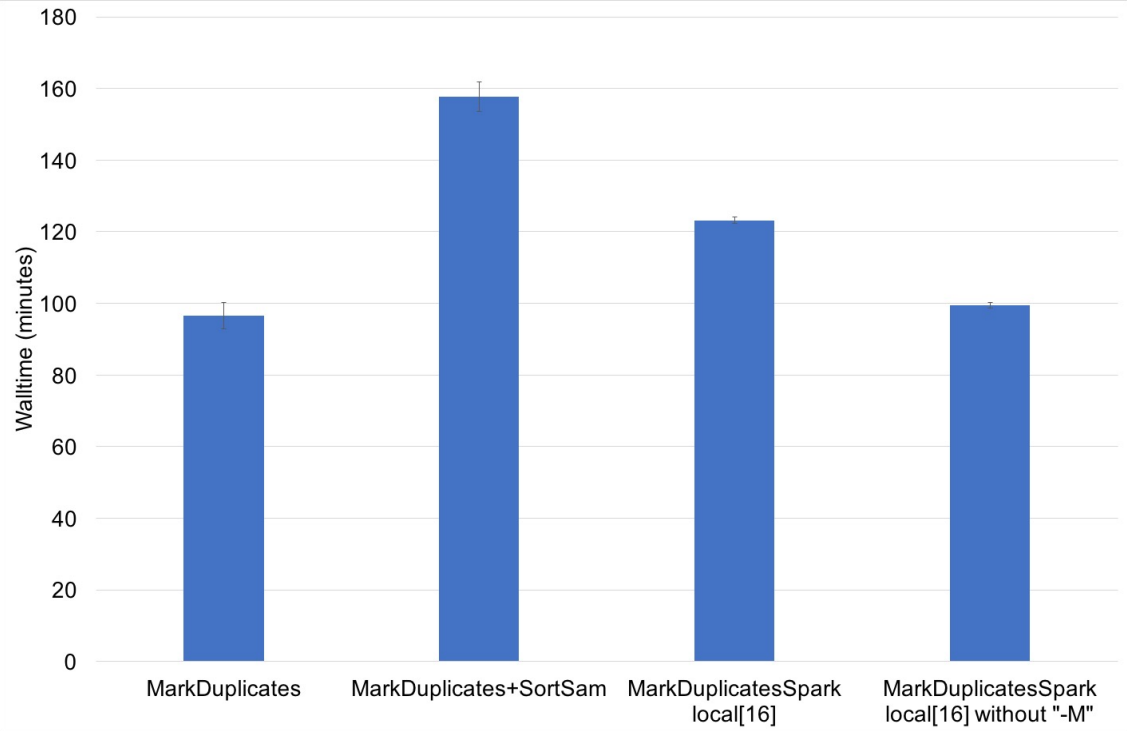

Figure 11. Real execution time(s) for GATK 4.1.0.0 MarkDuplicates, MarkDuplicates plus SortSam, MarkDuplicatesSpark enabled 16 cores Spark local runner with and without writing metrics file. Testing sample for MarkDuplicates and MarkDuplicatesSpark seperately is the output bam file of SortSam and SortSamSpark in Figure 10. Error bars denote one standard error around the mean of three runs.

GATK version 4.0.4.0 was the one I used for benchmarking the Best Practices workflows in this thesis. But MarkDuplicatesSpark of GATK 4.0.4.0 reported “Duplicate key -1” error and GATK team claimed that they have fixed this issue in version 4.0.5.0 [69]. However, MarkDuplicatesSpark of GATK 4.0.5.0 is still a beta version which is not suitable for production. Until GATK 4.1.0.0 version, MarkDuplicatesSpark become a newly out-of-beta replacement for the old MarkDuplicates plus SortSam steps of the Best Practices workflows [70]. The output of GATK 4.1.0.0 MarkDuplicatesSpark is not just marked duplicates but also coordinated sorted. Thus, I compared the execution time of MarkDuplicatesSpark with the execution time for MarkDuplicates plus SortSam. In addition, according Ref. [71], GATK team mentioned that the writing metrics file step is likely the bottlenecks and there is a standalone tool called EstimateLibraryComplexity which can be used to collect the exact same metrics file. Separating the duplicates marking and metrics file writing is a good choice to run tool MarkDuplicatesSpark more efficiently, special for these users who don’t need the metrics file. Thus, I collected the real execution time(s) for MarkDuplicatesSpark enabled local 16 cores with and without writing metrics file.

From Figure 10, I found that non-Spark version of SortSam cost longest execution time which was around 61 minutes and SortSamSpark with local 16 cores cost shortest execution time which was around 10 minutes showing 83.6% improvement. When I doubled the local cores from 8 to 16, the excution time decreased from around 15 minutes to 10 minutes showing 33.3% improvement.

From Figure 11, I found that non-Spark version MarkDuplicates cost shortest time which was around 97 minutes. However, as I mentioned above, GATK 4.1.0.0 MarkDuplicatesSpark is the replacement of the old MarkDuplicates and SortSam. I summed their execution times and compared it with the execution time of MarkDuplicatesSpark enabled local 16 cores with and without writing metrics file. Non-Spark version of MarkDuplicates plus SortSam cost the longest time which was around 158 minutes and MarkDuplicatesSpark enabled 16 cores cost around 123

minutes, showing 22.2% improvement. When I omitted the metrics file writing step, it cost only 99 minutes which showing 19.5% more speed-up.

4.3 PairHMM Scalability in GATK4 HaplotypeCaller

As discussed in section 3.3.2, Genomics Kernel Library (GKL) speeds up GATK4 analysis operation and optimizes algorithms. HaplotypeCaller is an essential tool of GATK4 for Best Practices. GKL's key points for optimizing the HaplotypeCaller tool are: Introduced the AVX optimized version of the PairHMM and Smith-Waterman algorithms, and added OpenMP support to the PairHMM algorithm for multi-threaded control [72]. AVX [65] is the abbreviation of Advanced Vector Extensions which refers to a set of instructions for doing 256-bits single instruction multiple data operations on Intel architecture CPUs. It improves processor's floating-point processing performance and scientific computing power. GATK4 has built-in pre-configuration to enable automatic detection for AVX hardware support. In this thesis, our testing cPouta Virtual Machine features Intel(R) Xeon(R) CPU E5-2680 v3, with hyper-threading and Intel AVX2. Users can easily enable multi-threading of PairHMM in HaplotypeCaller –ERC mode by adding two flags: "--pair-hmm-implementation AVX_LOGLESS_CACHING_OMP" and "--native-pair-hmm-threads n". “n” stands for the number of threads.

After testing, the optimal number of threads for GATK4 HaplotypeCaller –ERC mode seems to be 12. Compared with 1 threads, execution time reduced from 26.7 minutes to 23.4 minutes, 12.4% improvement. Detailed data are showed in Figure 12.

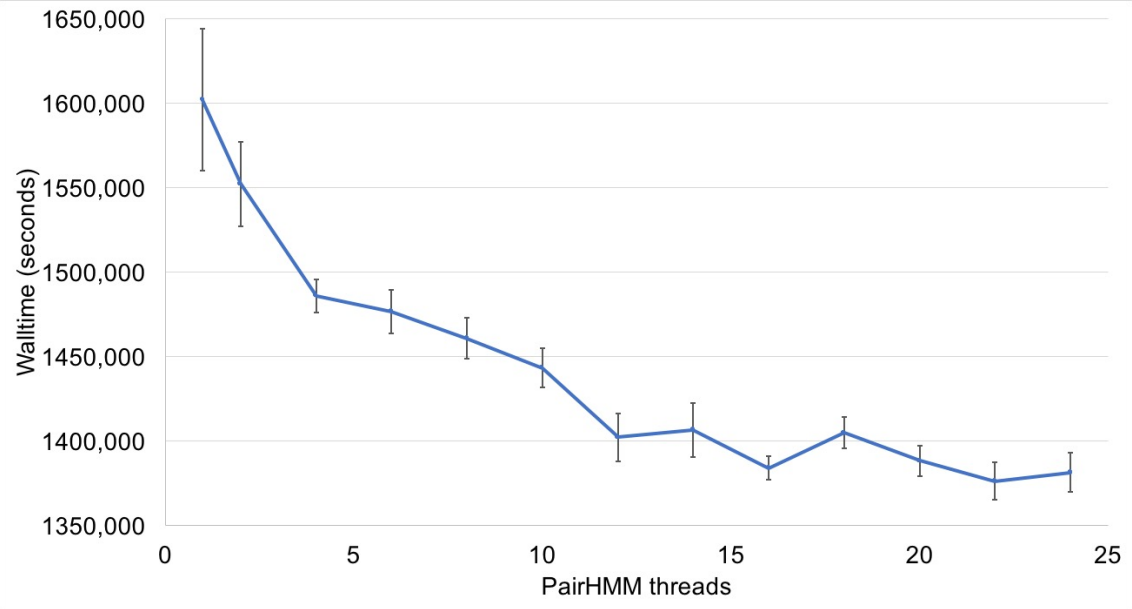

Figure 12. GATK4 thread scalability in HaplotypeCaller. The measurements at 1, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22 and 24 PairHMM thread were presented. Testing Sample is NA12878.hg38.bam -L chr20:1-25000000. The error bars denote one standard error around the mean of three runs.

4.4 GATK4 Parallel garbage collection

In Ref. [73], enabling Java Parallel Garbage Collection (PGC) in GATK3.7 reduced the tool execution time but did not show the same improvement in GATK3.8. Here, I enabled PGC in GATK4.0.4.0 MarkDuplicates tool with threads: 1, 2, 4, 8, 16 and 20 in our virtual machine and compared the execution time(s). The java option argument used to enable PGC is: --java-options

"-XX:+UseParallelGC -XX:ParallelGCThreads=n" and n stands for number of ParallelGC Threads. The results are showed in Figure 13.

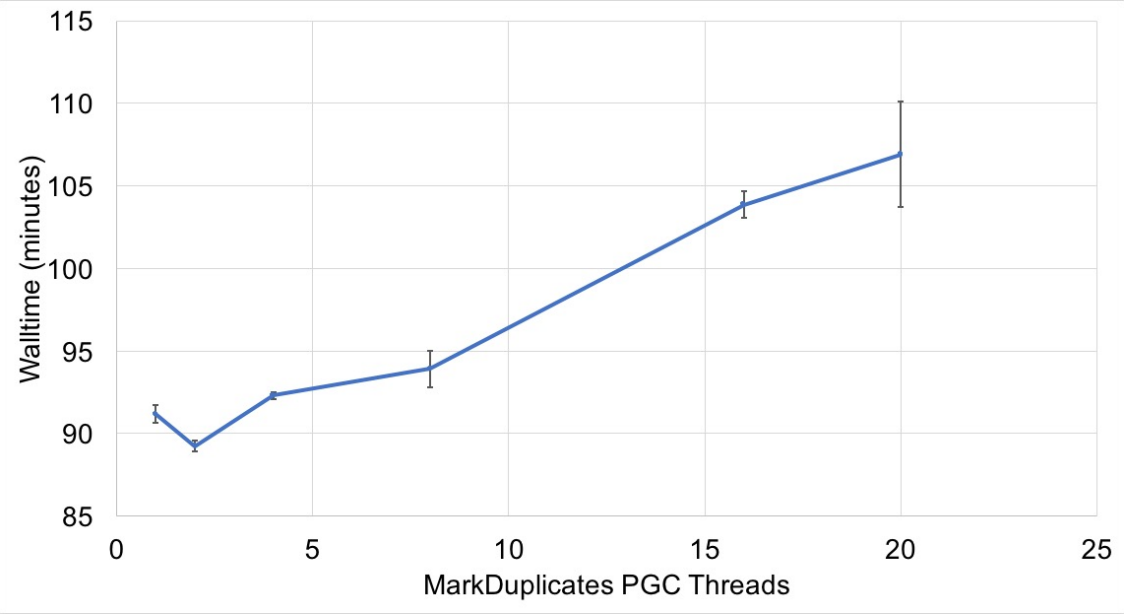

Figure 13. GATK4 MarkDuplicates scalability for Java Parallel Garbage Collection. The measurements at 1, 2, 4, 8, 16 and 20 PGC thread were presented. Testing sample is ERR194147_C0D8DACXX.1.unmapped.aligned.unsorted.bam which is the ERR194147_C0D8DACXX.1 unmapped bam file (listed in Table 5) aligned to reference genome via Best Practices workflows. The error bars denote one standard error around the mean of three runs.

I found that the number of optimal PGC threads was 2 for GATK4 tool MarkDuplicates giving rise to a speed-up of 2.1%. When enabled with PGC 2, the mean real execution time of three replicates for MarkDuplicates is 89.2 minutes which is the shortest. After that, the execution time was increasing with the number of PGC threads.

4.5 Summary of optimized parameter values

The tested optimizations of GATK4 tools in order to run Best Practices more efficiently in this thesis are listed in Table 9.

Table 9. *Summary of optimized parameters.*

Tool Name	Spark local runner	PGC threads	AVX threads
SortSamSpark	16	N/A	N/A
MarkDuolicatesSpark	16	N/A	N/A
MarkDuplicates	N/A	2	N/A
HaplotypeCaller	N/A	N/A	12

Among these optimizations, enabling multicore spark local runner for GATK4 Spark tools presented highest speed-up while enabling multiple PGC threads presented lowest speed-up. However, all the GATK4 spark tools are still in beta version, except the MarkDuplicatesSpark in GATK4.1.0.0 [70]. In addition, to efficiently ran the whole workflow of Best Practices for germline short variants (SNPs and Indels) discovery, I enabled 16 threads for bwa mem tool.

5. CONCLUSIONS

As mentioned in the Introduction, because of the prosperity of NGS technology and the rich information contained in NGS data from where there is the potential to find various type of genetic mutations, a large number of institutions, tools and workflows for sequencing information analysis came to market. GATK and its Best Practices from Broad Institute are among the best, especially the implementation which combines the latest version GATK 4 with the Cromwell execution engine, WDL scripts and Docker containers. Because of the advantages of faster speed, higher throughput and accuracy, the rich ecosystem and complete and uniform workflows which starts from DNA library preparation to final VCF collection, GATK and its Best Practices have become the industrial gold-standard for variant calling.

This project belongs to a pilot project hold by CSC and FIMM which aims at connecting FIMM´s genome sequencers directly to CSC´s computing platform and automatically processing NGS data analysis on CSC Pouta Cloud environment. Its tasks contain enabling pipeline implementation and performance benchmarking of GATK4 Best Practices on cPouta virtual machine. Moreover, most of the data will remain in Finland where using CSC services.

Institute of Molecular Medicine Finland (FIMM) focuses on human genomic research and personalized medicine. It owns several mainstream sequencers, such as NovaSeq, HiSeq and Miseq. FIMM also provided downstream analysis services. The biggest demanding is the variant calling workflow. FIMM wants complete automated NGS data analysis for variant calling workflows, from DNA library preparation to the final VCF generation. Accurate, automatic or even one-click completion. Currently, Broad Institute cooperates with Alibaba Cloud, Amazon Web Services, Intel, Cloudera, IBM, Microsoft Genomics and Google Cloud providing a similar automated analysis platform but there is no benchmarking data released for GATK4 and no cooperation with other cloud platforms.

cPouta IaaS Cloud is the main production IaaS Cloud provided by CSC which runs on OpenStack cloud software and can be connected to the external IP address enabling customers widely available service. It provides a large amount of computing resources for Finnish research institutions and universities for academic use. In addition, cPouta has other advantages, such as providing a programmable API and a web interface that enable users to easily generate, control and manage their virtual machines online; providing both compute nodes for HPC and more genetic computing load. cPouta environment links to other CSC computing environments, I.e., Taito and Sisu supercomputers which enables fast and barrier-free data transmission. The work presented in this thesis run on virtual machines generated on cPouta IaaS Cloud, with flavor hpc-gen2.24 core. The flavour featured Intel(R) Xeon(R) CPU E5-2680 v3, with hyper-threading, 24 VCPUs.

This thesis presented the detailed VM hardware setup, five variant calling pipelines codes and algorithm introduction, benchmark pipeline execution time(s) from single sample calling to joint calling pipelines with three WGS datasets: NA12878, NA12891 and NA12892 (except bwa-mem testing with 16 threads, other tools ran with default value which means single-threaded). I also explored optimized run-time parameters for GATK4 tools, PairHMM thread scalability in HaplotypeCaller, GATK4 thread scalability for PGC in MarkDuplicates and execution time(s) comparison for GATK4 SortSam vs SortSamSpark and MarkDuplicates vs MarkDuplicatesSpark. Results showed pipelines execution time(s) for similar WGS datasets with different size and features were quite close and execution time and dataset size were roughly positive correlated. The optimal threads number is 12 for GATK4 HaplotypeCaller with ERC mode which reduced the average execution time(s) of three replications for NA12878.hg38.bam -L chr20:1-25000000 from 26.7 minutes to 23.4 minutes (12.4% improvement). The optimal PGC threads number is 2 for GATK4 MarkDuplicates. Multi-threading with Spark local runner did highly speed up tool

SortSam(SortSamSpark) and MarkDuplicates(MarkDuplicatesSpark) execution. SortSamSpark enabled 16 local cores gave rise to a speed-up of 83.6%. MarkDuplicatesSpark enabled 16 local cores gave rise to a speed-up of 22.2% and 37.3%, if omit metrics file writing step via removing "`-M`" argument.

This thesis describes five separate pipelines that can be flexibly combined according to the different types of input files and analysis stages. Acceptable input file formats including: FASTQ, uBAM, and GVCF. This workflow is extensible for companies that need to apply this workflow to multiple data sets simultaneously. By deploying this workflow on a computer cluster, the respective sample analysis process can be processed on each node in parallel at the same time.

Optimizing the pipeline will always be an important issue. It is generally considered from two perspectives: improving hardware performance and optimizing software.

Multi-threading in PGC and PairHMM gives rise to very limited computing speed improvement. But GATK4 is not like GATK3 which can easily and directly enable multi-threading and multicore with two arguments. However, according to Broad Institute´s description, GATK4 is the newest version of GATK with completely rewrote core code, also with better accuracy and faster speed. Its parallel algorithm is mainly implemented by Spark, both local runner and Spark cluster runner. Spark cluster runner is a trend of development in the industry. At present a large number of GATK4 tools´ Spark versions still under development which are not suitable for production, as of the completion of this thesis. In the future, when Broad Institute developed the mature Spark version, it will be very useful to replace the normal version with the Spark version in the Best Practices and benchmarking performance. The Intel white paper for GATK Best Practices Pipeline Deployment [74] mentioned that faster storage and disk I/O significantly affected GATK3 tools performance when running multiple pipelines simultaneously. It is worth to test the GATK4 tools execution time(s) using SSD and HDD in the future.

The paper's GATK4 Best Practices benchmarking result can help customers predict sample analysis time and price and make more reasonable experimental arrangements. Parameter optimization evaluation of some tools in GATK4 can help sequencing analysis companies to improve pipelines, reduce time consumption and unit price of sample analysis and enhance market competitiveness.

REFERENCES

[1] Rantsiou K, Kathariou S, Winkler A, et al. Next generation microbiological risk assessment: opportunities of whole genome sequencing (WGS) for foodborne pathogen surveillance, source tracking and risk assessment. *International Journal of Food Microbiology* 2018; 287: 3–9.

[2] McGann P, Bunin JL, Snesrud E, et al. Real time application of whole genome sequencing for outbreak investigation – What is an achievable turnaround time? *Diagnostic Microbiology and Infectious Disease* 2016; 85: 277–282.

[3] Salipante SJ, SenGupta DJ, Cummings LA, et al. Application of Whole-Genome Sequencing for Bacterial Strain Typing in Molecular Epidemiology. *J Clin Microbiol* 2015; 53: 1072–1079.

[4] Kwong JC, McCallum N, Sintchenko V, et al. Whole genome sequencing in clinical and public health microbiology. *Pathology* 2015; 47: 199–210.

[5] Ormond KE, Wheeler MT, Hudgins L, et al. Challenges in the clinical application of whole-genome sequencing. *The Lancet* 2010; 375: 1749–1751.

[6] Shendure J. Accurate Multiplex Polony Sequencing of an Evolved Bacterial Genome. *Science* 2005; 309: 1728–1732.

[7] Margulies M, Egholm M, Altman WE, et al. Genome sequencing in microfabricated high-density picolitre reactors. *Nature* 2005; 437: 376–380.

[8] Mardis ER. A decade's perspective on DNA sequencing technology. *Nature* 2011; 470: 198–203.

[9] Goldberg B, Sichtig H, Geyer C, et al. Making the Leap from Research Laboratory to Clinic: Challenges and Opportunities for Next-Generation Sequencing in Infectious Disease Diagnostics. *mBio* 2015; 6: e01888-15.

[10] McKenna A, Hanna M, Banks E, et al. The Genome Analysis Toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Res* 2010; 20: 1297–1303.

[11] DePristo MA, Banks E, Poplin R, et al. A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nature Genetics* 2011; 43: 491–498.

[12] Merkel D. Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux J*; 2014, http://dl.acm.org/citation.cfm?id=2600239.2600241 (2014, accessed 18 February 2019).

[13] Zaharia M, Chowdhury M, Das T, et al. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. 2012; 2–2.

[14] Thibault J, Hartl C, Poplin R, et al. From FastQ Data to High-Confidence Variant Calls: The Genome Analysis Toolkit Best Practices Pipeline: The Genome Analysis Toolkit Best Practices Pipeline. In: *Current Protocols in Bioinformatics*, pp. 43(1):11–0.

[15] Sanger F, Coulson AR. A rapid method for determining sequences in DNA by primed synthesis with DNA polymerase. *Journal of Molecular Biology* 1975; 94: 441–448.

[16] Gilbert W, Maxam A. The Nucleotide Sequence of the lac Operator. *Proc Natl Acad Sci U S A* 1973; 70: 3581–3584.

[17] Sanger F, Air GM, Barrell BG, et al. Nucleotide sequence of bacteriophage φX174 DNA. *Nature* 1977; 265: 687.

[18] Staden R. A strategy of DNA sequencing employing computer programs. *Nucleic Acids Res* 1979; 6: 2601–2610.

[19] Gocayne J, Robinson DA, FitzGerald MG, et al. Primary structure of rat cardiac beta-adrenergic and muscarinic cholinergic receptors obtained by automated DNA sequence analysis: further evidence for a multigene family. *Proc Natl Acad Sci U S A* 1987; 84: 8296–8300.

[20] Smith LM, Sanders JZ, Kaiser RJ, et al. Fluorescence detection in automated DNA sequence analysis. *Nature* 1986; 321: 674–679.

[21] Connell C, Fung S, Heiner C, et al. *Automated DNA-sequence analysis. BioTechniques*. 1987.

[22] Alföldi J, Lindblad-Toh K. Comparative genomics as a tool to understand evolution and disease. *Genome Res* 2013; 23: 1063–1068.

[23] Roberts L. Timeline: A History of the Human Genome Project. *Science* 2001; 291: 1195–1200.

[24] Liu L, Li Y, Li S, et al. Comparison of Next-Generation Sequencing Systems. *BioMed Research International*; 2012.

[25] Illumina. NovaSeq 6000 Sequencing System Site Prep Guide. 2019.

[26] Bruce S D, Yakov R. *MPLS: Technology and Applications*. 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000.

[27] The International SNP Map Working Group. A map of human genome sequence variation containing 1.42 million single nucleotide polymorphisms. *Nature* 2001; 409: 928–933.

[28] Freeman JL, Perry GH, Feuk L, et al. Copy number variation: New insights in genome diversity. *Genome Res* 2006; 16: 949–961.

[29] Ophir R, Graur D. Patterns and rates of indel evolution in processed pseudogenes from humans and murids. *Gene* 1997; 205: 191–202.

[30] Redon R, Ishikawa S, Fitch KR, et al. Global variation in copy number in the human genome. *Nature* 2006; 444: 444–454.

[31] Stranger BE, Forrest MS, Dunning M, et al. Relative Impact of Nucleotide and Copy Number Variation on Gene Expression Phenotypes. *Science* 2007; 315: 848–853.

[32] de Smith AJ, Walters RG, Froguel P, et al. Human genes involved in copy number variation: mechanisms of origin, functional effects and implications for disease. *Cytogenet Genome Res* 2009; 123: 17–26.

[33] de Koning APJ, Gu W, Castoe TA, et al. Repetitive Elements May Comprise Over Two-Thirds of the Human Genome. *PLoS Genet* 2011; 7: e1002384.

[34] Cock PJA, Fields CJ, Goto N, et al. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Res* 2010; 38: 1767–1771.

[35] Li H, Handsaker B, Wysoker A, et al. The Sequence Alignment/Map format and SAMtools. *Bioinformatics* 2009; 25: 2078–2079.

[36] Fritz MH-Y, Leinonen R, Cochrane G, et al. Efficient storage of high throughput DNA sequencing data using reference-based compression. *Genome Res* 2011; 21: 734–740.

[37] Broad Institute. Picard Tools. *GitHub repository*, http://broadinstitute.github.io/picard/ (2018).

[38] Broad Institute. GATK Tool HaplotypeCaller. *gatk documentation*, https://software.broadinstitute.org/gatk/documentation/tooldocs/4.0.4.0/org_broadinstitute_hellbender_tools_walkers_haplotypecaller_HaplotypeCaller.php (2018).

[39] Broad Institute. GATK Tool GenotypeGVCFs. *gatk documentation*, https://software.broadinstitute.org/gatk/documentation/tooldocs/4.0.4.0/org_broadinstitute_hellbender_tools_walkers_GenotypeGVCFs.php (2018).

[40] The 1000 Genomes Project Consortium. A global reference for human genetic variation. *Nature* 2015; 526: 68–74.

[41] Danecek P, Auton A, Abecasis G, et al. The variant call format and VCFtools. *Bioinformatics* 2011; 27: 2156–2158.

[42] Robinson JT, Thorvaldsdóttir H, Winckler W, et al. Integrative genomics viewer. *Nature Biotechnology* 2011; 29: 24–26.

[43] Thorvaldsdóttir H, Robinson JT, Mesirov JP. Integrative Genomics Viewer (IGV): high-performance genomics data visualization and exploration. *Brief Bioinform* 2013; 14: 178–192.

[44] Challis D, Yu J, Evani US, et al. An integrative variant analysis suite for whole exome next-generation sequencing data. *BMC Bioinformatics* 2012; 13: 8.

[45] Liu X, Han S, Wang Z, et al. Variant Callers for Next-Generation Sequencing Data: A Comparison Study. *PLoS ONE* 2013; 8: e75619.

[46] Broad Institute. broadinstitute/gatk - Docker Hub, https://hub.docker.com/r/broadinstitute/gatk/ (2018).

[47] Broad Institute. GATK | Current version, https://software.broadinstitute.org/gatk/download/ (2018).

[48] Odersky M, Altherr P, Cremet V, et al. An Overview of the Scala Programming Language. *Technical Report LAMP-REPORT-2006-001* 2006; 20.

[49] Voss K, Gentry J, Van der Auwera G. Full-stack genomics pipelining with GATK4 + WDL + Cromwell. *F1000Research 2017*; 6.

[50] N PE, Mulerickal FJP, Paul B, et al. Evaluation of Docker containers based on hardware utilization. In: *2015 International Conference on Control Communication Computing India (ICCC)*. 2015, pp. 697–700.

[51] Felter W, Ferreira A, Rajamony R, et al. An updated performance comparison of virtual machines and Linux containers. In: *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2015, pp. 171–172.

[52] Salah T, Zemerly MJ, Yeun CY, et al. Performance comparison between container-based and VM-based services. In: *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*. 2017, pp. 185–190.

[53] Joy AM. Performance comparison between Linux containers and virtual machines. In: *2015 International Conference on Advances in Computer Engineering and Applications*. 2015, pp. 342–346.

[54] Zhang Q, Liu L, Pu C, et al. A Comparative Study of Containers and Virtual Machines in Big Data Environment. *arXiv:180701842*.

[55] Mishchenko D. *VMware ESXi: Planning, Implementation, and Security*. Cengage Learning, 2010.

[56] Watson J. VirtualBox: Bits and Bytes Masquerading As Machines. *Linux Journal* 2008; 2008(166): 1.

[57] Barham P, Dragovic B, Fraser K, et al. Xen and the Art of Virtualization. In: *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*. New York, NY, USA: ACM, pp. 164–177.

[58] Kivity A, Kamay Y, Laor D. kvm: the Linux Virtual Machine Monitor. 8.

[59] European Nucleotide Archive. Whole genome sequencing and variant calls for the Coriell CEPH/UTAH 1463 family. *European Bioinformatics Institute*, https://www.ebi.ac.uk/ena/data/view/PRJEB3381.

[60] Broad Institute. GATK | Tool Documentation | BaseRecalibrator, https://software.broadinstitute.org/gatk/documentation/tooldocs/4.0.4.0/org_broadinstitute_hellbender_tools_walkers_bqsr_BaseRecalibrator.php (2018).

[61] Alpaydin E. *Introduction to Machine Learning*. 2nd ed. The MIT Press, 2010.

[62] rpoplin. Base Quality Score Recalibration (BQSR) [GATK Forum]. https://gatkforums.broadinstitute.org/gatk/discussion/44/base-quality-score-recalibration-bqsr (2012 July).

[63] Thorisson GA, Smith AV, Krishnan L, et al. The International HapMap Project Web site. *Genome Res* 2005; 15: 1592–1593.

[64] Broad Institute. Google Cloud Bucket - genomics public data. https://console.cloud.google.com/storage/browser/genomics-public-data/resources/broad/hg38/v0 (2019).

[65] James Guilford, George Powley, Greg Tucker, et al. Accelerating the Compression and Decompression of Genomics Data using GKL Provided by Intel.

[66] Dagum L, Menon R. OpenMP: An Industry-Standard API for Shared-Memory Programming. *IEEE Comput Sci Eng* 1998; 5: 46–55.

[67] Broad Institute. GATK | Pipelining Options. https://software.broadinstitute.org/gatk/documentation/pipelines?v=3 (2018).

[68] Harrison PW, Alako B, Amid C, et al. The European Nucleotide Archive in 2018. *Nucleic Acids Res* 2019; 47: D84–D88.

[69] Broad Institute. BwaAndMarkDuplicatesPipelineSpark Duplicate Key -1 (GATK) v4.0.4.0 · Issue #4820 · broadinstitute/gatk. *GitHub*, https://github.com/broadinstitute/gatk/issues/4820 (2018).

[70] Tom White, James Emery. GATK | Blog |Spark Improvements in 4.1: Delivering results faster!, https://software.broadinstitute.org/gatk/blog?id=23420 (2019).

[71] shlee. Re: MarkDuplicateSpark is slower than normal MarkDuplicates [GATK Forum], https://gatkforums.broadinstitute.org/gatk/discussion/23441/markduplicatespark-is-slower-than-normal-markduplicates) (2019).

[72] Intel-HLS. GKL. *GitHub repository*, https://github.com/Intel-HLS/GKL (2018).

[73] Heldenbrand JR, Baheti S, Bockol MA, et al. Performance benchmarking of GATK3.8 and GATK4. Epub ahead of print 18 June 2018. DOI: 10.1101/348565.

[74] Prabhakaran A, Shifaw B, Naik M, et al. Infrastructure for Deploying GATK Best Practices Pipeline. 28.

APPENDIX A: ARGUMENTS FOR MAIN TOOLS

Arguments for main tools involved in described pipelines are listed in below Program1. (All bash variables needed for the command line have been set before).

```
   MarkDuplicates
   # Identifies and tags duplicate reads in a BAM or SAM file.
   --INPUT ${sep=' --INPUT ' input_bams} \
   --OUTPUT ${output_bam_basename}.bam \
   --METRICS_FILE ${metrics_filename} \
   --VALIDATION_STRINGENCY SILENT \
   --OPTICAL_DUPLICATE_PIXEL_DISTANCE 2500 \
   --ASSUME_SORT_ORDER "queryname" \
   --CREATE_MD5_FILE true

   SortSam
   # Sorts the input SAM or BAM file based on its records such as: coordinate and queryname (QNAME) then produces an index file.
   --INPUT ${input_bam} \
   --OUTPUT /dev/stdout \
   --SORT_ORDER "coordinate" \
   --CREATE_INDEX false \
   --CREATE_MD5_FILE false \

   BaseRecalibrator
   # Detects systematic errors in base quality scores with machine learning model.
   -R ${ref_fasta} \
   -I ${input_bam} \
   --use-original-qualities \
   -O ${recalibration_report_filename} \
   --known-sites ${dbSNP_vcf} \
   --known-sites ${sep=" --known-sites " known_indels_sites_VCFs} \
   -L ${sep=" -L " sequence_group_interval}

   ApplyBQSR
   # Applies base quality score recalibration.
   -R ${ref_fasta} \
   -I ${input_bam} \
   -O ${output_bam_basename}.bam \
   -L ${sep=" -L " sequence_group_interval} \
   -bqsr ${recalibration_report} \
   --static-quantized-quals 10 --static-quantized-quals 20 --static-quantized-quals 30 \
   --add-output-sam-program-record \
   --create-output-bam-md5 \
   --use-original-qualities

   HaplotypeCaller
   # Call germline SNPs and indels via local re-assembly of haplotypes in –ERC GVCF mode.
   -R ${ref_fasta} \
   -I ${input_bam} \
   -L ${interval_list} \
   -O ${output_filename} \
   -contamination ${default=0 contamination} ${true="-ERC GVCF" false="" make_gvcf}

   MergeVcfs
```

```
    # Combines multiple variant files into a single variant file.
    --INPUT ${sep=' --INPUT ' input_vcfs} \
    --OUTPUT ${output_filename}

    GenomicsDBImport
    # Import VCFs to GenomicsDB datastore.
    --genomicsdb-workspace-path ${workspace_dir_name} \
    --batch-size ${batch_size} \
    -L ${interval} \
    --sample-name-map inputs.list \
    --reader-threads 5 \
    -ip 500

    GenotypeGVCFs
    # Perform joint genotyping on three input GVCFs in the form of GenomicsDB workspace.
    -R ${ref_fasta} \
    -O ${output_vcf_filename} \
    -D ${dbsnp_vcf} \
    -G StandardAnnotation \
    --only-output-calls-starting-in-intervals \
    --use-new-qual-calculator \
    -V gendb://$WORKSPACE \
    -L ${interval}

    VariantRecalibrator \
    # Uses machine learning method to build a filtering model for Indel variants based on high-accuracy known variant sites
    -V ${sites_only_variant_filtered_vcf} \
    -O ${recalibration_filename} \
    --tranches-file ${tranches_filename} \
    --trust-all-polymorphic \
    -tranche ${sep=' -tranche ' recalibration_tranche_values} \
    -an ${sep=' -an ' recalibration_annotation_values} \
    -mode INDEL \
    --max-gaussians 4 \
    -resource                                    mills,known=false,train-
ing=true,truth=true,prior=12:${mills_resource_vcf} \
    -resource                                axiomPoly,known=false,train-
ing=true,truth=false,prior=10:${axiomPoly_resource_vcf} \
    -resource                                     dbsnp,known=true,train-
ing=false,truth=false,prior=2:${dbsnp_resource_vcf}

    VariantRecalibrator \
    # Uses machine learning method to build a filtering model for SNP variants based on high-accuracy known variant sites
    -V ${sites_only_variant_filtered_vcf} \
    -O ${recalibration_filename} \
    --tranches-file ${tranches_filename} \
    --trust-all-polymorphic \
    -tranche ${sep=' -tranche ' recalibration_tranche_values} \
    -an ${sep=' -an ' recalibration_annotation_values} \
    -mode SNP \
    --sample-every-Nth-variant ${downsampleFactor} \
    --output-model ${model_report_filename} \
    --max-gaussians 6 \
    -resource                                   hapmap,known=false,train-
ing=true,truth=true,prior=15:${hapmap_resource_vcf} \
```

```
114     -resource                                    omni,known=false,train-
    ing=true,truth=true,prior=12:${omni_resource_vcf} \
116     -resource                                  1000G,known=false,train-
    ing=true,truth=false,prior=10:${one_thousand_genomes_resource_vcf} \
118     -resource                                    dbsnp,known=true,train-
    ing=false,truth=false,prior=7:${dbsnp_resource_vcf}
120
        ApplyVQSR \
122     # Apply variant quality socre recalibration for Indel variants.
        -O tmp.indel.recalibrated.vcf \
124     -V ${input_vcf} \
        --recal-file ${indels_recalibration} \
126     --tranches-file ${indels_tranches} \
        --truth-sensitivity-filter-level ${indel_filter_level} \
128     --create-output-variant-index true \
        -mode INDEL
130
        ApplyVQSR \
132     # Apply variant quality socre recalibration for SNP variants.
        -O ${recalibrated_vcf_filename} \
134     -V tmp.indel.recalibrated.vcf \
        --recal-file ${snps_recalibration} \
136     --tranches-file ${snps_tranches} \
        --truth-sensitivity-filter-level ${snp_filter_level} \
138     --create-output-variant-index true \
        -mode SNP
```

Program 1. *Arguments for main tools.*